UNIVERSITY OF CALIFORNIA,
IRVINE

Simulation and Control of Biological Stochasticity

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Mathematics

by

Christopher Vincent Rackauckas

Dissertation Committee:
Chancellor's Professor Qing Nie, Chair
Associate Professor German Enciso
Professor Long Chen

2018

DEDICATION

To my fiancée Diana Navarrete. Finding her was and is my best discovery.
To my mother and father, Judy and Andy Rackauckas, for their love and patience.
To Tara Rackauckas for always finding the pawsitive parts of life. Woof.
To the late Jesse Rowsell. May his influence and cheer live on.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I would like to thank Dr. Qing Nie. Throughout my graduate studies he has been a source of guidance and inspiration.

I am grateful to the Schilling lab for taking the time to help teach me biological wet lab work. Dr. Julian Sosnik, Dr. Tom Schilling, Dr. Arul Subramanian, Dr. Praveer Sharma, and Diego Hoyle were all great colleagues and friends who me bridge into the world of biology.

I would like to express my gratitude to all of the professors associated with the Center for Complex Biological Systems (CCBS), including Dr. Arthur Lander, Dr. German Enciso, and Dr. Jun Allard. It has been an enormous pleasure to have taken part in the interdisciplinary community throughout these years.

I am thankful to all of the members of the Nie lab who helped me throughout my graduate career, including Dr. Dongyong Wang, Dr. Likun Zheng, Dr. Tian Hong, Dr. Weitao Chen, Dr. Catherine Ta, Dr. Seth Figueroa, Dr. Adam MacLean, and Dr. Lina Meinecke.

Lastly, I would like to thank the Julia community, including all of the contributors to the JuliaDiffEq organization, to their hard work, dedication, and support. Dr. Viral Shah, Dr. Stefan Karpinski, Dr. Gabriel Gellner, Dr. Sheehan Olver, Dr. Vijay Ivaturi, Dr. Simon Byrne, Lyndon White, @dextorious, Yingbo Ma, Scott Jones, Dr. Tom Short, David Widmann, Dr. Hendrik Ranocha, Dr. Mauro Werder, Mike Innes, Tom Breloff, Dr. David Sanders, and the mysterious Cormullion have all been a large positive influence.

The text of this thesis/dissertation contains reprints of the material as it appears in journals. Chapter 2 was done in collaboration with Thomas Schilling and Qing Nie and was published in [93]. Chapter 4 was done in collaboration with Qing Nie and was published in [91]. Chapter 5 was done in collaboration with Qing Nie and was published in [92].

This work was partially supported by NIH grants R01GM107264 and P50GM76516 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 NIH Award T32 EB009418. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the NIH.

CURRICULUM VITAE

Christopher Vincent Rackauckas

EDUCATION

Doctor of Philosophy in Mathematics	**2018**
University of California, Irvine	*Irvine, California*
Masters in Mathematics	**2015**
University of California, Irvine	*Irvine, California*
Bachelor of Science in Mathematics	**2013**
Oberlin College	*Oberlin, Ohio*

RESEARCH EXPERIENCE

Graduate Research Assistant and Fellow	**2013–2018**
University of California, Irvine	*Irvine, California*
Undergraduate Research Assistant	**2012–2013**
Oberlin College	*Oberlin, Ohio*

TEACHING EXPERIENCE

Google Summer of Code Administrator and Mentor	**2016–2018**
Julialang Google Summer of Code	*N/A*
Data Science Instructor	**2016–2018**
University of California, Irvine Data Science Initiative	*Irvine, California*
Teaching Assistant	**2014–2015**
University of California, Irvine	*Irvine, California*

FELLOWSHIPS, SCHOLARSHIPS, AND AWARDS

Tsukuba Global Science Week Best Speaker Award	**2017**
Data Science Initiative Summer Fellowship	**2016**
XSEDE Allocation DMS160004	**2016**
Center for Complex Biological Systems Opportunity Award	**2015**
National Science Foundation Graduate Research Fellowship	**2014**
Ford Foundation Predoctoral Fellowship	**2014**
National Institutes of Health T32 Predoctoral Training Grant	**2014**
Margaret C. Etter Student Lecturer Award	**2013**
Graduate Deans Recruitment Fellowship	**2013**
Mathematical and Computational Biology (MCB) Fellowship	**2013**
National Science Foundation S-STEM Scholarship	**2010**
John F. Oberlin Scholarship	**2009**

REFEREED JOURNAL PUBLICATIONS

Mean-Independent Noise Control of Cell Fates via Intermediate States **2018**
iScience

DifferentialEquations.jl - A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia **2017**
Journal of Open Research Software

Adaptive Methods for Stochastic Differential Equations via Natural Embeddings and Rejection Sampling with Memory **2017**
Discrete and Continuous Dynamical Systems Series B

Noise modulation in retinoic acid signaling sharpens segmental boundaries of gene expression in the embryonic zebrafish hindbrain **2018**
eLife

On The Budyko-Sellers Energy Balance Climate Model with Ice Line Coupling **2015**
Discrete and Continuous Dynamical Systems Series B

Assessment of Statistical Methods for Water Quality Monitoring in Maryland's Tidal Waterways **2013**
SIAM Undergraduate Research Online

SOFTWARE

DifferentialEquations.jl `github.com/JuliaDiffEq/DifferentialEquations.jl`
Julia suite for high-performance solvers of differential equations.

ABSTRACT OF THE DISSERTATION

Simulation and Control of Biological Stochasticity

By

Christopher Vincent Rackauckas

Doctor of Philosophy in Mathematics

University of California, Irvine, 2018

Chancellor's Professor Qing Nie, Chair

Stochastic models of biochemical interactions elucidate essential properties of the network which are not accessible to deterministic modeling. In this thesis it is described how a network motif, the proportional-reversibility interaction with active intermediate states, gives rise to the ability for the variance of biochemical signals to be controlled without changing the mean, a property designated as mean-independent noise control (MINC). This noise control is demonstrated to be essential for macro-scale biological processes via spatial models of the zebrafish hindbrain boundary sharpening. Additionally, the ability to deduce noise origin from the aggregate noise properties is shown.

However, these large-scale stochastic models of developmental processes required significant advances in the methodology and tooling for solving stochastic differential equations. Two improvements to stochastic integration methods, an efficient method for time stepping adaptivity on high order stochastic Runge-Kutta methods termed Rejection Sampling with Memory (RSwM) and optimal-stability stochastic Runge-Kutta methods, are combined to give over 1000 times speedups on biological models over previously used methodologies. In addition, a new software for solving differential equations in the Julia programming language is detailed. Its unique features for handling complex biological models, along with its high performance (routinely benchmarking as faster than classic C++ and Fortran integrators of similar implementations) and new methods, give rise to an accessible tool for simulation of large-scale stochastic biological models.

Chapter 1

Introduction

Biochemical signals are inherently noisy. The process of diffusion and every binding/unbinding event is the product of billions of molecular interactions which give rise to stochastic phenomenon. While it may seem at face value that information is lost when modeling or observing the aggregate stochastic system, the resulting stochastic observations contain clues indicative of the generating processes. This relationship between the biochemical interaction network and the collective observations can in turn be utilized to both identify the network and control its properties.

In Chapter 2, the relationship between the observed variance in zebrafish retinoic acid measurements and the underlying gene regulatory network is elucidated. By analyzing the properties of the core parts of the interaction network, we identify how proportional-reversibility and active intermediate states naturally give rise to a mechanism for mean-independent variance control (MINC). It is shown how this motif can give rise to the noise control property in many different network architectures, thus giving a robust relation between an aggregate stochastic property and the network structure. Three key properties of this relationship are demonstrated:

1. The ability for the MINC mechanism to control macroscale phonotypes is shown by demonstrating its ability to stabilize the boundary sharpening mechanism in the zebrafish hindbrain rhombomeres 4/5.

2. The relation between noise properties and aggregate developmental processes is revealed by identifying a region of noise quantity for which proper hindbrain development can occur. It is shown and explained how substantial decreases or increases in the amount of noise break down the ability for cell switching mechanisms to accurately detect signaling boundaries.

3. The ability for stochastic measurements to identify the network properties and noise origin are demonstrated.

Together, these identified features show that details in the stochastic nature of biochemical interactions are essential for understanding macro-scale biological processes.

However, this study uncovered the inadequacy of current stochastic modeling methods and software. Fully discrete modeling tools like Gillespie SSA and tau-leaping are designed for small-scale simulation and are not computationally sufficient to handle the long time concentration-scale spatial models. While the approach of using continuous approximations via stochastic differential equations (such as the Chemical Langevin Equation) and stochastic partial differential equations gives rise to a simplified mathematical structure which can be exploited for efficient computation while retaining the key properties of the stochastic system, major developments in the methodology and tooling were required in order to handle the size and unique features of these biochemical models. In Chapter 3 a method for efficient adaptive time stepping in high order stochastic Runge-Kutta methods, termed Rejection Sampling with Memory (RSwM), is developed. In Chapter 4, new adaptive high order Runge-Kutta methods with optimal stability are derived. It is demonstrated that these new methods are able to solve equations modeling biochemical networks over 1,000

times faster than the traditional methods built into Problem Solving Environments (PSEs) like MATLAB or toolboxes like SDETools. Lastly, Chapter 5 describes a software suite, DifferentialEquations.jl, for solving a large array of differential equation types using the Julia programming language. The structure of this software is shown to uniquely be able to handle complex biological models by allowing abstract number and array handling, dynamic resizing, and mixing of Poisson processes with the differential equations. Benchmarks of this software suite routinely demonstrate an order of magnitude speedup against standard C++ and Fortran ODE solvers on stiff and non-stiff test problems, and over two orders of magnitude against comparable software suites in PSEs like R, MATLAB, and Python. Together, these new methodologies in the new software suite aggregate to several orders of magnitude speedup which brings previously incomputable models of biological properties to the realm of computability.

Chapter 2

Mean-Independent Noise Control of Cell Fates via Intermediate States

This chapter was published as [93]. It details the mean-independent noise control (MINC) property of the proportional-reversibility motif with active intermediate states and shows the downstream effects of noise control on macro-scale biological processes. The computational requirements imposed by the spatial simulations in this chapter were the impetus for the creation of software for efficient stability-optimized high order adaptive integrators for stochastic differential equations, which are the focus of Chapter 3, 4, and 5.

2.1 Summary

Stochasticity affects accurate signal detection and robust generation of correct cell fates. While many known regulatory mechanisms may reduce fluctuations in signals, most simultaneously influence their mean dynamics, leading to unfaithful cell fates. Through analysis and computation, we demonstrate that a reversible signaling mechanism acting through intermediate states can reduce noise while maintaining the mean. This mean-independent noise control (MINC) mechanism is investigated in the context of an intracellular binding protein that regulates retinoic acid (RA) signaling during zebrafish hindbrain development. By comparing our models with experimental data, we find that a MINC mechanism allows for sharp boundaries of gene expression without sacrificing boundary accuracy. In addition, this MINC mechanism can modulate noise to levels that we show are beneficial to spatial patterning through noise-induced cell fate switching. These results reveal a design principle that may be important for noise regulation in many systems that control cell fate determination.

2.2 Introduction

Stochasticity is prevalent in cell signaling networks, yet organisms manage to determine cell fates and coordinate their development in response to signals robustly in spite of this noise. Stochasticity has been directly observed in gene expression [29, 108] and identified as a cause of cell-to-cell variation [77, 94]. The amount of stochasticity in gene expression can change qualitative features of the system, such as introducing bistability in a deterministically monostable system and vice versa [60], or by allowing switching between stable states [84, 66, 129, 128]. Noise can have adverse effects by distorting downstream signals [33] and by disrupting entrainment of biochemical oscillators [39], suggesting the existence of noise control mechanisms to ensure robustness. However, these mechanisms are not well understood.

There is increasing evidence that such systems of noise control exist in signaling pathways. Differences in regulatory network architecture correlate with both the amplification and attenuation of expression noise [22, 85], and increased complexity of a biochemical network has been shown to correlate with reduced noise [18]. For one autoregulatory protein it has been suggested that negative feedback decreases system noise [115]. In addition, increased growth rates in a single- celled organism (yeast) can increase noise in gene expression [57], and studies in Drosophila suggest that these principles apply to noise in spatial signals in a multicellular (albeit syncytial) context [38]. Multiple binding sites for the Bicoid morphogen in the Hunchback promotor buffer spatial noise in the Bicoid morphogen gradient [47]. In addition, specific noise levels are advantageous for establishing population heterogeneity [57], and noise can facilitate sharp segmental boundaries of gene expression in the zebrafish hindbrain, but only within a limited range of levels of signaling noise [131]. It has become increasingly clear that noise not only needs to be attenuated but also that the appropriate levels of noise are critical for accurate cellular responses to a signal.

One example of noise regulation occurs in the developing vertebrate hindbrain, which is patterned by a retinoic acid (RA) morphogen gradient. RA is a well-known signaling molecule [80] that controls the formation of hindbrain segments [109], as well as patterning and differentiation of many other cell types and tissues. Models of morphogen gradients suggest that they can cause all-or-none regulatory responses [78, 124], even in the presence of stochasticity, and consistent with this RA signaling specifies segmental patterns of hindbrain gene expression despite substantial levels of noise [101]. We used fluorescence lifetime imaging microscopy (FLIM) to measure stochasticity in the RA gradient and directly demonstrated the existence of large fluctuations in the gradient [111]. One of the four cellular RA-binding proteins (Crabp2a) in zebrafish is essential for robust patterning in the developing hindbrain [16]. Experiments increasing or decreasing Crabp2a levels show that it acts to attenuate noise in RA while leaving the mean unchanged [111], pointing to the existence of a mechanism that allows for the desired noise levels to be achieved without shifting the gradient.

Using the RA signaling network in zebrafish hindbrain development as an example, we investigate design principles for controlling noise in the signal without affecting mean levels. We find that a coupling of reversible reactions gives rise to mean-independent noise control (MINC), and this coupling naturally arises in complex systems through the presence of an intermediate state. In the example of spatial patterning of hindbrain segments, a MINC mechanism involving Crabp2a enables mean-independent sharpening of gene expression boundaries in the correct locations in response to RA. In addition, we find that the degree to which mean and variance are coupled distinguishes between different noise origins, and when analyzed with the FLIM data obtained from zebrafish embryos, our results suggest that the dominant noise source is endogenous to the RA signaling pathway.

2.3 Results

2.3.1 Proportional-Reversibility Enables Mean-Independent Noise Control

First, we considered a simplification of the differential equation model [101] to capture the essential qualitative features. In the retinoic acid (RA) signaling pathway, extracellular RA enters the cell and binds to an intermediate (CRABP) which shuttles it to the nucleus to bind to a receptor (RAR) and form a compound (RA-RAR) which binds to the DNA to both signal downstream targets and produce a protein (CYP) which in turn inactivates RA (depicted in Figure 2.1A). The basic interaction (denoted as the Simple Model, SM) can be modeled as a two-state stochastic differential equation (SDE):

$$\begin{aligned} d\,[RA] &= (\beta + \delta\,[RAR] - (\gamma + \eta)\,[RA])\,dt + \sigma dW_t, \qquad (2.1) \\ d\,[RAR] &= (\gamma\,[RA] - \delta\,[RAR])\,dt, \end{aligned}$$

where the deterministic portion of the equation is due to mass-action laws and the additional

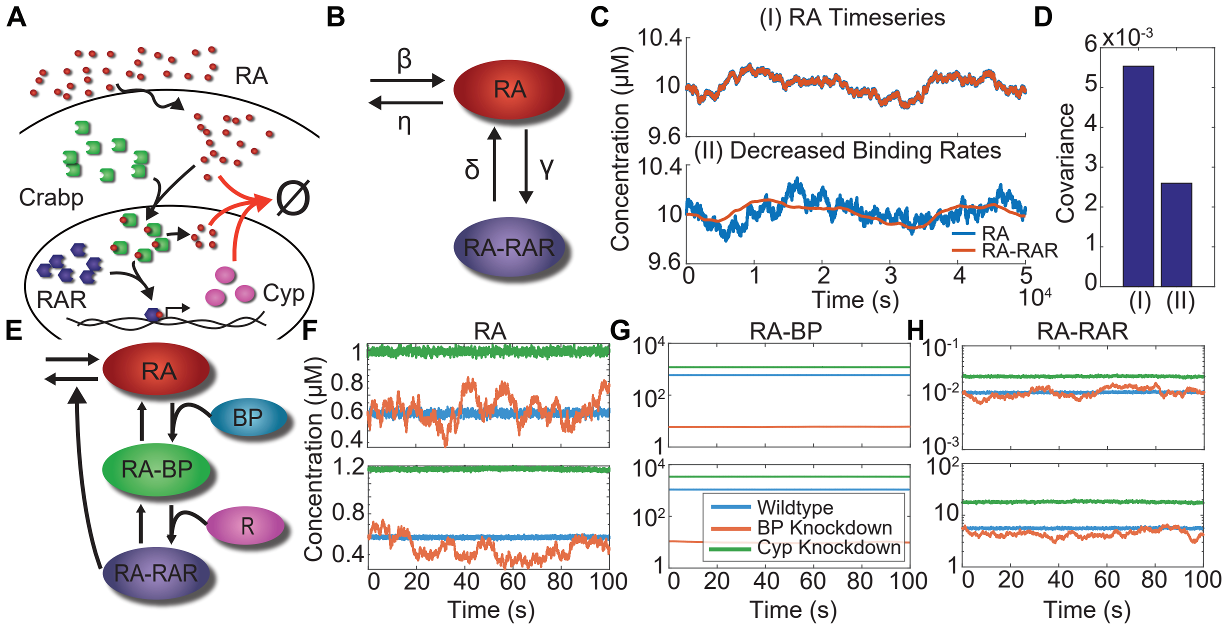


Figure 2.1: **The Proportional-Reversibility Strategy for Noise Attenuation. (A)** A diagram of the retinoic acid (RA) signaling network. RA (shown in red) enters the cell to bind with the cellular retinoic acid binding proteins (CRABP, shown in green), which shuttles RA into the nucleus. There the RA binds with its receptor (RAR, shown in purple) to produce Cyp26 (shown in pink). The Cyp26 proteins in turn deactivate the RA signaling proteins. **(B)** A schematic depiction of the two-state simplified model (SM) of the RA network. The red node in the graph is the concentration of RA, and the blue node is RA bound to its receptor (RA-RAR). Birth and death of RA is allowed, along with reversible binding/unbinding with RAR. **(C)** Representative time-series solutions to SM. These were obtained using the Euler-Maruyama method. (I) shows the solution using parameters from Table 2.1 while (II) uses the same parameters with γ and δ decreased by a factor of 0.005. Both trajectories start from the expected value of 10. Notice that in (I) the blue line is mostly covered by the orange line. **(D)** The covariances from experiments (I) and (II). **(E)** A schematic depiction of the full RA model RM. The red node depicts the concentration of RA, which binds with the binding protein (BP, shown in teal) to form the RA-BP intermediate complex (shown in green). From there, the RA binds to its receptor RAR (shown in pink) to form RA-RAR (shown in purple). This causes the transcription of Cyp26a1, which degrades RA. **(F,G,H)** Representative time-series solutions to RM. The parameters for the model were chosen randomly according to the method described in Section 2.5.9. The resulting stochastic differential equations were solved using the Euler-Maruyama method for 100 seconds to give the blue line for the wild-type value. The value γ was reduced by 90% and the simulation was re-solved to give the orange line. The value for γ was reset, and the value for α was reduced by 90% , and the simulation was re-solved to give the green line.

term (σdW_t) describes stochasticity in the production and degradation of RA (schematized in Figure 2.1B). The steady-state mean values are

$$\begin{aligned} E\left[RA\right] &= \frac{\beta}{\eta}, \\ E\left[RAR\right] &= \frac{\beta\gamma}{\delta\eta}, \end{aligned} \tag{2.2}$$

and the steady-state variances are

$$\begin{aligned} Var\left[RA\right] &= \frac{\left(\delta+\eta\right)\sigma^2}{2\eta\left(\gamma+\delta+\eta\right)}, \\ Var\left[RAR\right] &= \frac{\gamma^2\sigma^2}{2\delta\eta\left(\gamma+\delta+\eta\right)}. \end{aligned} \tag{2.3}$$

(see Section 2.5.1 for details of the derivation). Assume that the rates for the reversible binding and unbinding of the morphogen to its receptor are coupled via a constant C :

$$\delta = C\gamma. \tag{2.4}$$

Under this coupling assumption, the steady-state values become proportional,

$$E\left[RA\right] = \frac{\beta}{\eta}, \qquad (2.5)$$
$$E\left[RAR\right] = \frac{\beta}{C\eta} = \frac{E\left[RA\right]}{C},$$

with the mean concentrations determined solely by the production rate β and decay rate η of RA. However, under the same conditions,

$$Var\left[RA\right] = \frac{\left(C\gamma + \eta\right)\sigma^2}{2(1+C)\gamma\eta + 2\eta^2} \qquad (2.6)$$

the variance directly depends on γ , the rate of conversion from RA to RA-RAR. Note that increasing γ attenuates noise in the RA concentration (derived in Section 2.5.1). Thus, with this coupling assumption, changing the reaction rates of RA binding and dissociation from its receptor has no effect on steady-state mean concentrations but has a direct and directional effect on their variances. Comparing the temporal dynamics of the system to a setup with reduced binding rates suggests that this mean-independent variance effect is due to changes in the binding rates (Figure 2.1C).

The mean amounts of the proteins are left unchanged by the coupled reaction rates since the amount of RA that leaves its unbound state increases by the same amount that leaves its bound state due to the coupling assumption. To intuit why the variance decreases, we compute

$$Cov\left(\left[RA\right], \left[RAR\right]\right) = \frac{\gamma\sigma^2}{2\eta\left(\left(1+C\right)\gamma + \eta\right)}, \qquad (2.7)$$

which is an increasing function in γ . Thus when the binding and unbinding rates are higher, the concentrations of RA and RA-RAR tend to be in sync (Figure 2.1D). Therefore if $[RA]$ is above its steady state and the binding/unbinding rates are high, then it is highly likely that $[RA - RAR]$ will also be above its steady state levels. One can heuristically understand that when random fluctuations cause an excess of morphogen, the natural force towards steady state will drive the excess morphogen toward degradation. If $[RA]$ is below its steady state levels but $[RA - RAR]$ is even lower, the system could push some of the morphogen to its receptor-bound state even if the total morphogen in the system is lower than its equilibrium value, thus decreasing the total pull towards steady state. Therefore the total pull towards equilibrium is greatest when the correlation is highest, which explains why the variance decreases and saturates to a constant as $\gamma \to \infty$ leads to near perfect correlation. This intuition suggests that this feature is a property of the coupling of the reversibility. To see if this extends to more complex systems, we note that $[RA - RAR]$ enhances the production of a protein Cyp26a1, which in turn degrades $[RA]$ [125, 126]. We show that when incorporating this nonlinearity into the previous model, the essential feature of mean-independence from the reversible reaction rates holds under the same coupling assumptions (see Section 2.5.3). Additionally, the mean-independence, along with the noise attenuating and increase in covariance due to γ, holds for the general master equation formulation of the model (see Section 2.5.2).

The previous results rely on the assumption that the binding and unbinding rates are proportional. However, next we asked if this coupling naturally occurs in the presence of an intermediate state. The RA signaling network includes an intermediate state RA-BP where RA is bound to its binding protein Crabp2a (BP) which shuttles RA to the nucleus where it binds to its receptor to form RA-RAR [16]. Adding this interaction to the previous model gives the Intermediate Model (IM). Notice that in this model the flux into $[RA - RAR]$ at steady state is $\nu\,[RA - BP]_{ss}$ while the internal flux back to $[RA]$ is $\delta\,[RA - BP]_{ss}$ .Therefore, near steady state, $[RA - BP]$ is a natural coupling constant between the influx of RA and the influx of RA-RAR, suggesting a generalization of the proportional-reversibility mechanism. The heuristic derivation is confirmed since one can show that a proportional coupling of the rate from $[RA]$ to $[RA - BP]$ and the rate from $[RA - RAR]$ to $[RA - BP]$ produces the same mean-independence and variance-dependence on the coupled reaction rates behavior as in the previous model, and that $E\left[RA - BP\right]$ is proportional to the proportionality constant (see 2.5.4 for details). Additionally, we can extend the signaling pathway by adding an intermediate unbinding in the nucleus step or model Crabp2as shuttling of RA to Cyp26a1 as a separate pool of RA and in this extended model the MINC property still holds (see Section 2.5.5).

We build upon this initial analysis to generate a more mechanistic model that fully incorporates the BP and receptor concentrations, which we refer to as the Retinoic Acid Model (RM) (Figure 2.1E). In this model we derive the same natural coupling via $E\left[RA - BP\right]$, but also show that $E\left[RA - BP\right]$ is proportional to $E\left[BP\right]$, which in turn are directly determined by the production and degradation rates of BP (see Section 2.5.6 for details of the derivation). Therefore production and degradation rates of BP have no effect on the mean amounts of $[RA]$ and $[RA - RAR]$ while directly affecting variance. Comparisons of the concentration of $[RA]$, $[RA - BP]$ and $[RA - RAR]$, respectively, for two separate trajectories of the system show that the means of $[RA]$ and $[RAR]$ are unchanged by perturbations in $[BP]$ but variance changes (Figures 1F-H). Perturbations in [Cyp] change the mean as well as the variance, but the latter to a much lesser extent.

Lastly, we note the experimental evidence that RA-RAR upregulates the production of Crabp2a but no other Crabps [16, 101]. To simulate the effect of RA-RAR signaling on Crabp production, along with other possibly indirect downstream effects such as Eph and Ephrins on the RA signaling pathway [122], we introduce the Retinoic Acid Model with Feedback (RMF) by adding upregulation of BP by $[RA - RAR]$ to RM. We derive that the mean-independent variance control via a coupling related to $E\left[BP\right]$ still holds in the RMF model (see Section 2.5.7). This shows that by incorporating the dynamics of an intermediate state in the model we find a natural and controllable coupling of mean reaction rates that allows for the amount of intermediate to attenuate the system noise without changing mean levels of the signal.

2.3.2 Active Intermediate States Naturally Lead to Proportional-Reversibility Control

The previous results were determined by linearizations of stochastic models around steady-state values. In order to establish the mean-independent noise attenuation property directly on nonlinear complex models, we performed simulations. First, we introduced a Mean-Variance Dependence Index (MVDI),

$$\zeta = \frac{\%\Delta Variance}{\%\Delta Mean + \%\Delta Variance}, \tag{2.8}$$

to encapsulate the relation of mean and variance changes into a single non-dimensionalized value. If the variance changes but the mean does not, ζ is exactly 1. If the mean changes but the variance does not, then ζ is 0. Thus the MVDI ζ offers a measure of the degree of isolation of the mean relative to variance changes with respect to a perturbation. Four separate models of the RA signaling pathway (SM, IM, RM, and RMF) were simulated in order to determine the robustness of the mean-independent noise attenuation property to the network topology for RA alone (Figure 2.2A-D) or bound to RAR (Figure 2.2E-H). Each model was simulated using random reaction rates 100,000 times and ζ s calculated from reductions in BP and Cyp (Table 2.1). Over 85% of the simulations resulted in a $\zeta > 0.9$ with a decrease in BP, indicating that for most parameter sets the relative change

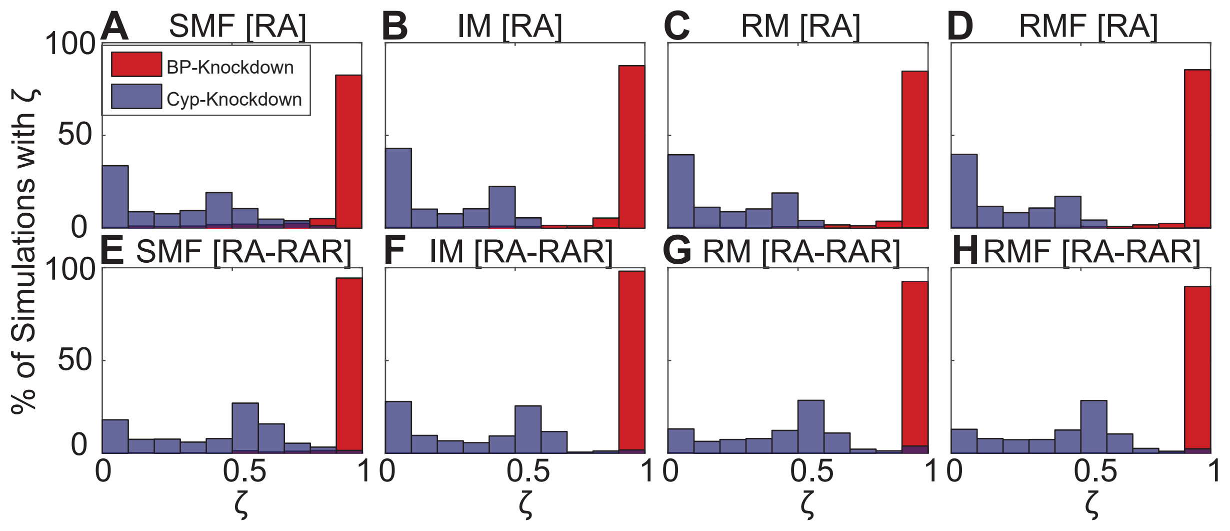


Figure 2.2: **Mean and Variance Knockdown Distributions. (A)-(H)** Histograms depicting the ζ distribution due to additive noise. The models were solved using the Euler-Maruyama method on the timespan of $t = [0, 200]$, and re-solved with a 90% knockdown to the associated parameter. The value ζ was calculated from these two series, and this was repeated 100,000 times. More details outlining the experiments are found in Section 2.5.9. **(A)-(D)** ζ 's calculated for $[RA]$ on the respective models. **(E)-(H)** ζ 's calculated for $[RA - RAR]$ on the respective models.

in variance was much larger than the relative change in mean. In addition, when the same computational experiment was applied with multiplicative noise, over 90% of the simulations resulted in a $\zeta > 0.9$ with BP knockdown (Figure 2.3). We note that from the experimental data [111] we estimate that the true ζ after depleting BP is $\zeta = 0.97$. This shows that in almost all of the trajectories in each model, the variance changes with only minor changes in the mean, confirming that proportional-reversibility and intermediate states naturally give rise to mean-independent noise control. Additionally, over 90% of simulations result in $\zeta < 0.6$ with reductions of Cyp under both additive and multiplicative noise, showing that this property is specific to manipulation of BP levels.

Parameter	$baseExp_p$
β	2
a	0
b	1
α	2
u	1
ζ	0
η	1
r	1
All others	3

Table 2.1: **Numerical Knockdown Experiment Base Exponents.** Each parameter was chosen by taking x_p uniformly from [-2,2] and calculating $p = 10^{-x_p - baseExp_p}$. On the left is the parameter and on the right is the corresponding $baseExp_p$ value. All values not listed in the table had the value $baseExp_p = 3$.

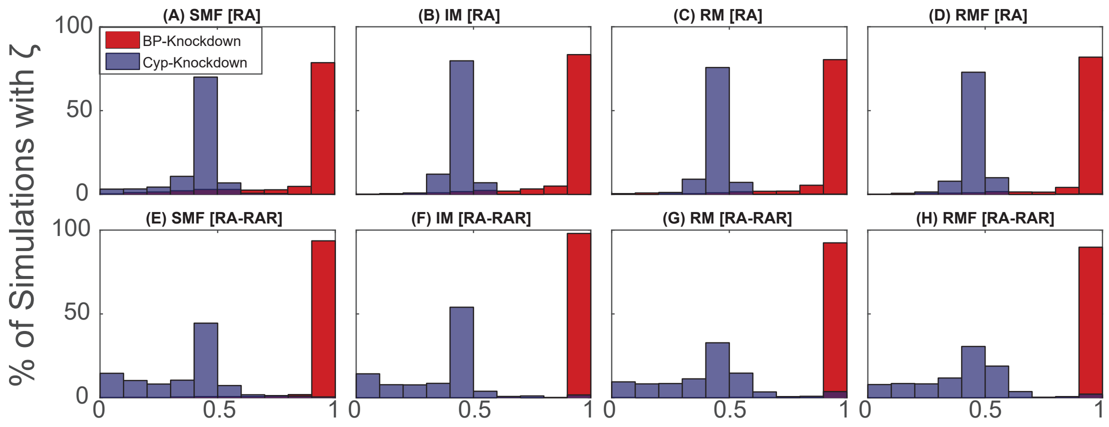


Figure 2.3: **Mean and Variance Knockdown Distributions with Multiplicative.** **(A)-(H)** Histograms depicting the ζ distribution due to multiplicative noise. The models were solved using the Euler-Maruyama method on the timespan of $t = [0, 200]$, and re-solved with a 90% knockdown to the associated parameter. The value ζ was calculated from these two series, and this was repeated 100,000 times. More details outlining the experiments are found in Section 2.5.9. **(A)-(D)** ζ 's calculated for $[RA]$ with the respective models. **(E)-(H)** ζ 's calculated for $[RA - RAR]$ with the respective models.

2.3.3 Spatial MINC Enables Accurate Specification of Landmark Locations

We hypothesize that the mean-independent variance attenuation property of BP should locally smooth spatial gradients. To study the MINC property in spatial signaling, we first extended RMF to a two-dimensional stochastic reaction-diffusion system with space-time white noise where the extracellular RA, RA_{out} , diffuses throughout space. This model, RMFS, is thus given by a stochastic partial differential equation (SPDE) (defined in Section 2.5.8). As counterparts to the mean and variance in space, we measured the mean and variance of the location where the gradient hits specific values. To determine if the same mean/variance relationship is associated with these properties of the morphogen gradient, we simulated BP depletion experiments on the RMFS model with random parameters (Figure 2.5 and detailed in Section 2.5.10). Gradients of $[RA]_{in}$ and $[RA-RAR]$ exhibit less noise in wildtype than after BP depletion (Figure 2.4A-D). Given that the average size of cells in the zebrafish hindbrain is around $10\mu m$, our simulations suggest that changes in the amount of BP change the sharpness of the RA signaling boundary without changing its mean location by more than one cell diameter over 90% of the tested parameters (Figure 2.4E-F). In addition, the mean shift in the RA-RAR gradient with decreased Cyp shifts the threshold approximately 4 cell diameters, showing that spatial MINC is a special feature that does not extend to other interactions (Figure 2.7B).

The receptor-bound RA signal induces expression of Hoxb1a and Krox20 which mutually repress each other [131], which we include downstream in the RMFS model (depicted in Figure 2.7A). By preferentially upregulating Krox20, the boundary between rhombomeres 4 and 5 (r4/5) is established at a threshold where the RA gradient is sufficiently high enough for the initial Hoxb1a expression to be replaced by Krox20. Stochasticity in the Hox-Krox interaction allows for the initial r4/5 boundary to sharpen in the wildtype organism (Figures 4A-D). In addition, reducing BP disrupts the sharpening of the r4/5 boundary without

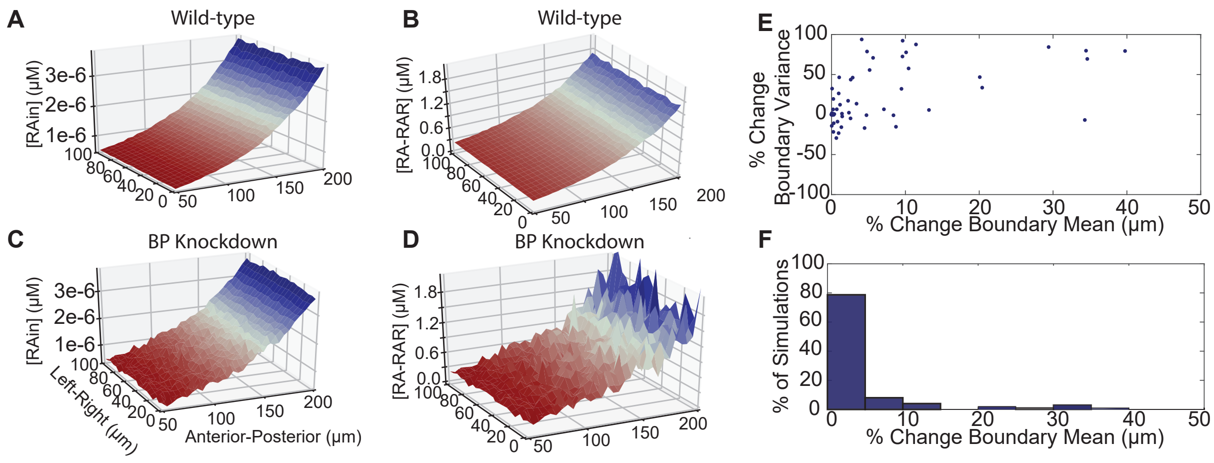


Figure 2.4: **Location-Independent Boundary Sharpening.** **(A)-(D)** Representative solution of RMFS. Shown are the two-dimensional gradients of intracellular $[RA_{in}]$ (plots **A** and **C**) and $[RA - RAR]$ (plots **B** and **D**). Random parameters were chosen according to Section 2.5.10. The x-axis runs from the anterior to the posterior of the zebrafish hindbrain. The simulations were run to determine the steady-state gradients, and then were run for 100 more seconds to give a snapshot of the spatial stochasticity. The top row depicts the wild-type results for a given parameter set, and the bottom row depicts the results when simulated again but with the parameter for binding protein (BP) reduced by 90% . The color levels are fixed between the wild-type and BP-deficient plots to provide accurate comparisons. **(E)** Scatter plot of the percent change in variance versus the change in the average boundary location (μm). 100 parameter sets were chosen and for each one the threshold concentration was taken to be 60% of the maximum value. The mean and the variance of the boundary location was calculated, the simulation was solved once more using the BP-deficient value, and the mean and variance of the boundary location were calculated using the same threshold. **(F)** Histogram of the number of simulations with a given change in mean boundary location (μm).

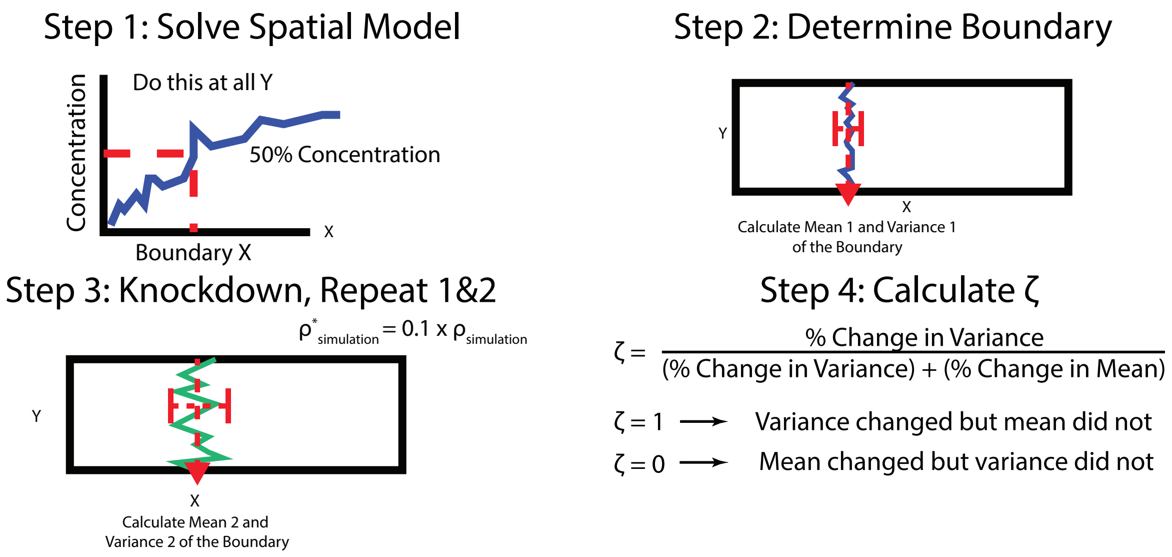

Step 1: Solve Spatial Model
Do this at all Y
Concentration
50% Concentration
X
Boundary X
Step 2: Determine Boundary
Y
X
Calculate Mean 1 and Variance 1 of the Boundary
Step 3: Knockdown, Repeat 1&2
$\rho^*_{simulation} = 0.1 \times \rho_{simulation}$
Y
X
Calculate Mean 2 and Variance 2 of the Boundary
Step 4: Calculate ζ
$\zeta = \frac{\%\text{ Change in Variance}}{(\%\text{ Change in Variance}) + (\%\text{ Change in Mean})}$
$\zeta = 1 \longrightarrow$ Variance changed but mean did not
$\zeta = 0 \longrightarrow$ Mean changed but variance did not

Figure 2.5: **Location-Independent Boundary Sharpening Experimental Diagram.** A diagram of the numerical scheme for the ζ histogram experiments for spatial location versus threshold sharpness. The SPDE model is solved and the 50% concentration point at each location along the x-axis is found. Then the mean and the variance of these x locations are saved. This process is then repeated with reduced binding protein and the resulting mean and variance values are compared with the previous to get a value for ζ.

moving its location, while reducing Cyp causes a shift in the segmental boundary position. Over time the number of predominantly Hox-expressing cells displaced in the wild-type saturates to approximately 3, while with depletion of BP the number of displaced cells saturates to approximately 10 (Figure 2.6E). On average a stochastic trajectory of the wild-type has a maximal displacement of predominantly Hox-expressing cells by 1-2 cell diameters, whereas with reduced BP there is a maximal displacement of predominantly Hox-expressing cells of around 4 cell diameters. Segmental sharpening in terms of a previously-defined sharpening index (SI) from [131] reveals a similar disruption of the sharpening mechanism under this measurement (Figure 2.7C). Together these results show that the loss of BP disrupts downstream segmental sharpening, consistent with the previous in vivo experimental observations in zebrafish [111].

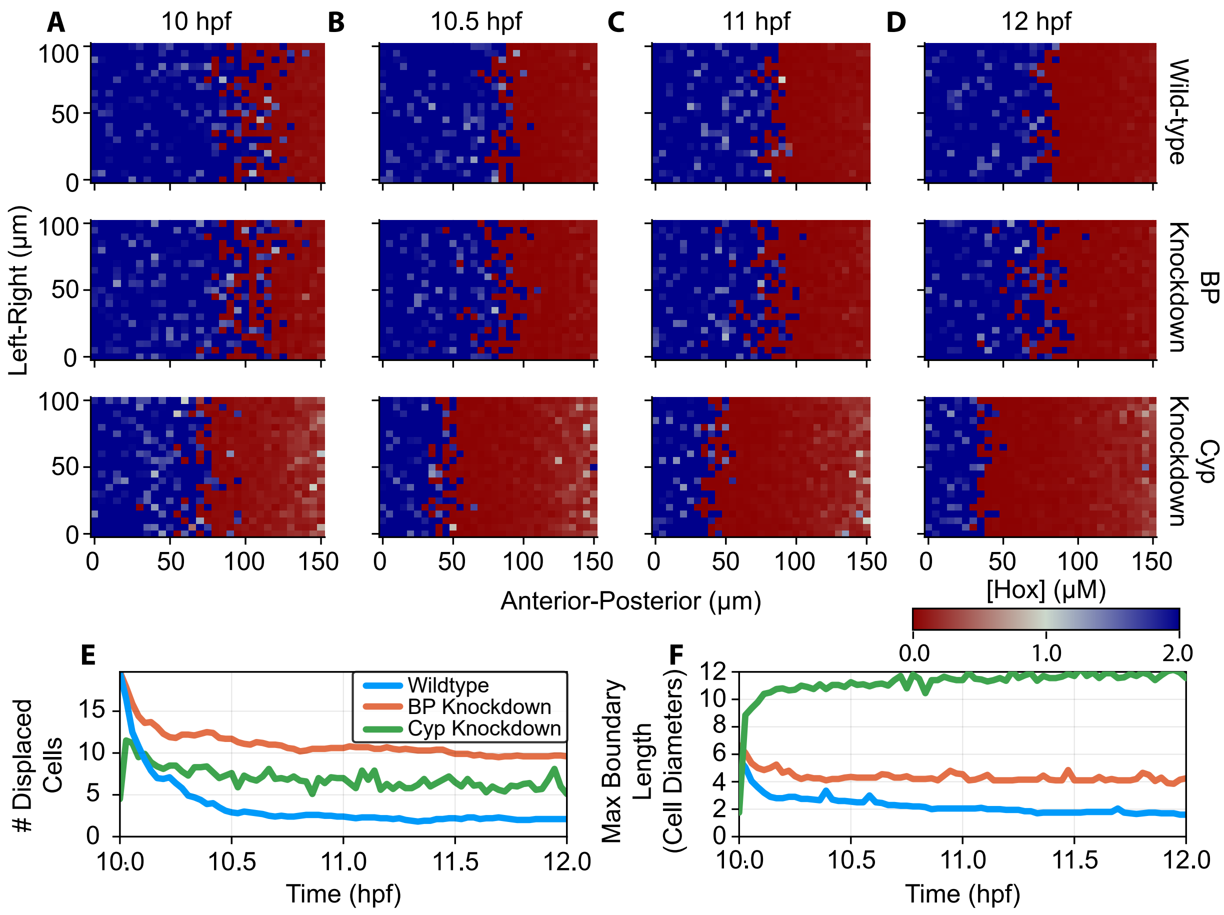
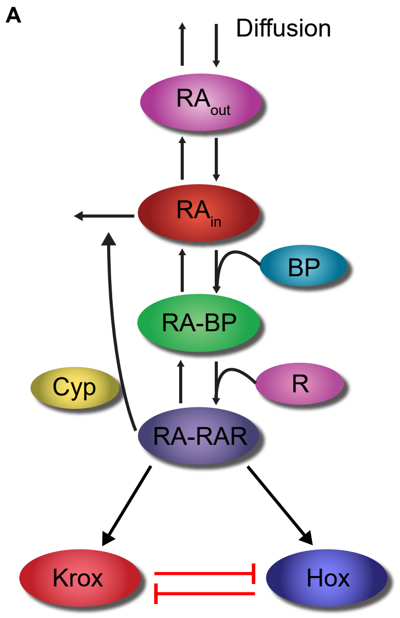
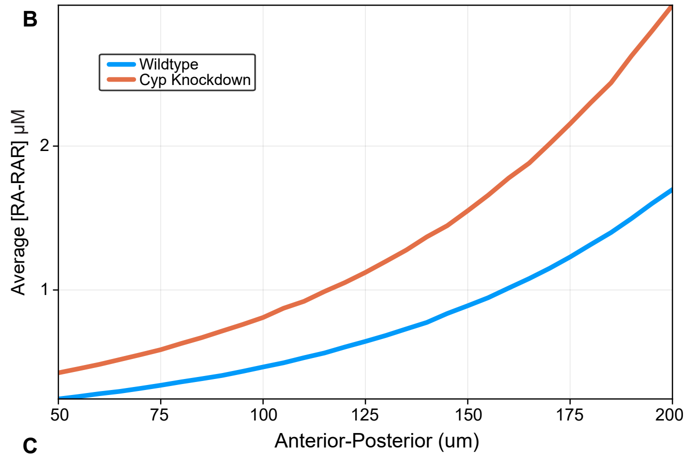


Figure 2.6: **Disruption of Downstream Boundary Sharpening Due to Perturbations in Binding Protein (BP) and Cyp26 Concentration. (A)-(D)** Representations of the rhombomere 4/5 segmental boundary at 10 hours post-fertilization (hpf), 10.5 hpf, 11 hpf, and 12 hpf respectively. The trajectories were calculated according to the RMFS model with the Hox-Krox extension (described in 2.5.11). The color corresponds to the concentration of Hox at a particular region in the Anterior-Posterior versus the Left-Right plane. The top panels correspond to the wild-type model, the middle panels are the models with reduced BP, and the bottom panels are the models with reduced Cyp26. **(E)** Number of displaced cells over time. The y-axis corresponds to the number of displaced cells, calculated as a predominantly Hox-expressing cell lying one cell length posterior to a predominantly Krox-expressing cell. The x-axis shows time in terms of hpf. Each condition was run 10 times and the results were averaged. **(F)** Maximum displacement over time. The y-axis corresponds to the maximum displacement, calculated as the maximum distance between a predominantly Hox-expressing cell that is posterior to a predominantly Krox-expressing cell. The axis is in cell diameters which correspond to 10 μm . The x-axis shows time in terms of hpf. Each condition was run 10 times and the results were averaged.





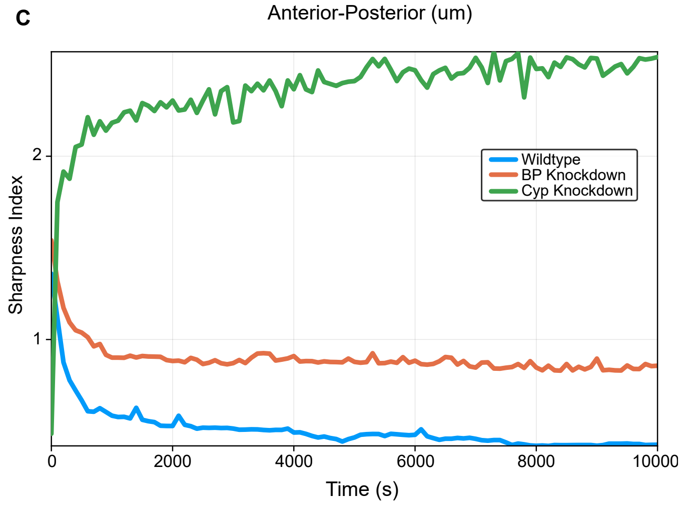


Figure 2.7: **Boundary Sharpening Disruption Extended Figures.** **(A)** Diagram of the extended retinoic acid (RA) model with downstream Hox-Krox signaling. The model starts with diffusive RA_{out} entering the cell to become RA_{in} and binding to BP to become $RA-BP$, which then binds RAR to produce $RA-RAR$. This induces Cyp which deactivates (and thus degrades) the intracellular RA. Additionally, $RA - RAR$ acts as a signal to the downstream and transcription factors, which are mutually antagonistic. **(B)** Mean shift of the RAR signal due to Cyp knockdown. For the wildtype and Cyp setups in the extended RA model with Hox-Krox, the mean of the gradient was calculated between each of the 10 runs at the ending timepoint. **(C)** Sharpness Index. The y-axis corresponds to the sharpness index defined in [131]. The x-axis shows time in terms of hours postfertilization (hpf) in zebrafish. Each condition was repeated 10 times and the results were averaged.

Parameter	Value
$\sigma_{RA_{in}}$,σ_{RA-RAR},$\sigma_{RA_{out}}$	$0.03\mu m/s$
b	$0.017\mu m/s$
α	$10{,}000\mu m/s$
β_0	$1\mu m/s$
c	$0.1\mu m/s$
ω	$100\mu m$
γ	$3.0\mu m/s$
δ	$0.0013\mu m/s$
η	$0.0001\mu m/s$
r	$0.0001\mu m/s$
ν	$0.85\mu m/s$
λ	$0.85\mu m/s$
u	$0.01\mu m/s$
d	$0.1\mu m/s$
e	$1\mu m$
a	$1\mu m/s$
ζ	$0.02\mu m/s$
c_h	$7.5\mu m/s$
c_k	$3.0\mu m/s$
k_h	$0.4\mu m/s$
k_k	$4.0\mu m/s$
d_h, d_k	$0.4\mu m/s$
a_h, a_k	$0.2\mu m/s$
D	$250.46\mu m/s$

Table 2.2: **Disruption of Downstream Boundary Sharpening Parameters.** Parameters correspond to RMFS with Hox-Krox interactions.

2.3.4 Noise Levels are Regulators for Patterning and Indicative of the Sources of Stochasticity

To further investigate the significance of noise levels in RA signaling, we next investigated requirements for an optimal range of noise levels in gene expressions, and we explored the relationship between the noise level and the origins of the stochasticity. To establish a direct relationship between the noise levels controlled via BP and successful segmentation of the zebrafish hindbrain, we defined the effective noise in the $[RA - RAR]$ signal as a measure of the variance of the gradient (described in Section 2.5.11). We simulated the full spatial model RMFS with varied [BP] and Hox-Krox signaling noise to determine if noise levels affected the ability to sharpen the r4/5 boundary. A successful sharpening event was defined as having less than or equal to 3 cells displaced by more than one cell diameter. All of the successful sharpening events had Hox-Krox noise levels in the range $(0.175 - 0.275)$ with an upper limit on the effective $[RA - RAR]$ signaling noise of approximately 10^{-3} (Figure 2.8A). Similar qualitative results were obtained when successful sharpening events were defined instead in terms of a threshold on mean displacement and maximal displacement (Figure 2.9). This shows that an optimal range for the effective noise is required for segmental patterning to occur, indicating the necessity of noise control mechanisms for properly regulating downstream signals.

ζ calculations for changes in BP do not distinguish between common noise types such as multiplicative or additive noise, but the probability distribution of ζ with respect to changes in the amount of Cyp strongly depends on the choice of the noise term (compare Figure 2.2 with Figure 2.3). Therefore, we investigated the validity of particular noise terms by comparing these probability distributions to experimental data. We examined data collected from free intracellular RA in the zebrafish hindbrain morphogen gradients with a morpholino knockdown of Cyp26a1 [111]. The methodology for the analysis (see Section 2.5.12) estimates the average experimental value as $\zeta \approx 0.62$.

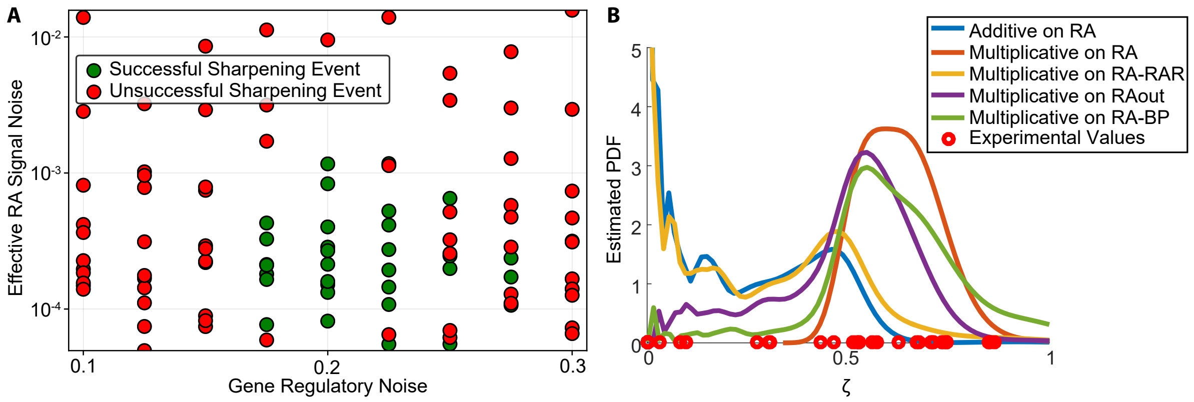
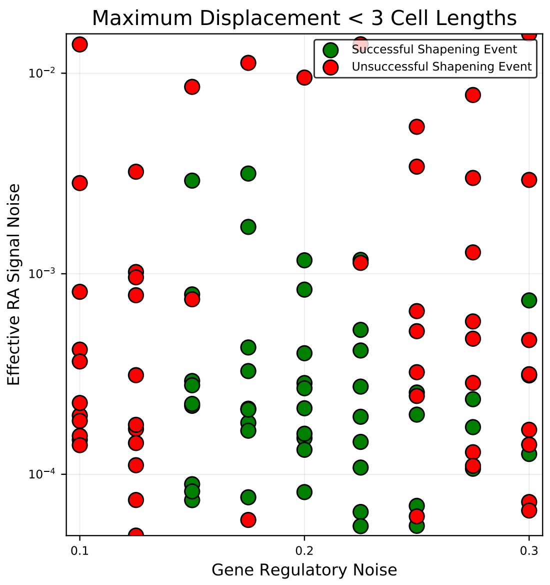


Figure 2.8: **Noise Levels Distinguish Between Models and Developmental Phenotypes.** **(A)** Scatter plot of the successful and unsuccessful sharpening events. BP production was taken as 5, 15, . . . , 105 and Hox-Krox regulatory noise was taken as 0.1, 0.125, . . . , 0.3 and each pairing in the grid was solved once. The effective RA noise was calculated according to the measure from Section 2.5.11 and a sharpening event was declared successful if after 2 hours less than 3 cells were displaced by more than a one cell diameter. **(B)** Depiction of the probability distribution for ζ according to the parameter search scheme from the parameter search scheme on RMF. The simulations which produced these distributions are discussed in Section 2.5.9. Shown are the kernel density estimates from the ζ values from the stochastic simulations. The different colored lines show the distributions for different noise types. The red circles depict experimental values for ζ computed pairwise.

To determine the likelihood that this system uses one type of noise versus another, we utilized the same parameter search scheme as previously described to calculate probability distribution for ζ with different noise types and depletion of Cyp on the model RMF (Figure 2.8B). An experimental ζ was calculated pairwise between each wild-type and Cyp-deficient embryo since there does not exist a canonical pairing, thus giving a distribution of 27 experimental ζ values which center around $\zeta = 0.6$ with a tail towards zero. ζ was less than the mean experimental ζ for $> 90\%$ of the parameters when the noise was additive for $[RA]_{in}$ or multiplicative for $[RA - RAR]$. In contrast, the ζ distributions with multiplicative noise for $[RA]_{in}$, $[RA - BP]$, and $[RA]_{out}$ all peak around the mean ζ value. Similar results were obtained for the cumulative distribution of ζ (Figure 2.10). Because of this reliance of the noise on the levels of $[RA]_{in}$ itself (all of these concentrations are directly linked), this strongly suggests that the dominant form of noise in the zebrafish hindbrain signaling network is intrinsic to stochastic processes related to $[RA]_{in}$ and establishes that exogenous



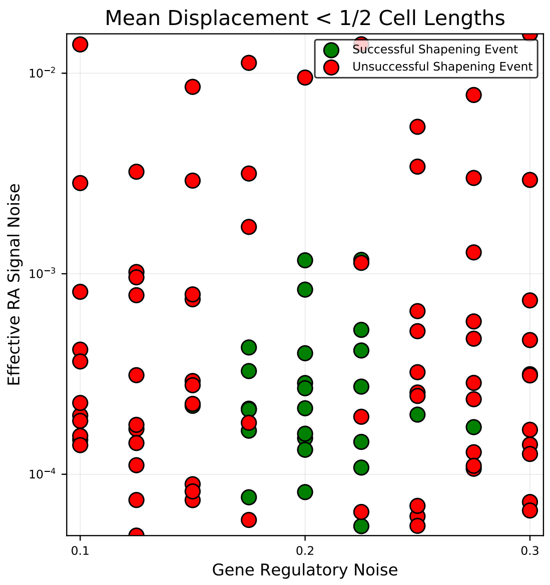


Figure 2.9: **Characterization of Successful Sharpening via Additional Measures.** Binding protein production taken as 5, 15, ... , 105 and Hox-Krox regulatory noise was taken as 0.1, 0.125, ... , 0.3 and each pairing in the grid was solved once. The effective RA noise was calculated according to the measure from Section 2.5.11. In **(A)** a successful sharpening event was characterized by having the maximum displacement between Hox and Krox dominated cells of less than 3 cell diameters. In **(B)** a successful sharpening event was characterized by having the mean displacement between Hox and Krox dominated cells as less than half of a cell diameter.

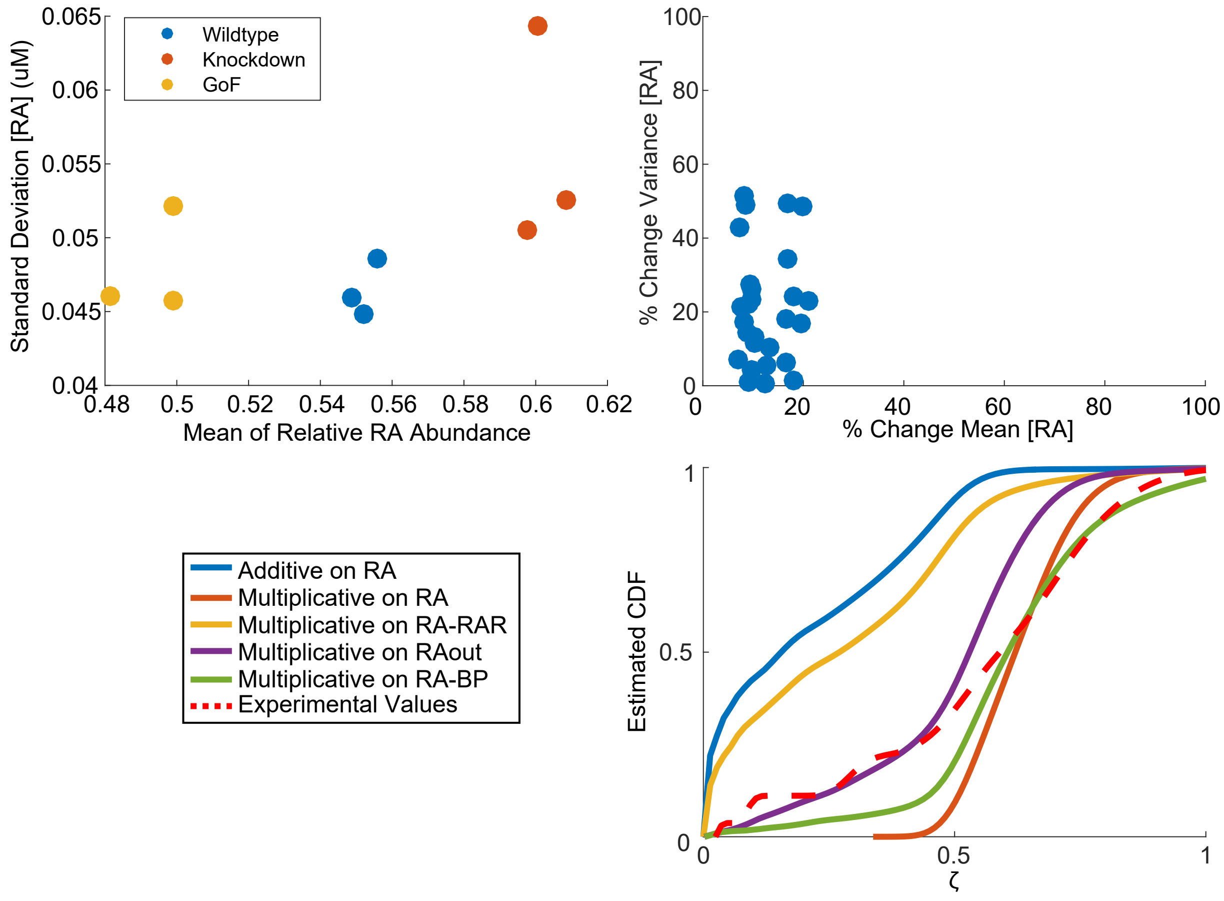


Figure 2.10: **Characterization of ζ applied to FLIM Data. (A)** Scatter plot of the data points from [111]. **(B)** Pairwise Differences. The percent change in mean and variance was calculated pairwise between each pair of higher and lower data points. A scatter plot of the results is shown. **(C)** Estimated CDF. Depiction of the commutative probability distribution for ζ according to the parameter search scheme from the parameter search scheme on RMF. The simulations that produced these distributions are discussed in Section 2.5.12. Shown are the kernel density estimates from the ζ values from the stochastic simulations. The different colored lines show the distributions for different noise types. The red line depicts experimental CDF for ζ computed using the pairwise data points.

noise is not likely to fit the data.

2.4 Discussion

Stochasticity in gene regulatory networks naturally exists [29], and noise attenuation or control is needed to enable proper biological functions. For example, in the zebrafish hindbrain, noise regulation is required for subsequent boundary sharpening processes to occur properly [131]. Here we uncover a mean-independent noise control (MINC) mechanism that can tune the level of noise in the downstream components of a gene regulatory network (GRN), without affecting the mean of the signal. In the zebrafish hindbrain system, this MINC mechanism provides a way (through Crabp2a) to achieve the required noise levels in the RA morphogen without disturbing other aspects of its spatial gradient. Together, we directly link the preservation of a stochastic spatial phenotype to a noise control mechanism, demonstrating a potential path through which developmental processes could have evolved to overcome inherent biochemical stochasticity in order to achieve robust spatial patterning.

The robustness in the choice of biologically-reasonable parameters needed to achieve such a mechanism indicate that it is an intrinsic property of the GRN topology. The core feature underlying this is a coupling assumption that arises naturally with the existence of intermediate states under mass action assumptions. Furthermore, we obtain similar results with five separate models, all with this same coupling. This suggests that MINC may be a general phenomenon related to the existence of intermediate states and probably exists in other GRNs. Computational simulations of epithelial-mesenchymal transitions (EMT) recently has shown that increased numbers of intermediate states attenuates noise in cellular fate decisions [114], which is analogous to our predictions of how pooling in the intermediate state reduces noise. Primed lineages in hematopoietic stem cells can be represented as an intermediate state with reversible changes, which could explain data showing mean-independent noise attenuation due to a lineage commitment factor [118]. However, we note that the spatial control results were only tested in the areas of the RA gradient where the concentration is sufficiently high. There may be other factors required for robust noise control and boundary sharpening in anterior rhombomeres (r1-3) where the RA concentration is low and the gradient is shallow. Additionally, our models do not take into account the effects of cell proliferation. While the cell cycle rapidly increases to 4 hours around the time of boundary sharpening [61] which is a much slower time scale than the noise processes, further research could better quantify the effects of cell divisions on the spatial noise.

Furthermore, our methods uncover a novel relationship between the noise source, the network topology, and the relationship between the mean and the variance using the perturbation data in experiments. Our analysis suggests intrinsic noise due to RA as the most likely dominant noise source in the zebrafish hindbrain given the known GRN topology and mean-variance relationship. With the increasing precision in experimental quantification of variance changes, this methodology could be used to identify noise sources and provide further evidence for/against GRN topologies. For example, microfluidic measurements have accurately measured noise dynamics in individual aging yeast cells [73] and a similar dynamic analysis to the one shown here could restrain the possible GRNs by requiring that not only the mean but also the noise dynamics match the data. Most importantly, this approach may be used to distinguish between models that have similar qualitative behavior with respect to the mean, thereby providing a new way to uncover details of biochemical networks from the noise in gene knockdown experiments.

Software and Algorithms	Source
MATLAB 2015b	The MathWorks Inc. 2015
Simulations made in MATLAB	Github: ChrisRackauckas/MINC
Julia	Github: Julialang/julia
DifferentialEquations.jl	JuliaDiffEq/DifferentialEquations.jl
Simulations made in Julia with DifferentialEquations.jl	Github: ChrisRackauckas/MINC
Plots.jl	Github: JuliaPlots/Plots.jl

Table 2.3: **Associated Software.** Software used for the numerical simulations.

2.5 Extended Information

2.5.1 Method Details

Associated Software

The associated software is described in Table 2.3.

Steady State Analysis

For the SODE $dX_t = f\left(X_t\right)dt + g\left(X_t\right)dW_t$, we calculated the mean of X_t using a linearization of the drift term (f) and solving for the unique positive steady state. To calculate the variance, we used the linearization of the Fluctuation-Dissipation Theorem where for the Jacobian of the drift at the steady-state $J\left(X_{ss}\right)$, we have that

$$J\left(X_{ss}\right)\Sigma\left(X_{ss}\right)+\Sigma\left(X_{ss}\right)J^{T}\left(X_{SS}\right)=-g^{2}\left(X_{ss}\right), \tag{2.9}$$

where $\Sigma\left(X_{SS}\right)$ is the covariance matrix at the steady state X_{ss} , and thus its diagonal value in column i gives the variance of the ith component when near the steady state. These computations were performed using Mathematica.

Monotonicity of Variance

Take the variance equation

$$Var\left[RA\right] = \frac{\left(C\gamma + \eta\right)\sigma^2}{2\left(1+C\right)\gamma\eta + 2\eta^2}. \tag{2.10}$$

Notice that

$$\frac{dVar\left[RA\right]}{d\gamma} = \frac{C\sigma^2}{2\eta\left(\gamma + \gamma C + \eta\right)} - \frac{\left(C+1\right)\sigma^{2\left(\gamma C + \eta\right)}}{2\eta\left(\gamma + \gamma C + \eta\right)^2}. \tag{2.11}$$

From Mathematica we see that $\frac{dVar[RA]}{d\gamma} = 0$ if and only if $\sigma = 0$. Therefore $Var\left[RA\right]$ is monotonic in γ . To see that it increases, we used the Mathematica Solve function to attempt to find values for which the derivative was negative. Mathematica could find no parameter regime where this was the case. As verification, we used the Mathematica Solve function to find the values for which the derivative was positive. The function returned no constraints, indicating that this always holds.

2.5.2 Mathematical Models and Steady-State Results

General Master Equation (SM)

The SM model can be written in the general master equation framework as:

$$\frac{\partial p}{\partial t} = (E_1 - 1)\,\eta n p + \left(E_1^{-1} E_2 - 1\right) m\delta p + \left(E_1 E_2^{-1} - 1\right) n\gamma p + \left(E_1^{-1} - 1\right) \beta p, \tag{2.12}$$

where $p = p\,(n, m; t)$ with n being the number of RA particles and m being the number of RA-RAR particles, and E_i being the step operators ($Ef\,(n) = f\,(n+1)$, implying $(E_1 - 1)$ is the annihilation of RA) for RA and RA-RAR respectively. Following [123, 1], we write the general master equation in the form:

$$\frac{\partial p}{\partial t} = \sum_{l_1, l_2} \left(E_1^{l_1} E_2^{l_2} - 1\right) \left[\Omega_{n \to n - l_1, m \to m - l_2}\, p\right], \tag{2.13}$$

where $\Omega_{n \to n - l_1, m \to m - l_2}$ is a reaction rate for the operation of losing l_1 RA and l_2 RA-RAR, and calculate the identities via algebraic manipulation:

$$\frac{d\langle n\rangle}{dt} = -\sum_{l_1,l_2}\langle l_1\Omega_{n\to n-l_1,m\to m-l_2}\rangle, \tag{2.14}$$

$$\frac{d\langle m\rangle}{dt} = -\sum_{l_1,l_2}\langle l_2\Omega_{n\to n-l_1,m\to m-l_2}\rangle, \tag{2.15}$$

$$\frac{d\langle n^2\rangle}{dt} = -\sum_{l_1,l_2}\langle l_1\left(l_1-2n\right)\Omega_{n\to n-l_1,m\to m-l_2}\rangle, \tag{2.16}$$

$$\frac{d\langle m^2\rangle}{dt} = -\sum_{l_1,l_2}\langle l_2\left(l_2-2m\right)\Omega_{n\to n-l_1,m\to m-l_2}\rangle, \tag{2.17}$$

$$\frac{d\langle nm\rangle}{dt} = -\sum_{l_1,l_2}\langle\left(l_1n+l_2m-l_1l_2\right)\Omega_{n\to n-l_1,m\to m-l_2}\rangle. \tag{2.18}$$

(For example: multiply by n , then re-define n to be shifted by l_1 and simplify. The others follow from similar manipulations). This gives the system of ODEs:

$$\frac{d\langle n\rangle}{dt} = -\left(\eta+\gamma\right)\langle n\rangle+\delta\langle m\rangle, \tag{2.19}$$

$$\frac{d\langle m\rangle}{dt} = \gamma\langle n\rangle-\delta\langle m\rangle, \tag{2.20}$$

$$\frac{d\langle n^2\rangle}{dt} = -\left(\eta+\gamma\right)\langle n\left(1-2n\right)\rangle-\delta\langle m\left(1+2n\right)\rangle-\beta\langle 1+2n\rangle, \tag{2.21}$$

$$\frac{d\langle m^2\rangle}{dt} = -\delta\langle m\left(1-2m\right)\rangle-\gamma\langle n\left(1+2m\right)\rangle, \tag{2.22}$$

$$\frac{d\langle nm\rangle}{dt} = -\eta\langle n^2\rangle+\delta\langle m\left(-n+m-1\right)\rangle+\gamma\langle n\left(n-m-1\right)\rangle+\beta\langle n\rangle. \tag{2.23}$$

Setting the derivatives to zero, we receive the steady-state values (calculations in the Mathematica notebooks):

$$E\left[RA\right] = \frac{\beta}{\eta}, \tag{2.24}$$

$$Var\left[RA\right] = \frac{\beta\left(\gamma+\eta+c\eta\right)}{\eta\left(\gamma+\eta+2c\eta\right)}, \tag{2.25}$$

$$Cov\left(\left[RA\right],\left[RA-RAR\right]\right) = -\frac{\beta\left(\gamma+\eta\right)}{\gamma\left(\gamma+\eta+2\eta c\right)}. \tag{2.26}$$

We note that the derivatives of the variance and covariance equations by γ are non-zero for all positive parameter values. Thus both equations are increasing functions of γ .

2.5.3 Simple Model with Feedback (SMF)

$$d\left[RA\right] = \left(\beta+\delta\left[RA-RAR\right]-\left(\gamma+\eta+\frac{\alpha\left[RA-RAR\right]}{\omega+\left[RA-RAR\right]}\right)\left[RA\right]\right)dt \tag{2.27}$$

$$+\sigma dW_t, \tag{2.28}$$

$$d\left[RA-RAR\right] = \left(\gamma\left[RA\right]-\delta\left[RA-RAR\right]\right)dt, \tag{2.29}$$

$$E\left[RA\right] = \frac{\sqrt{2\beta C\omega\left(2\alpha+\eta\right)+\beta^2+C^2\eta^2\omega^2}+\beta-C\eta\omega}{2\left(\alpha+\eta\right)}, \tag{2.30}$$

$$E\left[RA-RAR\right] = \frac{\sqrt{4\beta c\omega\left(\alpha+\eta\right)+\left(\beta-c\eta\omega\right)^2}+\beta-c\eta\omega}{2c\left(\alpha+\eta\right)}, \tag{2.31}$$

$Var\left[RA\right]:$

$$\frac{\sigma^2\left(E\left[RA\right]+C\omega\right)\left(E\left[RA\right]^2\left(\alpha+\eta\right)+2E\left[RA\right]C\omega\left(\alpha+\eta\right)+C\gamma\left(E\left[RA\right]+C\omega\right)^2+C^2\eta\omega^2\right)}{2\left(E\left[RA\right]^2\left(\alpha+\eta\right)+2E\left[RA\right]C\omega\left(\alpha+\eta\right)+C^2\eta\omega^2\right)\left(E\left[RA\right]\left(\alpha+\eta\right)+\left(C+1\right)\gamma\left(E\left[RA\right]+C\omega\right)+C\eta\omega\right)}, \tag{2.32}$$

$Var\left[RA-RAR\right]:$

$$\frac{\gamma\sigma^2\left(E\left[RA-RAR\right]+\omega\right)^3}{2\left(E\left[RA-RAR\right]^2C\left(\alpha+\eta\right)+E\left[RA-RAR\right]C\omega\left(\alpha+2\eta\right)+\omega\left(\alpha+C\eta\omega\right)\right)\left(E\left[RA-RAR\right]\left(\alpha+\eta\right)+\left(C+1\right)\gamma\left(E\left[RA-RAR\right]+\omega\right)+\eta\omega\right)}. \tag{2.33}$$

where $\delta = C\gamma$. Importantly we note that $E\left[RA\right]$ and $E\left[RA-RAR\right]$ are independent of δ and γ .

2.5.4 Intermediate Model (IM)

$$d[RA] = \left(\beta + \delta[RA-BP] - \left(\frac{\alpha[RA-RAR]}{\omega + [RA-RAR]} + \gamma + \eta\right)[RA]\right)dt \tag{2.34}$$

$$+ \sigma dW_t \tag{2.35}$$

$$d[RA-BP] = (\gamma[RA] + \lambda[RA-RAR] - (\delta+\nu)[RA-BP])\,dt, \tag{2.36}$$

$$d[RA-RAR] = (\nu[RA-BP] - \lambda[RA-RAR])\,dt, \tag{2.37}$$

$$E[RA] = \frac{\sqrt{4\beta\gamma\delta\lambda\nu\omega(\alpha+\eta) + (\beta\gamma\nu - \delta\eta\lambda\omega)^2} + \beta\gamma\nu - \delta\eta\lambda\omega}{2\gamma\nu(\alpha+\eta)}, \tag{2.38}$$

$$E[RA-BP] = \frac{\sqrt{4\beta\gamma\delta\lambda\nu\omega(\alpha+\eta) + (\beta\gamma\nu - \delta\eta\lambda\omega)^2} + \beta\gamma\nu - \delta\eta\lambda\omega}{2\delta\nu(\alpha+\eta)}, \tag{2.39}$$

$$E[RA-RAR] = \frac{\sqrt{4\beta\gamma\delta\lambda\nu\omega(\alpha+\eta) + (\beta\gamma\nu - \delta\eta\lambda\omega)^2} + \beta\gamma\nu - \delta\eta\lambda\omega}{2\delta\gamma(\alpha+\eta)}. \tag{2.40}$$

The variance equations are too large to fit in normal text and are thus contained in the respective Mathematica notebooks. Notice that when $\lambda = C\gamma$ or $\nu = C\delta$, the respective terms cancel out of $E[RA]$ and $E[RA-RAR]$

2.5.5 Intermediate Model (IM) with More Signaling Steps and a Separate Pool for Cyp Degradation

$$d\,[RA] = (\beta + \delta\,[RA-BP] - (\gamma+\eta)\,[RA])\,dt + \sigma dW_t \tag{2.41}$$

$$d\,[RA_2] = \left(\delta_2\,[RA-BP] - \left(\frac{\alpha\,[RA-RAR]}{\omega + [RA-RAR]} + \gamma_2\right)[RA]\right)dt \tag{2.42}$$

$$d\,[RA-BP] = (\gamma\,[RA] + \gamma_2\,[RA_2] + \lambda\,[RA_N] - (\delta+\delta_2+\nu)\,[RA-BP])\,dt, \tag{2.43}$$

$$d\,[RA_N] = (\nu\,[RA-BP] + \Lambda\,[RA-RAR] - (\lambda+\Gamma)\,[RA_N])\,dt, \tag{2.44}$$

$$d\,[RA-RAR] = (\Gamma\,[RA_N] - \Lambda\,[RA-RAR])\,dt, \tag{2.45}$$

The mean and variance equations are too large to fit in normal text and are thus contained in the respective Mathematica notebooks. Notice that when $\lambda = C\gamma = C_2\gamma_2$ or $\nu = C\delta = C_2\delta_2$, the respective terms cancel out of $E\,[RA]$ and $E\,[RA-RAR]$.

2.5.6 Retinoic Acid Model (RM)

$$d\left[RA_{out}\right] = \left(\beta - b\left[RA_{out}\right] + c\left[RA_{in}\right]\right)dt, \tag{2.46}$$

$$d\left[RA_{in}\right] = \left(b\left[RA_{out}\right] + \delta\left[RA-BP\right] - \left(\gamma\left[BP\right] + \eta + \frac{\alpha\left[RA-RAR\right]}{\omega + \left[RA-RAR\right]} - c\right)\left[RA_{in}\right]\right)dt \tag{2.47}$$

$$+\sigma dW_t, \tag{2.48}$$

$$d\left[RA-BP\right] = \left(\gamma\left[BP\right]\left[RA_{in}\right] + \lambda\left[BP\right]\left[RA-RAR\right] - \left(\delta + \nu\left[RAR\right]\right)\left[RA-BP\right]\right)dt, \tag{2.49}$$

$$d\left[RA-RAR\right] = \left(\nu\left[RA-BP\right]\left[RAR\right] - \lambda\left[BP\right]\left[RA-RAR\right]\right)dt, \tag{2.50}$$

$$d\left[RAR\right] = \left(\zeta - \nu\left[RA-BP\right]\left[RAR\right] + \lambda\left[BP\right]\left[RA-RAR\right] - r\left[RAR\right]\right)dt, \tag{2.51}$$

$$d\left[BP\right] = \left(a - \lambda\left[BP\right]\left[RA-RAR\right] - \gamma\left[BP\right]\left[RA_{in}\right] + \left(\delta + \nu\left[RAR\right]\right)\left[RA-BP\right] - u\left[BP\right]\right)dt, \tag{2.52}$$

$$E\left[RA_{out}\right] = \frac{\beta\gamma\zeta\nu\left(2\left(\alpha+\eta\right)+c\right) + c\sqrt{4\beta\gamma\delta\zeta\lambda\nu r\omega\left(\alpha+\eta\right) + \left(\beta\gamma\zeta\nu - \delta\eta\lambda r\omega\right)^2} + c\delta\eta\lambda\left(-r\right)\omega}{2b\gamma\zeta\nu\left(\alpha+\eta\right)}, \tag{2.53}$$

$$E\left[RA_{in}\right] = \frac{\beta\gamma\zeta\nu + \sqrt{4\beta\gamma\delta\zeta\lambda\nu r\omega\left(\alpha+\eta\right) + \left(\beta\gamma\zeta\nu - \delta\eta\lambda r\omega\right)^2} - \delta\eta\lambda r\omega}{2\gamma\zeta\nu\left(\alpha+\eta\right)}, \tag{2.54}$$

$$E\left[RA-BP\right] = \frac{a\left(\beta\gamma\zeta\nu + \sqrt{4\beta\gamma\delta\zeta\lambda\nu r\omega\left(\alpha+\eta\right) + \left(\beta\gamma\zeta\nu - \delta\eta\lambda r\omega\right)^2} - \delta\eta\lambda r\omega\right)}{2\delta\zeta\nu u\left(\alpha+\eta\right)}, \tag{2.55}$$

$$E\left[RA-RAR\right] = \frac{\beta\gamma\zeta\nu + \sqrt{4\beta\gamma\delta\zeta\lambda\nu r\omega\left(\alpha+\eta\right) + \left(\beta\gamma\zeta\nu - \delta\eta\lambda r\omega\right)^2} - \delta\eta\lambda r\omega}{2\delta\lambda r\left(\alpha+\eta\right)}, \tag{2.56}$$

$$E\left[RAR\right] = \frac{\zeta}{r}, \tag{2.57}$$

$$E\left[BP\right] = \frac{a}{u}. \tag{2.58}$$

The variance equations are too large to fit in normal text and are thus contained in the respective Mathematica notebooks. Notice $E\left[RA_{in}\right]$ and $E\left[RA-RAR\right]$ are independent of

a . By substitution we have that:

$$E\left[RA-BP\right]=\frac{\beta\gamma\zeta\nu+\sqrt{4\beta\gamma\delta\zeta\lambda\nu r\omega\left(\alpha+\eta\right)+\left(\beta\gamma\zeta\nu-\delta\eta\lambda r\omega\right)^{2}}-\delta\eta\lambda r\omega}{2\delta\zeta\nu\left(\alpha+\eta\right)}E\left[BP\right]. \quad (2.59)$$

2.5.7 Retinoic Acid Model with Binding Protein Feedback (RMF)

The equations are the same as RM except for:

$$d\left[BP\right]=\left(a-\lambda\left[BP\right]\left[RA-RAR\right]-\gamma\left[BP\right]\left[RA_{in}\right]+\left(\delta+\nu\left[RAR\right]\right)\left[RA-BP\right]-u\left[BP\right]+\frac{d\left[RA-RAR\right]}{e+\left[RA-RAR\right]}\right)dt. \quad (2.60)$$

The steady-state analysis results are too large to fit in normal text and are thus contained in the Mathematica notebooks.

2.5.8 Spatial Retinoic Acid Model

The equations are the same as RM with BP feedback except

$$d\left[RA_{out}\right]=\left(\beta\left(x\right)+D\Delta\left[RA_{out}\right]-b\left[RA_{out}\right]+c\left[RA_{in}\right]\right)dt+\sigma_{RA_{out}}\left[RA_{out}\right]dW_{t}^{out}, \quad (2.61)$$

where $\beta\left(x\right)=\beta_0 H\left(x-40\right)$ where H is the Heaviside step function denoting $x_0=40\mu m$ is the edge of production,

$$d\left[RA_{in}\right] = \left(b\left[RA_{out}\right] + \delta\left[BP\right]\left[RA-RAR\right] - \left(\gamma\left[BP\right] + \eta + \frac{\alpha\left[RA-RAR\right]}{\omega+\left[RA-RAR\right]} - c\right)\left[RA_{in}\right]\right)dt \tag{2.62}$$

$$+\sigma_{RA}\left[RA_{in}\right]dW_t^{in}, \tag{2.63}$$

and

$$d\left[RA-RAR\right] = \left(\nu\left[RA-BP\right]\left[RAR\right] - \lambda\left[BP\right]\left[RA-RAR\right]\right)dt \tag{2.64}$$

$$+\sigma_{RA-RAR}\left[RA-RAR\right]dW_t^{RA-RAR}, \tag{2.65}$$

where each dW_t is an uncorrelated Gaussian white noise. The spatial domain was a two-dimensional box with the x-domain [-100,400] and the y-domain [0,50] with units of μm . The problem was discretized to ODEs via the method of lines with a second-order discretization of the Laplacian and $dx = dy = 5\mu m$. For all sections, we fixed $D = 25.46\ \mu m^2/s$. The boundary was reflective on all ends except the right boundary, which was leaky with parameter 0.002.

When the Hox-Krox interactions are included, those portions of the system are defined by:

$$dg_h = \frac{c_h g_h^2 + \left(\kappa_h\left[RA-RAR\right]\right)^2}{1 + c_h g_h^2 + c_k g_k^2 + \left(\kappa_h\left[RA-RAR\right]\right)^2} - d_h g_h + a_h g_h dW_t^h, \tag{2.66}$$

$$dg_k = \frac{c_k g_k^2 + \left(\kappa_k\left[RA-RAR\right]\right)^2}{1 + c_h g_h^2 + c_k g_k^2 + \left(\kappa_h\left[RA-RAR\right]\right)^2} - d_k g_k + a_k g_k dW_t^h. \tag{2.67}$$

2.5.9 Numerical Parameter Search in Knockdown Experiments

The scheme is:

1. For every parameter p , take $x_p \in [-5,5]$ uniformly, and let $p = 10^{-x_p - baseExp_p}$.
2. Solve the model for 200 seconds with the initial condition at the steady-state value. Calculate the mean and variance.
3. Knock down the associated parameter by 90% and redo step 2.
4. Calculate the value ζ.

The $baseExp_p$ values are given in Table 2.1. 100,000 simulations were run per model. The simulation was solved using the Euler-Maruyama method with a $dt = 10^{-4}$ and the mean/-variance was calculated. For models without the explicit binding protein, the parameter γ was the one affected. For models with the explicit binding protein, the production parameter was the one affected. The simulation was solved using the same solver settings and the same Brownian path (the same Brownian path was used to simulate the embryo with the same conditions but with a different epigenetic makeup). Cyp was knocked down (from the parameter set that did not include the BP knockdown) by a 90% decrease to α . The same solver settings and the same Brownian path were used and the mean/variance was calculated. Using the mean/variance calculations for these three runs, a ζ was calculated for the BP-knockdown and a ζ was calculated for the Cyp-knockdown. Note that the percentages were calculated relative to the larger quantity, e.g.

$$\%\Delta Mean = \frac{abs\left(Mean_1 - Mean_2\right)}{max\left(Mean_1, Mean_2\right)} \tag{2.68}$$

to ensure a value between 0 and 1.

2.5.10 Numerical Parameter Search in Spatial Knockdown Experiments

100 simulations were run, with random parameter sets chosen by taking x_p uniformly from [-2,2] and letting $p = 10^{-x_p - baseExp_p}$. $baseExp_p$ was chosen so that the parameter range covers all of the most likely parameters, but slightly biased in order to decrease the amount of time to steady state to make the problem computationally feasible (boosting degradation, and increasing production so as to keep the total concentrations at reasonable levels). The simulations were accelerated using NVIDIA GTX 970 and GTX 980Ti GPUs via MATLAB's CUDA interface. The base values are given in Table 2.1.

The simulation was first solved to steady-state without noise at the highest possible dt , and then the model was solved using a more stable variant of a second order Runge-Kutta method via a method of lines discretization with $dt = 5 \times 10^{-5}s$ for 100 seconds, roughly matching the experimental setup of [111]. We note that the results are robust to the choice of final time point being an order magnitude less or greater, indicating convergence of the stochastic model to a quasi-steady distribution. The model was first solved using x_p and then with a 90% knockdown of a . At the end of each run, the 60% threshold from the non-knockdown control was used to set the boundary location. For each y, the lowest x above the threshold was chosen as the boundary location. The mean and the variance of these x values was used as the boundary mean and variance. This scheme is diagrammed in Figure 2.5.

2.5.11 Spatial Boundary Sharpening Experiments

The boundary sharpening experiments were solved using a method of lines approach. The steady-state gradient was first established by turning off the noise and solving the discretized PDE using an adaptive second order Rosenbrock method from DifferentialEquations.jl. Then the SPDE was solved for 500 seconds using the adaptive SRIW1[90]. After that, the Hox-Krox interactions were initiated, starting with a random steady state where Krox was zero and Hox started with each point in space having $0.1605+0.2X$ where X is a uniform random number. This was solved to steady state using the Tsit5 algorithm from DifferentialEquations.jl and then solved with noise for 10,000 seconds using the adaptive SRIW1 method.

For the boundary sharpening experiments, parameters were chosen to conform to regimes specified in previous models. The parameters were chosen as detailed in Table 2.1. From the results, the effective RA noise was calculated as the variance of RA at $x = 125\mu m$, which was the Hox-Krox boundary in the absence of noise.

2.5.12 ζ Determination From Data

To determine ζ from the data of Sosnik et al., 2016, the relative concentration values for free intracellular RA had to be converted to a sensible absolute concentration value in some arbitrary units by determining a 0. This background was discarded by subtracting out the mean relative abundance of the control experiment, which was .3132. The 0-adjusted values are given in in the accompanying MATLAB script. Since the embryos have no preferred pairing, a separate ζ was estimated from each pairwise interaction between knockdowns.

Chapter 3

Adaptive Methods for Stochastic Differential Equations via Natural Embeddings and Rejection Sampling with Memory

This chapter was published as [91]. It derives two algorithms: error estimators for high order Stochastic Runge-Kutta methods and a method for adaptive time stepping for stochastic equations termed Rejection Sampling with Memory (RSwM). These two methods together give rise to high order adaptive integrators for stochastic differential equations which are shown to give substantial speedups on models of biological processes like Epithelial-Mesnchymal Transitions (EMT).

3.1 Summary

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.

3.2 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 [26, 103, 20]. 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 [88, 8]. Together, the algorithm is able to effectively 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

$$dX_t = f(t, X_t)dt + g(t, X_t)dW_t, \tag{3.1}$$

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 [63, 50]. 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 [127, 97]. While strong order 2.0 SRK methods have been studied [10, 63], existing works mainly focus on strong order 1.5 Stochastic Runge-Kutta methods, such as a series of methods for Stratanovich SDEs [12, 9] and for Ito SDEs [58].

One recent advance in SRK methods is Rößler's methods for Ito SDEs [95]. 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 [13]. 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 [34] utilizes an order 1.0 algorithm with a Brownian path stored at the h, $\frac{h}{2}$, $\frac{h}{4}$ timepoints, 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 [81]. Lamba [67] 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 [13] . 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 3.1 and gives an overview of how the various developments come together to form one complete algorithm.

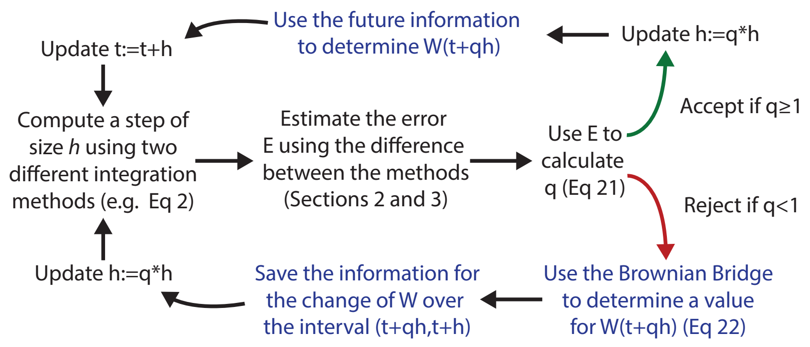

Update t:=t+h
Use the future information to determine W(t+qh)
Update h:=q*h
Compute a step of size *h* using two different integration methods (e.g. Eq 2)
Estimate the error E using the difference between the methods (Sections 2 and 3)
Use E to calculate q (Eq 21)
Accept if q≥1
Reject if q<1
Update h:=q*h
Save the information for the change of W over the interval (t+qh,t+h)
Use the Brownian Bridge to determine a value for W(t+qh) (Eq 22)

Figure 3.1: **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 3.5.

In section 3.3 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 3.4 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 3.5 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 3.6, 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 3.7, 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.

3.3 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 [95]. 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 [63]. 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 subsection 3.10.1, taking the following form:

$$\begin{aligned} U_{n+1} &= U_n + \sum_{i=1}^{s} \alpha_i f\left(t_n + c_i^{(0)} \Delta t, H_i^{(0)}\right) \Delta t + \qquad (3.2) \\ &\sum_{i=1}^{s} \left(\beta_i^{(1)} I_{(1)} + \beta_i^{(2)} \frac{I_{(1,1)}}{\sqrt{\Delta t}} + \beta_i^{(3)} \frac{I_{(1,0)}}{\Delta t} + \beta_i^{(4)} \frac{I_{(1,1,1)}}{\Delta t} \right) g\left(t_n + c_i^{(1)} \Delta t, H_i^{(1)}\right), \end{aligned}$$

with stages

$$\begin{aligned} H_i^{(0)} &= U_n + \sum_{j=1}^{s} A_{ij}^{(0)} f\left(t_n + c_j^{(0)} \Delta t, H_j^{(0)}\right) \Delta t \qquad (3.3) \\ &+ \sum_{j=1}^{s} B_{ij}^{(0)} g\left(t_n + c_j^{(1)} \Delta t, H_j^{(1)}\right) \frac{I_{(1,0)}}{\Delta t}, \\ H_i^{(1)} &= U_n + \sum_{j=1}^{s} A_{ij}^{(1)} f\left(t_n + c_j^{(0)} \Delta t, H_j^{(0)}\right) \Delta t \qquad (3.4) \\ &+ \sum_{j=1}^{s} B_{ij}^{(1)} g\left(t_n + c_j^{(1)} \Delta t, H_j^{(1)}\right) \sqrt{\Delta t}. \end{aligned}$$

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 subsection 3.10.3. 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 subsection 3.10.2, taking the form

$$\begin{aligned} U_{n+1} &= U_n + \sum_{i=1}^{s} \alpha_i f\left(t_n + c_i^{(0)}\Delta t, H_i^{(0)}\right)\Delta t + \\ &\quad \sum_{i=1}^{s}\left(\beta_i^{(1)} I_{(1)} + \beta_i^{(2)} \frac{I_{(1,0)}}{\Delta t}\right) g(t_n + c_i^{(1)}\Delta t), \end{aligned} \tag{3.5}$$

with stages

$$H_i^{(0)} = U_n + \sum_{j=1}^{s} A_{ij}^{(0)} f\left(t_n + c_j^{(0)}\Delta t, H_j^{(0)}\right)\Delta t + \sum_{j=1}^{s} B_{ij}^{(0)} g\left(t_n + c_j^{(1)}\Delta t\right)\frac{I_{(1,0)}}{\Delta t}. \tag{3.6}$$

Given the efficiency of order 1.5 SRK methods for stochastic problems, we will develop Runge-Kutta-Fehlburg-like embedded algorithms [8], 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 [95] denoted by coefficients $\left(A^{(i)}, B^{(i)}, \beta^{(i)}, \alpha\right)$. Notice that the order 1.0 conditions from subsection 3.10.1 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 3.1, and $\tilde{X}$ be the numerical solution from the algorithm with coefficients $\left(A^{(i)}, B^{(i)}, \tilde{\beta}^{(i)}, \alpha\right)$. By Equation 3.2, the difference between the numerical solutions on the next step is

$$\bar{X} - \tilde{X} = \sum_{i=1}^{s}\left(\beta_i^{(3)}\frac{I_{(1,0)}}{\Delta t} + \beta_i^{(4)}\frac{I_{(1,1,1)}}{\Delta t}\right) g\left(t_n + c_i^{(1)}\Delta t, H_i^{(1)}\right). \tag{3.7}$$

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 subsection 3.10.3) 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 3.1 using the coefficients $(A_0, B_0, \beta_i, \tilde{\alpha})$, then by Equation 3.2 notice

$$\bar{X} - \hat{X} = -\frac{\Delta t}{6} f\left(t_n + c_1^{(0)}\Delta t, H_1^{(0)}\right) + \frac{\Delta t}{6} f\left(t_n + c_2^{(0)}\Delta t, H_2^{(0)}\right). \tag{3.8}$$

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 3.1 for which steps are taken according to Equation 3.2, Equation 3.3, Equation 3.4.

Adding the two previous error equations Equation 3.7 and Equation 3.8, gives E, the error estimation via the difference between the order 1.5 and order 1.0 method, as

$$\begin{aligned} E \;=\; & -\frac{\Delta t}{6} f\left(t_n + c_1^{(0)}\Delta t, H_1^{(0)}\right) + \frac{\Delta t}{6} f\left(t_n + c_2^{(0)}\Delta t, H_2^{(0)}\right) \\ & + \sum_{i=1}^{s} \left(\beta_i^{(3)} \frac{I_{(1,0)}}{\Delta t} + \beta_i^{(4)} \frac{I_{(1,1,1)}}{\Delta t}\right) g\left(t_n + c_i^{(1)}\Delta t, H_i^{(1)}\right). \end{aligned} \tag{3.9}$$

Thus ESRK1 is a pair of order 1.0/order 1.5 embedded algorithms. Notice that every term in the error calculation Equation 3.9 is also part of the calculation for the order 1.5 timestep from Equation 3.2 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.

$c^{(0)}$	$A^{(0)}$	$B^{(0)}$	
$c^{(1)}$	$A^{(1)}$	$B^{(1)}$	
	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 3.1: **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 $\widetilde{\beta^{(i)}}$. 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.

3.4 Error estimation via natural embeddings

The construction from section 3.3 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 3.2, this gives the deterministic error estimate

$$\begin{aligned} E_D &= \Delta t \left(f\left(t_n + c_{k_1}^{(0)} \Delta t, H_{k_1}^{(0)}\right) + f\left(t_n + c_{k_2}^{(0)} \Delta t, H_{k_2}^{(0)}\right)\right) \\ &\quad - \left(f\left(t_n + c_{k_3}^{(0)} \Delta t, H_{k_3}^{(0)}\right) + f\left(t_n + c_{k_4}^{(0)} \Delta t, H_{k_4}^{(0)}\right)\right), \end{aligned} \tag{3.10}$$

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

$$E_N = \left| \sum_{i \in I_2} \left(\beta_i^{(3)} \frac{I_{(1,0)}}{\Delta t} + \beta_i^{(4)} \frac{I_{(1,1,1)}}{\Delta t} \right) g\left(t_n + c_i^{(1)} \Delta t, H_i^{(1)}\right) \right|. \tag{3.11}$$

By the order conditions of subsection 3.10.1, 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*

$$E = \delta E_D + E_N, \tag{3.12}$$

where

$$E_D = \left|\Delta t \sum_{i \in I_1} (-1)^{\sigma(i)} f\left(t_n + c_i^{(0)}\Delta t, H_i^{(0)}\right)\right| \leq \Delta t \sum_{i \in I_1} \left|f\left(t_n + c_i^{(0)}\Delta t, H_i^{(0)}\right)\right|, \tag{3.13}$$

$$E_N = \left|\sum_{i \in I_2} \left(\beta_i^{(3)} \frac{I_{(1,0)}}{\Delta t} + \beta_i^{(4)} \frac{I_{(1,1,1)}}{\Delta t}\right) g\left(t_n + c_i^{(1)}\Delta t, H_i^{(1)}\right)\right|. \tag{3.14}$$

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 3.3. 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 3.2. As a result, the error estimate is naturally obtained without any additional function evaluations. Lastly, the equation Equation 3.12 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

$$dX_t = f(X_t,t)dt + \sum_{k}^{d} g_k(X_t,t)dW_t, \tag{3.15}$$

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

$$\begin{aligned} E_D &= \left|\Delta t \sum_{i\in I_1}(-1)^{\sigma(i)} f\left(t_n + c_i^{(0)}\Delta t, H_i^{(0)}\right)\right| \leq \Delta t \sum_{i\in I_1}\left| f\left(t_n + c_i^{(0)}\Delta t, H_i^{(0)}\right)\right|, \qquad (3.16)\\ E_N &= \left|\sum_{k}^{d}\sum_{i\in I_2}\left(\beta_i^{(3)}\frac{I_{(1,0)}}{\Delta t} + \beta_i^{(4)}\frac{I_{(1,1,1)}}{\Delta t}\right) g_k\left(t_n + c_i^{(1)}\Delta t, H_i^{(1)}\right)\right|. \qquad (3.17)\end{aligned}$$

Note that in the case of SRA-type methods (additive noise), a similar theorem also holds. By Equation 3.5 on the multiple Ito dimension SDE Equation 3.15 we utilize the same construction and receive a natural embedding with the error estimation Equation 3.12 containing the same deterministic error estimation *Equation* 3.13, with a new noise-error estimation:

$$E_N = \left|\sum_{k}^{d}\sum_{i=1}^{s}\beta_i^{(2)}\frac{I_{(1,0)}}{\Delta t} g_k(t_n + c_i^{(1)}\Delta t)\right|. \tag{3.18}$$

3.5 Three algorithms for rejection sampling with memory

3.5.1 Rejection sampling for stochastic differential equations

As in deterministic rejection sampling algorithms [8, 88, 40], define

$$e = \sqrt{\frac{1}{n}\sum_{i}^{n}\left(\frac{E_i}{sc_i}\right)}, \tag{3.19}$$

where

$$sc_i = \epsilon_{abs} + X_t^{(i)}\epsilon_{rel}, \tag{3.20}$$

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 3.4, 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

$$q = \left(\frac{1}{\gamma e}\right)^{1/(\mathcal{O}+1)}, \tag{3.21}$$

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 $\min\left(h_{max}, qh\right)$ 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 subsection 3.10.3, 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 [83, 71], if $W(0) = 0$, $W(h) = L$, and $q \in (0, 1)$, then

$$W(qh) \sim N\left(qL, (1-q)qh\right). \tag{3.22}$$

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 3.22.

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 3.22, giving all of the necessary information for computing a step of size qh.

3.5.2 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 3.22. 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 3.22 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.

3.5.3 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 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 subsection 3.10.4. 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 3.6,

Algorithm 1 RSwM1

```
1:  Set the values $\epsilon$, $h_{max}$, $T$
2:  Set $t = 0$, $W = 0$, $Z = 0$, $X = X_0$
3:  Take an initial $h$, $\Delta Z, \Delta W \sim N(0, h)$
4:  while $t < T$ do
5:      Attempt a step with $h$, $\Delta W$, $\Delta Z$ to calculate $X_{temp}$ according to (2)
6:      Calculate E according to (9)
7:      Update $q$ using (21)
8:      if $(q < 1)$ then                                              ▷ % Reject the Step
9:          Take $\Delta\tilde{W} \sim N\left(q\Delta W, (1-q)qh\right)$ and $\Delta\tilde{Z} \sim N\left(q\Delta Z, (1-q)qh\right)$
10:         Calculate $\overline{\Delta W} = \Delta W - \Delta\tilde{W}$ and $\overline{\Delta Z} = \Delta Z - \Delta\tilde{Z}$
11:         Push $\left((1-q)\,h, \overline{\Delta W}, \overline{\Delta Z}\right)$ into stack $S$
12:         Update $h := qh$
13:         Update $\Delta W := \Delta\tilde{W}$, $\Delta Z := \Delta\tilde{Z}$
14:     else                                                           ▷ % Accept the Step
15:         Update $t := t + h$, $W := W + \Delta W$, $Z := Z + \Delta Z$, $X = X + X_{temp}$
16:         if ($S$ is empty) then
17:             Update $c := \min(h_{max}, qh)$, $h := \min(c, T - t)$
18:             Take $\Delta W, \Delta Z \sim N(0, h)$
19:         else
20:             Pop the top of $S$ as $L$
21:             Update $h := L_1$, $\Delta W := L_2$, $\Delta Z := L_3$
22:         end if
23:     end if
24: end while
```

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 3.6 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.

3.6 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 3.31, Equation 3.33, and Equation 3.35 defined in subsection 3.10.5.

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.

Algorithm 2 RSwM3

```
 1: Set the values $\epsilon$, $h_{max}$, $T$
 2: Set $t = 0$, $W = 0$, $Z = 0$, $X = X_0$
 3: Take an initial $h$, $\Delta Z, \Delta W \sim N(0, h)$
 4: while $t < T$ do
 5:     Attempt a step with $h$, $\Delta W$, $\Delta Z$ to calculate $X_{temp}$ according to (2)
 6:     Calculate E according to (9)
 7:     Update $q$ using (21)
 8:     if $(q < 1)$ then                                          ▷ % Reject the Step
 9:         Set $h_s = 0$, $\Delta W = 0$, $\Delta Z = 0$
10:         while $S_2$ is not empty do
11:             Pop the top of $S_2$ as $L$
12:             if $h_s + L_1 < (1 - q)h$ then
13:                 Update $h_s := h_s + L_1$
14:                 Update $\Delta W_{tmp} := \Delta W_{tmp} + L_2$, $\Delta Z_{tmp} := \Delta Z_{tmp} + L_3$
15:             else
16:                 Push $L$ onto $S_2$ and break
17:             end if
18:         end while
19:         Set $h_K = h - h_s$, $K_2 = \Delta W - \Delta W_{tmp}$, $K_3 = \Delta Z - \Delta Z_{tmp}$
20:         Set $q_K = \frac{qh}{h_K}$
21:         Take $\Delta\tilde{W} \sim N(q_K K_2, (1 - q_K) q_K L_1)$
22:         Take $\Delta\tilde{Z} \sim N(q_K K_3, (1 - q_K) q_K L_1)$
23:         Pop $((1 - q_K) h_K, K_2 - \Delta\tilde{W}, K_3 - \Delta\tilde{Z})$ onto $S_1$
24:         Pop $(q_K L_1, \Delta\tilde{W}, \Delta\tilde{Z})$ onto $S_2$
25:         Update $\Delta W = \Delta\tilde{W}$, $\Delta Z = \Delta\tilde{Z}$, $h = qh$
26:     else                                                       ▷ % Accept the Step
27:         Update $t := t + h$, $W := W + \Delta W$, $Z := Z + \Delta Z$, $X = X + X_{temp}$
28:         Empty $S_2$
29:         Update $c := \min(h_{max}, qh)$, $h := \min(c, T - t)$
30:         Set $h_s = 0$, $\Delta W = 0$, $\Delta Z = 0$
31:         while $S$ is not empty do
32:             Pop the top of $S_1$ as $L$
33:             if $(h_s + L_1 < h)$ then                      ▷ % Temporary not far enough
34:                 Update $h_s := h_s + L_1$, $\Delta W := \Delta W + L_2$, $\Delta Z := \Delta Z + L_3$
35:                 Push a copy of $L$ onto $S_2$
36:             else                                           ▷ % Final part of step from stack
37:                 Set $q_{tmp} = \frac{h - h_s}{L_1}$
38:                 Let $\Delta\tilde{W} \sim N(q_{tmp} L_2, (1 - q_{tmp}) q_{tmp} L_1)$
39:                 Let $\Delta\tilde{Z} \sim N(q_{tmp} L_3, (1 - q_{tmp}) q_{tmp} L_1)$
40:                 Push $((1 - q_{tmp}) L_1, L_2 - \Delta\tilde{W}, L_3 - \Delta\tilde{Z})$ onto $S_1$
41:                 Update $\Delta W := \Delta W + \Delta\tilde{W}$, $\Delta Z := \Delta Z + \Delta\tilde{Z}$, $h_s := h_s + q_{tmp} L_1$
42:             end if
43:         end while
                        ▷ % Update for last portion to step. Note zero if final part is from stack
44:         if $(h - h_s$ is not zero) then
45:             Let $\eta_W, \eta_Z \sim N(0, h - h_s)$
46:             Update $\Delta W = \Delta W + \eta_W$, $\Delta Z = \Delta Z + \eta_Z$
47:             Push $(h - h_s, \eta_W, \eta_Z)$ onto $S_2$
48:         end if
49:     end if
50: end while
```

3.6.1 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 3.31, 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 non-parametric test of distributions which is known for being very sensitive and easily rejecting the null-hypothesis of standard normality [35]. 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 3.2. 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.

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 3.3. 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.

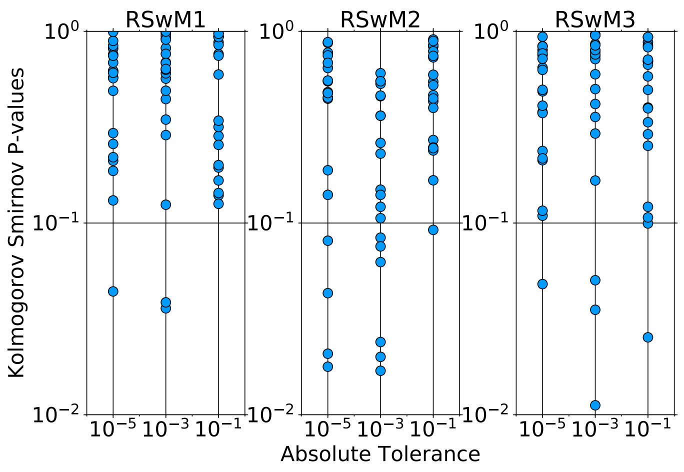


Figure 3.2: **Adaptive algorithm Kolmogorov-Smirnov Tests.** Equation 3.31 was solved from $t = 0$ to $t = 2$. Scatter plots of the p-values from Kolmogorov-Smirnov tests against the normal distribution. At the end of each run, a Kolmogorov-Smirnov test was performed on the values at end of the Brownian 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.

3.6.2 Accuracy and efficiency

Next we tested the accuracy and efficiency of the three algorithms RSwM1, RSwM2, RSwM3 on the three example SDEs Equation 3.31, Equation 3.33, and Equation 3.35. The results, as summarized in Figure 3.4 and Figure 3.5, 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.

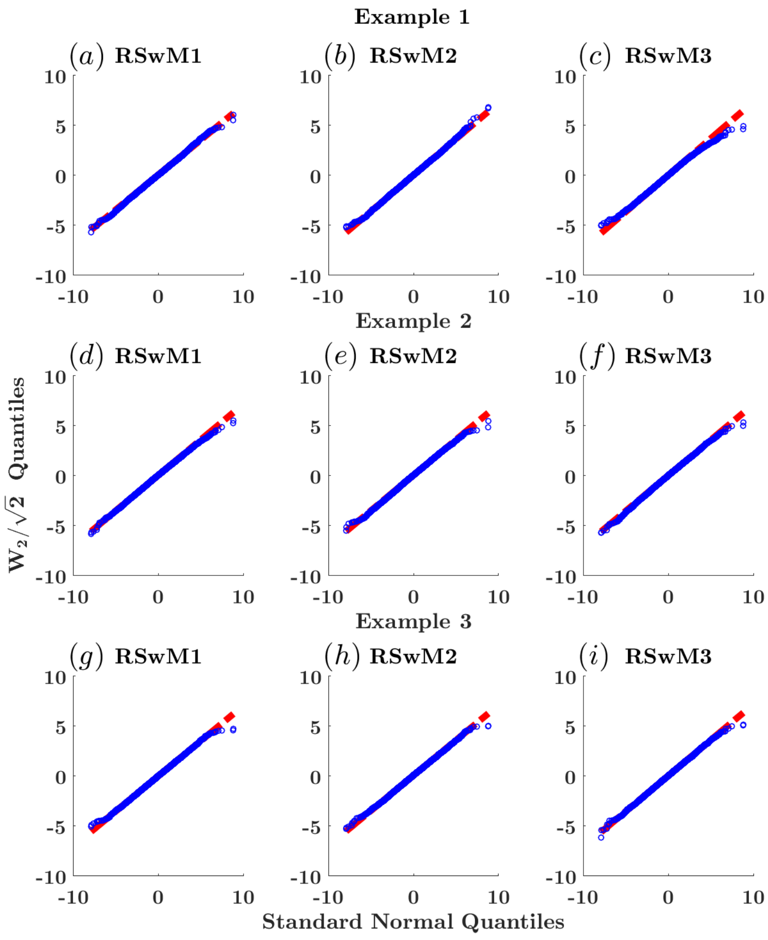


Figure 3.3: **Adaptive algorithm correctness checks.** QQplots of the distribution of the Brownian path at the end time $T = 2$ over 10,000 paths. The x-axis is the quantiles of the standard normal distribution while the y-axis is the estimated quantiles for the distribution of $W_2/\sqrt{2}$. Each row 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.

3.6.3 Summary of numerical correctness and efficiency

The results of the numerical experiments can be summarized as follows. From subsection 3.6.1 we see that by simulating Equation 3.31 Equation 3.33, and Equation 3.35 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

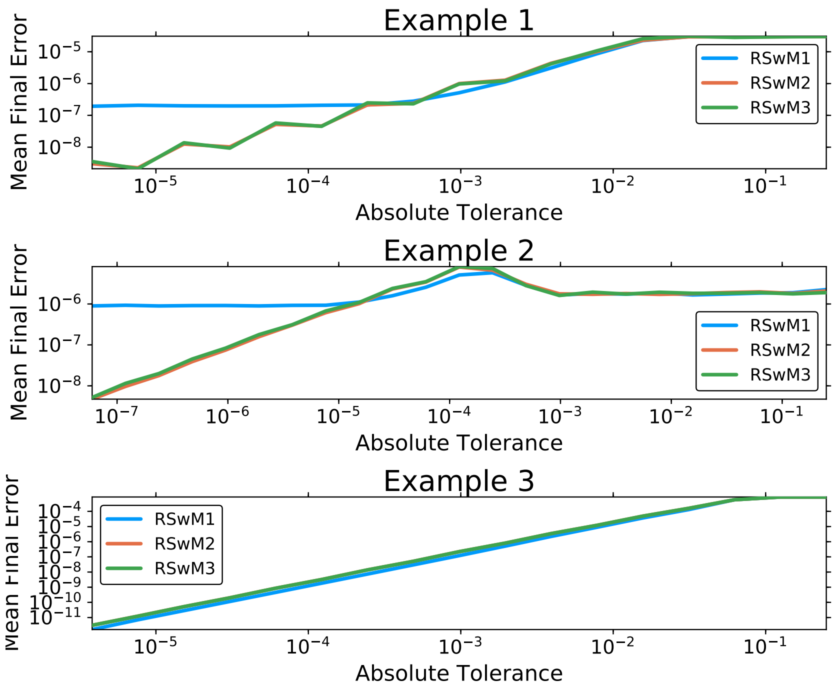


Figure 3.4: **Adaptive Algorithm Error Comparison on Equation 3.31, Equation 3.33, and Equation 3.35.** Comparison of the rejection sampling with memory algorithms on examples 1-3. Along the x-axis is ϵ which is the user chosen local error tolerance. The y-axis is the average l^2 error along the path. These values are taken as the mean for 100 runs. Both axes are log-scaled.

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

From subsection 3.6.2 we see that the RSwM2 and RSwM3 algorithms are the most efficient on the test equations Equation 3.31 and Equation 3.33 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

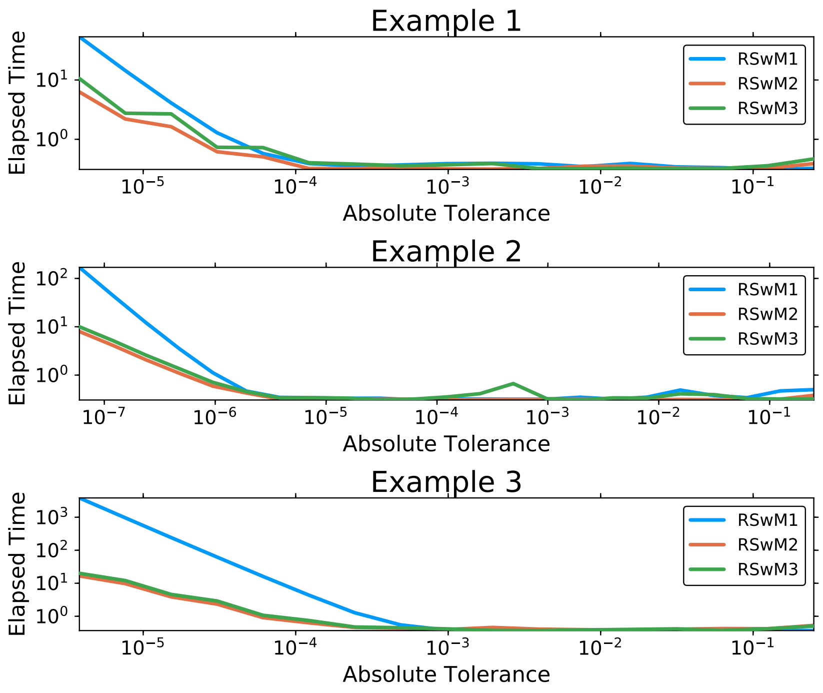


Figure 3.5: **Adaptive Algorithm Timing Comparison. Comparison of the rejection sampling with memory algorithms.** Along the x-axis is ϵ which is the user chosen local error tolerance. In the y-axis time is plotted (in seconds). The values are the elapsed time for a 1000 Both axes are log-scaled.

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.

3.7 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.

3.7.1 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 3.6. 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.

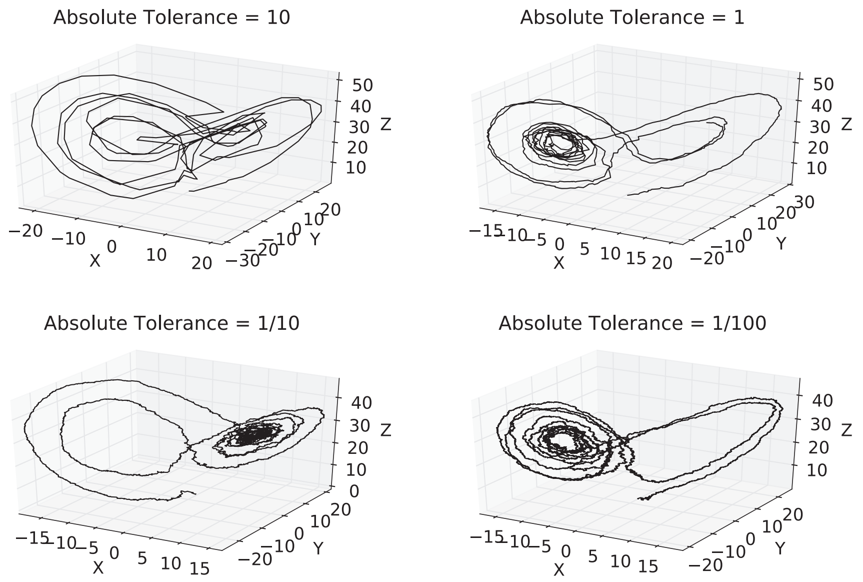


Figure 3.6: **Solution to the Lorenz system Equation 3.39**. Solved on $t \in [0, 10]$ with additive noise and parameters $\alpha = 10$, $\rho = 28$, $\sigma = 3$, and $\beta = 8/3$ at varying tolerances. The system was solved using ESRK+RSwM3 with the relative tolerance fixed at zero and varying absolute tolerances.

3.7.2 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 subsection 3.10.6. 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 3.7 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 subsection 3.10.6 10,000 times via the Euler-Maruyama, Runge-Kutta Milstein (RK-Mil) [63], 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 3.2. 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 3.7 we see that the most stiff portions

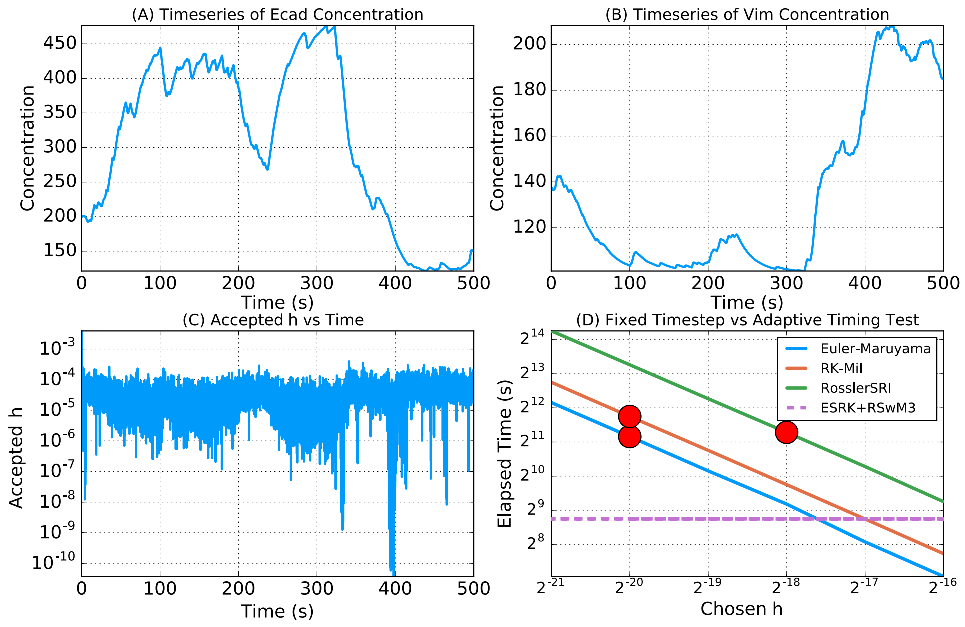


Figure 3.7: **Stochastic cell differentiation model solutions.** (A) Timeseries of the concentration of $[Ecad]$. The solution is plotted once every 100 accepted steps due to memory limitations. (B) Timeseries of the concentration of $[Vim]$. The solution is plotted once every 100 accepted steps due 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.

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.

	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 3.2: **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.

3.8 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 [21] 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

$$q = \left(\frac{1}{\gamma e}\right)^2, \tag{3.23}$$

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 [40]. We found that the following tends to work well in practice:

$$q = \min(qmax, \max(q, qmin)), \tag{3.24}$$

where $qmax$ and $qmin$ are parameters set by the user. While for deterministic systems it's

	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 3.3: *qmax* **determination tests.** Equation 3.31, Equation 3.33, and Equation 3.35 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 subsection 3.10.6 solved with the same tolerance settings as in subsection 3.7.2 to solve a Monte Carlo simulation of 10,000 simulations.

suggested that *qmax* is between 4 and 10 [40], 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 3.3 where we found a suitable value $qmax = 1.125$ which we used as the default throughout this paper (except in subsection 3.6.2 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 [40] 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

```
 1: Let $d_0 = \|X_0\|$
 2: Calculate $f_0 = f(X_0, t)$ and $\sigma_0 = 3g(X_0, t)$
 3: Let $d_1 = \|\max(|f_0 + \sigma_0|, |f_0 - \sigma_0|)\|$
 4: if $d_0 < 10^{-5}$ or $d_1 < 10^{-5}$ then
 5:     Let $h_0 = 10^{-6}$
 6: else
 7:     Let $h_0 = 0.01(d_0/d_1)$
 8: end if
 9: Calculate an Euler step: $X_1 = X_0 + h_0 f_0$
10: Calculate new estimates: $f_1 = f(X_1, t)$ and $\sigma_0 = 3g(X_1, t)$
11: Determine $\sigma_1^M = \max(|\sigma_0 + \sigma_1|), |\sigma_0 - \sigma_1|)$
12: Let $d_2 = \|\max(|f_1 - f_0 + \sigma_1^M|, |f_1 - f_0 - \sigma_1^M|)\|/h_0$
13: if $\max(d_1, d_2) < 10^{-15}$ then
14:     Let $h_1 = \max(10^{-6}, 10^{-3}h_0)$
15: else
16:     Let $h_1 = 10^{-(2+\log_{10}(\max(d_1,d_2))/(order+0.5)}$
17: end if
18: Let $h = \min(100h_0, h_1)$
```

Note that $\|\cdot\|$ is the norm as defined in Equation 3.19 with Equation 3.20. 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.

3.9 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 [76]. 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.

3.10 Extended Information

3.10.1 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 $f,g\in C^{1,2}(\mathcal{I}\times\mathbb{R}^d,\mathbb{R}^d)$, $c^{(i)}=A^{(i)}e$, $e=(1,1,1,1)^T$ [95].

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

and the additional conditions for order 1.5:

1. $\alpha^TB^{(0)}e=1$
2. $\alpha^TA^{(0)}e=\frac{1}{2}$
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 $g\in C^1(\mathcal{I},\mathbb{R}^d)$ [95]. From these conditions he proposed the following strong order 1.5 scheme found in Table 3.4 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 3.4: **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.

3.10.3 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:

$$I_{(1)} = \int_{t_n}^{t_{n+1}} dW_s, \tag{3.25}$$

$$I_{(1,1)} = \int_{t_n}^{t_{n+1}} \int_{t_n}^{s} dW_u dW_s, \tag{3.26}$$

$$I_{(1,1,\ldots,1_k)} = \int_{t_n}^{t_{n+1}} \int \ldots \int_{t_n}^{s} dW_{u_1} \ldots dW_s. \tag{3.27}$$

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 [127, 95] is as follows:

$$\begin{aligned} I_{(1,1)} &= \frac{1}{2}(I_{(1)}^2 - h), && (3.28)\\ I_{(1,1,1)} &= \frac{1}{6}(I_{(1)}^3 - 3hI_{(1)}), && (3.29)\\ I_{(1,0)} &= \frac{1}{2}h\left(I_{(1)} + \frac{1}{\sqrt{3}}\zeta\right). && (3.30)\end{aligned}$$

$I_{(1)}$ 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.

3.10.4 RSwM2 algorithm specification

The algorithm for RSwM2 is specified as Algorithm 4.

Algorithm 4 RSwM2

```
1: Set the values $\epsilon$, $h_{max}$, $T$
2: Set $t = 0$, $W = 0$, $Z = 0$, $X = X_0$
3: Take an initial $h$, $\Delta Z, \Delta W \sim N(0, h)$
4: while $t < T$ do
5:     Attempt a step with $h$, $\Delta W$, $\Delta Z$ to calculate $X_{temp}$ according to (2)
6:     Calculate E according to (9)
7:     Update $q$ using (21)
8:     if $(q < 1)$ then                                              ▷ % Reject the Step
9:         Take $\Delta\tilde{W} \sim N(q\Delta W, (1-q)qh)$ and $\Delta\tilde{Z} \sim N(q\Delta Z, (1-q)qh)$
10:        Calculate $\overline{\Delta W} = \Delta W - \Delta\tilde{W}$ and $\overline{\Delta Z} = \Delta Z - \Delta\tilde{Z}$
11:        Push $\left((1-q)h, \overline{\Delta W}, \overline{\Delta Z}\right)$ into stack $S$
12:        Update $h := qh$
13:        Update $\Delta W := \Delta\tilde{W}$, $\Delta Z := \Delta\tilde{Z}$
14:    else                                                            ▷ % Accept the Step
15:        Update $t := t + h$, $W := W + \Delta W$, $Z := Z + \Delta Z$, $X = X + X_{temp}$
16:        Update $c := \min(h_{max}, qh)$, $h := \min(c, T - t_n)$
17:        Set $h_s = 0$, $\Delta W = 0$, $\Delta Z = 0$
18:        while $S$ is not empty do
19:            Pop the top of $S$ as $L$
20:            if $(h_s + L_1 < h)$ then                            ▷ % Temporary not far enough
21:                Update $h_s := h_s + L_1$, $\Delta W := \Delta W + L_2$, $\Delta Z := \Delta Z + L_3$
22:            else                                                ▷ % Final part of step from stack
23:                Set $q_{tmp} = \frac{h-h_s}{L_1}$
24:                Let $\Delta\tilde{W} \sim N(q_{tmp}L_2, (1-q_{tmp})q_{tmp}L_1)$
25:                Let $\Delta\tilde{Z} \sim N(q_{tmp}L_3, (1-q_{tmp})q_{tmp}L_1)$
26:                Push $((1-q_{tmp})L_1, L_2 - \Delta\tilde{W}, L_3 - \Delta\tilde{Z})$ onto $S$
27:                Update $\Delta W := \Delta W + \Delta\tilde{W}$, $\Delta Z := \Delta\tilde{Z}$ , $h_s := h_s + q_{tmp}L_1$
28:            end if
29:        end while
                    ▷ % Update for last portion to step. Note zero if final part is from stack
30:        if $(h - h_s$ is not zero) then
31:            Let $\eta_W, \eta_Z \sim N(0, h - h_s)$
32:            Update $\Delta W = \Delta W + \eta_W$, $\Delta Z = \Delta Z + \eta_Z$
33:        end if
34:    end if
35: end while
```

3.10.5 Example equations

The three example equations are:

Example 1.

$$dX_t = \alpha X_t dt + \beta X_t dW_t, \;\; X_0 = \frac{1}{2}, \tag{3.31}$$

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

$$X_t = X_0 e^{\left(\beta - \frac{\alpha^2}{2}\right)t + \alpha W_t}. \tag{3.32}$$

Example 2.

$$dX_t = -\left(\frac{1}{10}\right)^2 \sin\left(X_t\right)\cos^3\left(X_t\right)dt + \frac{1}{10}\cos^2\left(X_t\right)dW_t, \;\; X_0 = \frac{1}{2}, \tag{3.33}$$

Actual Solution:

$$X_t = \arctan\left(\frac{1}{10}W_t + \tan\left(X_0\right)\right). \tag{3.34}$$

Example 3.

$$dX_t = \left(\frac{\beta}{\sqrt{1+t}} - \frac{1}{2\left(1+t\right)}X_t\right)dt + \frac{\alpha\beta}{\sqrt{1+t}}dW_t, \;\; X_0 = \frac{1}{2}, \tag{3.35}$$

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

$$X_t = \frac{1}{\sqrt{1+t}} X_0 + \frac{\beta}{\sqrt{1+t}} \left(t + \alpha W_t\right). \tag{3.36}$$

Example 4.

$$\begin{aligned} dX_t &= \alpha(Y_t - X_t) + \sigma dW_t, && (3.37)\\ dY_t &= X_t(\rho - Z_t) - Y_t + \sigma dW_t, && (3.38)\\ dZ_t &= X_t Y_t - \beta Z_t + \sigma dW_t, && (3.39) \end{aligned}$$

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

3.10.6 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 [48] to model stochastic transitions between Epithelial and Mesenchymal states.

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

$$
\begin{aligned}
\frac{d\,[SNAIL]}{dt} &= k_{SNAIL}\left([snail1]-[SR]\right)-kd_{SNAIL}\,[SNAIL]\\
\frac{d\,[miR34]}{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}\,[SR]\\
\frac{d\,[SR]}{dt} &= Tk\left(K_{SR}\left([snail1]-[SR]\right)\left([miR34]-[SR]\right)-[SR]\right)\\
\frac{d\,[zeb]}{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}}}\\
&- 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}}
\end{aligned}
$$

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 3.5: **Table of Parameter Values for the Stochastic Cell Model.**

$$\begin{aligned}
\frac{d\left[Vim\right]}{dt} &= k0_V + \frac{B}{\left(1+\left[OVOL2\right]/J3_V\right)} - kd_V\left[Vim\right] \\
\frac{d\left[OVOL2\right]}{dt} &= k0_0 + k_0\frac{1}{1+\left(\left[ZEB\right]/J_0\right)^{n_0}} - kd_O\left[OVOL2\right]
\end{aligned}$$

where

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

The parameter values are given in Table 3.5.

Chapter 4

Stability-Optimized High Order Methods and Stiffness Detection for Pathwise Stiff Stochastic Differential Equations

This chapter was submitted to SIAM Scientific Computing. It details the derivation of stochastic Runge-Kutta methods with optimal stability. This chapter extends the results of Chapter 3 and thus utilizes the same adaptivity scheme and error estimates but with increased efficiency due to enhanced stability. The focus is on optimal stability methods for additive noise, affine noise, and diagonal noise. Explicit, implicit, and IMEX (implicit-explicit) integrators are derived and evaluated on various biological models, including the discretized stochastic partial differential equations of Chapter 2.

4.1 Summary

Stochastic differential equations (SDE) often exhibit large random transitions. This property, which we denote as pathwise stiffness, causes transient bursts of stiffness which limit the allowed step size for common fixed time step explicit and drift-implicit integrators. We present four separate methods to efficiently handle this stiffness. First, we utilize a computational technique to derive stability-optimized adaptive methods of strong order 1.5 for SDEs. The resulting explicit methods are shown to exhibit substantially enlarged stability regions which allows for them to solve pathwise stiff biological models orders of magnitude more efficiently than previous methods like SRIW1 and Euler-Maruyama. Secondly, these integrators include a stiffness estimator which allows for automatically switching between implicit and explicit schemes based on the current stiffness. In addition, adaptive L-stable strong order 1.5 implicit integrators for SDEs and stochastic differential algebraic equations (SDAEs) in mass-matrix form with additive noise are derived and are demonstrated as more efficient than the explicit methods on stiff chemical reaction networks by nearly 8x. Lastly, we developed an adaptive implicit-explicit (IMEX) integration method based off of a common method for diffusion-reaction-convection PDEs and show numerically that it can achieve strong order 1.5. These methods are benchmarked on a range of problems varying from non-stiff to extreme pathwise stiff and demonstrate speedups between 5x-6000x while showing computationally infeasibility of fixed time step integrators on many of these test equations.

4.2 Introduction

Stochastic differential equations (SDEs) are dynamic equations of the form

$$dX_t = f(t, X_t)dt + g(t, X_t)dW_t, \tag{4.1}$$

where X_t is a d-dimensional vector, $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is the drift coefficient, and $g : \mathbb{R}^d \rightarrow \mathbb{R}^{d\times m}$ is the diffusion coefficient which describes the amount and mixtures of the noise process W_t which is a m-dimensional Brownian motion. SDEs are of interest in scientific disciplines because they can exhibit behaviors which are not found in deterministic models. For example, An ODE model of a chemical reaction network may stay at a constant steady state, but in the presence of randomness the trajectories may be switching between various steady states [93, 120, 48]. In many cases, these unique features of stochastic models are pathwise-dependent and are thus not a property of the evolution of the mean trajectory. However, these same effects cause random events of high numerical stiffness, which we denote as pathwise stiffness, which can cause difficulties for numerical integration methods.

A minimal example of pathwise stiffness is demonstrated in the equation

$$dX_t = \left[-1000X_t\left(1 - X_t\right)\left(2 - X_t\right)\right]dt + g(t, X_t)dW_t, \;\; X_0 = 2, \;\; t \in [0, 5]\,. \tag{4.2}$$

with additive noise $g(t, X_t) = 10$ where a sample trajectory is shown in Figure 4.1. This equation has two stable steady states, one at $X = 0$ and another at $X = 2$, which the solution switches between when the noise is sufficiently large. While near a steady state the derivative is approximately zero making the problem non-stiff, during these transitions the derivative of the drift term reaches a maximum of ≈ 400. This means that in order to be stable, explicit Stochastic Runge-Kutta (SRK) must have a small Δt. This display of large, transient, and random switching behavior in a given trajectory causes stochastic bursts of numerical stiffness, a phenomena which we will denote pathwise stiffness. The fixed time step Euler-Maruyama method would require $dt < 4 \times 10^{-3}$ to be stable for most trajectories, thus requiring greater than 2×10^4 steps to solve this 1-dimensional SDE. In many cases the switching behavior can be rare (due to smaller amounts of noise) or can happen finitely many times like in the multiplicative noise version with $g(t, X_t) = 10X_t$. Yet even if these switches are only a small portion of the total time, the stability requirement imposed by their

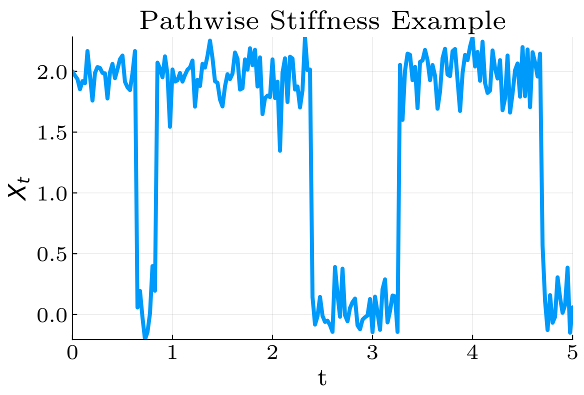


Figure 4.1: **Example of a Pathwise Stiff Solution**. Depicted is a sample trajectory of Equation 4.2 solved using the SOSRI methods developed in this manuscript with $reltol = abstol = 10^{-2}$.

existence determines the possible stepsizes and thus has a large contribution to the overall computational cost. While implicit methods can be used to increase the stability range, this can vastly increase the overall computational cost of each step, especially in the case large systems of SDEs like discretizations of stochastic reaction-diffusion equations. In addition, implicit solvers have in practice a smaller stability region due to requiring convergence of the quasi-Newton solvers for the implicit steps. This problem is mitigated in ODE software by high-quality stage predictors given by extrapolation algorithms for good initial conditions for the Newton steps [42]. However, there are no known algorithms for stage predictors in the presence of large noise bursts and thus we will demonstrate that classic implicit solvers have a form of instability. Thus both fixed time step explicit and implicit solvers are inadequate for efficiently handling this common class of SDEs.

Since these features exist in the single trajectories of the random processes, methods which attempt to account for the presence of such bursts must do so on each individual trajectory in order to be efficient. In previous work, the authors have shown that by using adaptive time-stepping, a stochastic reaction network of 19 reactants is able to be solved with an average time step 100,000 times larger than the value that was found necessary for stability during the random stiff events for a high order SRK method [90]. This demonstrated that the key to solving these equations efficiently required controlling the time steps in a pathwise manner.

However, the methods were still largely stability-bound, meaning the chosen tolerances to solve the model were determined by what was necessary for stability and was far below the error necessary for the application. The purpose of this investigation is to develop numerical methods with the ability to better handle pathwise stiffness and allow for efficient solving of large Monte Carlo experiments.

We approach this problem through four means. First, we develop adaptive stability-optimized SRK methods with enlarged stability regions. This builds off of similar work for ODE integrators which optimize the coefficients of a Butcher tableau to give enhanced stability [69, 3, 117]. Similar to the Runge-Kutta Chebyschev methods [42] (and the S-ROCK extension to the stochastic case [65, 2, 64]), these methods are designed to be efficient for equations which display stiffness without fully committing to implicit solvers. Given the complexity of the stochastic stability equations and order conditions, we develop a novel and scalable mechanism for the derivation of "optimal" Runge-Kutta methods. We use this method to design stability-optimized methods for additive noise and diagonal noise SDEs. We show through computational experiments that these adaptive stability-optimized SRK methods can adequately solve transiently stiff equations without losing efficiency in non-stiff problems.

On the other hand, to handle extreme stiffness we develop implicit RK methods for SDEs and stochastic differential algebraic equations (SDAEs) in mass matrix form with additive noise. We extend the definition of L-stability to additive noise SDEs and develop two strong order 1.5 methods: a fully implicit 2-stage L-stable method and an extension of the a well-known L-stable explicit first stage singly diagonally implicit RK (ESDIRK) method due to Kennedy and Carpenter which is commonly used for convection-diffusion-reaction equations [59]. To the author's knowledge, these are the first high order adaptive L-stable methods for SDEs and the first adaptive proposed SDAE integrators. In addition, to extend the utility of these additive noise methods, we derive an extension of the methods for additive SDEs to affine

SDEs (mixed multiplicative and additive noise terms) through a Lamperti transformation [79]. Lastly, in order to handle extreme transient stiffness, for each of these types of methods we derive computationally cheap methods for detecting stiffness and switching between implicit and explicit integrators in the presence of stiffness. We show that these methods can robustly detect pathwise stiff transients and thus can serve as the basis for automatic switching methods for SDEs. Together we test on non-stiff, semi-stiff, and stiff equations with 2 to $6 \times 20 \times 100$ SDEs from biological literature and show speedups between 6x-60x over the previous adaptive SRIW1 algorithm, and demonstrate the infeasibility of common explicit and implicit methods (Euler-Maruyama, Runge-Kutta Milstein, Drift-Implicit Stochastic θ-Method, and Drift-Implicit θ Runge-Kutta Milstein) found as the basis of many SDE solver packages [100, 37, 52].

4.3 Adaptive Strong Order 1.0/1.5 SRK Methods for Additive and Diagonal Noise SDEs

The class of methods we wish to study are the adaptive strong order 1.5 SRK methods for diagonal noise [95, 90]. Diagonal noise is the case where the diffusion term g is diagonal matrix $(\sigma_i X_t^i)$ and includes phenomenological noise models like multiplicative and affine noise. The diagonal noise methods utilize the same general form and order conditions as the methods for scalar noise so we use their notation for simplicity. The strong order 1.5 methods for scalar noise are of the form

$$X_{n+1} = X_n + \sum_{i=1}^{s} \alpha_i f\left(t_n + c_i^{(0)}h, H_i^{(0)}\right) + \tag{4.3}$$

$$\sum_{i=1}^{s}\left(\beta_i^{(1)} I_{(1)} + \beta_i^{(2)}\frac{I_{(1,1)}}{\sqrt{h}} + \beta_i^{(3)}\frac{I_{(1,0)}}{h} + \beta_i^{(4)}\frac{I_{(1,1,1)}}{h}\right) g\left(t_n + c_i^{(1)}h\right) \tag{4.4}$$

with stages

$$H_i^{(0)} = X_n + \sum_{j=1}^{s} A_{ij}^{(0)} f\left(t_n + c_j^{(0)}h, H_j^{(0)}\right) h + \sum_{j=1}^{s} B_{ij}^{(0)} g\left(t_n + c_j^{(1)}h, H_j^{(1)}\right) \frac{I_{(1,0)}}{h} \tag{4.5}$$

$$H_i^{(1)} = X_n + \sum_{j=1}^{s} A_{ij}^{(1)} f\left(t_n + c_j^{(0)}h, H_j^{(0)}\right) h + \sum_{j=1}^{s} B_{ij}^{(1)} g\left(t_n + c_j^{(1)}h, H_j^{(1)}\right) \sqrt{h}$$

where the I_j are the Wiktorsson approximations to the iterated stochastic integrals [127]. In the case of additive noise, defined as having the diffusion coefficient satisfy $g(t, X_t) \equiv g(t)$, reduces to the form

$$X_{n+1} = X_n + \sum_{i=1}^{s} \alpha_i f\left(t_n + c_i^{(0)}h, H_i^{(0)}\right) + \sum_{i=1}^{s}\left(\beta_i^{(1)} I_{(1)} + \beta_i^{(2)}\frac{I_{(1,0)}}{h}\right) g\left(t_n + c_i^{(1)}h\right) \tag{4.6}$$

with stages

$$H_i^{(0)} = X_n + \sum_{j=1}^{s} A_{ij}^{(0)} f\left(t_n + c_j^{(0)}h, H_j^{(0)}\right) h + \sum_{j=1}^{s} B_{ij}^{(0)} g\left(t_n + c_j^{(1)}h\right) \frac{I_{(1,0)}}{h}. \tag{4.7}$$

The tuple of coefficients $\left(A^{(j)}, B^{(j)}, \beta^{(j)}, \alpha\right)$ thus fully determines the SRK method. These coefficients must satisfy the constraint equations described in 4.9.4 in order to receive strong order 1.5. These methods are appended with error estimates

$$
\begin{aligned}
E_D &= \left|\Delta t \sum_{i\in I_1} (-1)^{\sigma(i)} f\left(t_n + c_i^{(0)}\Delta t, H_i^{(0)}\right)\right| \text{ or } E_D = \Delta t \sum_{i\in I_1} \left| f\left(t_n + c_i^{(0)}\Delta t, H_i^{(0)}\right)\right| \\
E_N &= \left|\sum_{i\in I_2} \left(\beta_i^{(3)} \frac{I_{(1,0)}}{\Delta t} + \beta_i^{(4)} \frac{I_{(1,1,1)}}{\Delta t}\right) g\left(t_n + c_i^{(1)}\Delta t, H_i^{(1)}\right)\right|
\end{aligned}
$$

and the rejection sampling with memory (RSwM) algorithm to give it fully adaptive time-stepping [90]. Thus unlike in the theory of ordinary differential equations [68, 27, 15, 116, 107], the choice of coefficients for SRK methods does not require explicitly finding an embedded method when developing an adaptive SRK method and we will therefore take for granted that each of the derived methods is adaptive.

4.4 Optimized-Stability High Order SRK Methods with Additive Noise

We use a previous definition of a discrete approximation as numerically stable if for any finite time interval $[t_0, T]$, there exists a positive constant Δ_0 such that for each $\epsilon > 0$ and each $\delta \in (0, \Delta_0)$

$$
\lim_{\left|X_0^\delta - \bar{X}_0^\delta\right| \to 0} \sup_{t_0 \le t \le T} P\left(\left|X_t^\delta - \bar{X}_t^\delta\right| \ge \epsilon\right) = 0 \tag{4.8}
$$

where X_n^δ is a discrete time approximation with maximum step size $\delta > 0$ starting at X_0^δ and $\bar{X}_n^\delta$ respectively starting at $\bar{X}_n^\delta$ [63]. For additive noise, we consider the complex-valued linear test equations

$$dX_t = \mu X_t dt + dW_t \tag{4.9}$$

where μ is a complex number. In this framework, a scheme which can be written in the form

$$X_{n+1}^h = X_n^h G(\mu h) + Z_n^\delta \tag{4.10}$$

with a constant step size $\delta \equiv h$ and Z_n^δ are random variables which do not depend on the Y_n^δ, then the region of absolute stability is the set where for $z = \mu h$, $|G(z)| < 1$.

The additive SRK method can be written as

$$X_{n+1}^h = X_n^h + z\left(\alpha \cdot H^{(0)}\right) + \beta^{(1)}\sigma I_{(1)} + \sigma\beta^{(2)}\frac{I_{(1,0)}}{h} \tag{4.11}$$

where

$$H^{(0)} = \left(I - zA^{(0)}\right)^{-1}\left(\hat{X}_n^h + B^{(0)}e\sigma\frac{I_{(1,0)}}{h}\right) \tag{4.12}$$

where $\hat{X}_n^h$ is the size s constant vector of elements X_n^h and $e = (1,1,1,1)^T$. By substitution we receive

$$X_{n+1}^h = X_n^h\left(1 + z\left(\alpha \cdot \left(I - zA^{(0)}\right)^{-1}\right)\right) + \tag{4.13}$$

$$\left(I - zA^{(0)}\right)^{-1} B^{(0)}e\sigma\frac{I_{(1,0)}}{h} + \beta^{(1)}\sigma I_{(1)} + \sigma\beta^{(2)}\frac{I_{(1,0)}}{h} \tag{4.14}$$

This set of equations decouples since the iterated stochastic integral approximation I_j are random numbers and are independent of the X_n^h. Thus the stability condition is determined by the equation

$$G(z) = 1 + z\alpha \cdot \left(I - zA^{(0)}\right)^{-1} \tag{4.15}$$

which one may notice is the stability equation of the drift tableau applied to a deterministic ODE [14]. Thus the stability properties of the deterministic Runge-Kutta methods carry over to the additive noise SRA methods on this test equation. However, most two-stage tableaus from ODE research were developed to satisfy higher order ODE order constraints which do not apply. Thus we will instead look to maximize stability while satisfying the stochastic order constraints.

4.4.1 Explicit Methods for Non-Stiff SDEs with Additive Noise

Stability-Optimal 2-Stage Explicit SRA Methods

For explicit

methods, $A^{(0)}$ and $B^{(0)}$ are lower diagonal and we receive the simplified stability function

$$G(z) = 1 + A_{21}z^2\alpha_2 + z\left(\alpha_1 + \alpha_2\right) \tag{4.16}$$

for a two-stage additive noise SRK method. For this method we will find the method which optimizes the stability in the real part of z. Thus we wish to find $A^{(0)}$ and α s.t. the negative real roots of $|G(z)| = 1$ are minimized. By the quadratic equation we see that there exists only a single negative root: $z = \frac{1-\sqrt{1+8\alpha_2}}{2\alpha_2}$. Using Mathematica's minimum function, we determine that the minimum value for this root subject to the order constraints is $z = \frac{3}{4}\left(1 - \sqrt{\frac{19}{3}}\right) \approx -1.13746$. This is achieved when $\alpha = \frac{2}{3}$, meaning that the SRA1 method due to Rossler achieves the maximum stability criteria. However, given extra degrees of freedom, we attempted to impose that $c_1^{(0)} = c_1^{(1)} = 0$ and $c_2^{(0)} = c_2^{(1)} = 1$ so that the error estimator spans the whole interval. This can lead to improved robustness of the adaptive error estimator. In fact, when trying to optimize the error estimator's span we find that there is no error estimator which satisfies $c_2^{(0)} > \frac{3}{4}$ which is the span of the SRA1 method [95]. Thus SRA1 is the stability-optimized 2-stage explicit method which achieves the most robust error estimator.

$$
\begin{aligned}
&A^{(0)} = \begin{pmatrix} 0 & 0 \\ \frac{3}{4} & 0 \end{pmatrix}, \; B^{(0)} = \begin{pmatrix} 0 & 0 \\ \frac{3}{2} & 0 \end{pmatrix}, \; \alpha = \begin{pmatrix} \frac{1}{3} \\ \frac{2}{3} \end{pmatrix} \\
&\beta^{(1)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \; \beta^{(2)} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \; c^{(0)} = \begin{pmatrix} 0 \\ \frac{3}{4} \end{pmatrix}, \; c^{(1)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}
\end{aligned} \tag{4.17}
$$

Stability-Optimal 3-Stage Explicit SRA Methods

For the 3-stage SRA method, we receive the simplified stability function

$$G(z) = A_{21}A_{31}\alpha_3 z^3 + A_{21}\alpha_2 z^2 + A_{31}\alpha_3 z^2 + A_{32}\alpha_3 z^2 + \alpha_1 z + \alpha_2 z + \alpha_3 z + 1 \tag{4.18}$$

To optimize this method, we attempted to use the same techniques as before and optimize the real values of the negative roots. However, in this case we have a cubic polynomial and the root equations are more difficult. Instead, we turn to a more general technique to handle the stability optimization which will be employed in later sections as well. To do so, we generate an optimization problem which we can numerically solve for the coefficients. To simplify the problem, we let $z \in \mathbb{R}$ and define the function:

$$f\left(z, w; N, M\right) = \int_D \chi_{G(z)\leq 1}(z) dz \tag{4.19}$$

Notice that f is the area of the stability region when D is sufficiently large. Thus we define the stability-optimized SRK method for additive noise SDEs as the set of coefficients which achieves

$$\max_{A^{(i)},B^{(i)},\beta^{(i)},\alpha} f(z) \tag{4.20}$$

$$\text{subject to: Order Constraints}$$

In all cases we impose $0 < c_i^{(0)}, c_i^{(1)} < 1$. We use the order constraints to simplify the problem to a nonlinear optimization problem on 14 variables with 3 equality constraints and 4 inequality constraints (with bound constraints on the 10 variables). However, we found that simplifying the problem even more to require $c_1^{(0)} = c_1^{(1)} = 0$ and $c_3^{(0)} = c_3^{(1)} = 1$ did not significantly impact the stability regions but helps the error estimator and thus we reduced the problem to 10 variables, 3 equality constraints, and 2 inequality constraints. This was optimized using the COBYLA local optimization algorithm [55, 87] with randomized initial conditions 100 times and all gave similar results. In the Mathematica notebook we show the effect of changing the numerical integration region D on the results, but conclude that a D which does not bias the result for better/worse real/complex handling does not improve the result. The resulting algorithm, SOSRA, we given by the coefficients in table in Section 4.9.3. Lastly, we used the condition that $c_2^{(0)} = c_3^{(0)} = c_2^{(1)} = c_3^{(1)} = 1$ to allow for free stability detection (discussed in Section 4.6.4). The method generated with this extra constraint is SOSRA2 whose coefficients are in the table in Section 4.9.3. These methods have their stability regions compared to SRA1 and SRA3 in Figure 4.2 where it is shown that the SOSRA methods more than doubles the allowed time steps when the eigenvalues of the Jacobian are dominated by the real part.

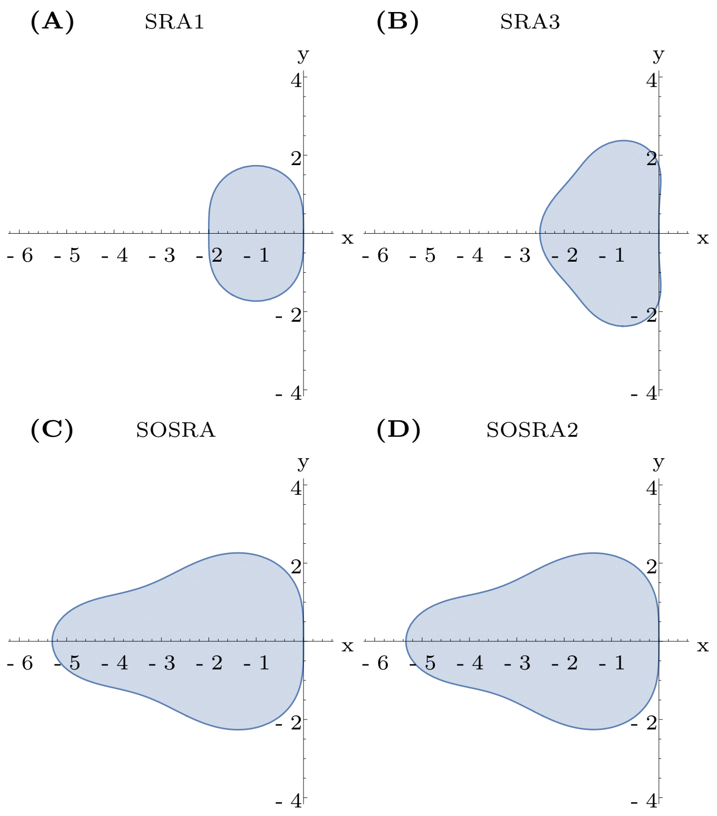


Figure 4.2: **SOSRA Stability Regions**. The stability regions ($|G(z)| < 1$) are plotted in the (x, y)-plane for $z = x + iy$. **(A)** SRA1. **(B)** SRA3. **(C)** SOSRA. **(D)** SOSRA2

4.4.2 Drift Implicit Methods for Stiff SDEs with Additive Noise

An L-Stable 2-Stage (Semi-)Implicit SRA Method

It's clear that, as in the case for deterministic equations, the explicit methods cannot be made A-stable. However, the implicit two-stage additive noise SRK method is determined by

$$G(z)=\frac{z(A_{11}(A_{22}z-\alpha_2 z-1)+A_{12}z(\alpha_1-A_{21})+A_{21}\alpha_2 z-A_{22}(\alpha_1 z+1)+\alpha_1+\alpha_2)+1}{A_{11}z(A_{22}z-1)-z(A_{12}A_{21}z+A_{22})+1} \tag{4.21}$$

which is *A*-stable if

$$A_{11}z(A_{22}z-1)-z(A_{12}A_{21}z+A_{22})+1>z(A_{11}(A_{22}z-\alpha_2 z-1)+A_{12}z(\alpha_1-A_{21}) \tag{4.22}$$

$$+A_{21}\alpha_2 z-A_{22}(\alpha_1 z+1)+\alpha_1+\alpha_2)+1. \tag{4.23}$$

Notice that the numerator equals the denominator if and only if $z=0$ or

$$z=\frac{\alpha_1+\alpha_2}{(A_{22}-A_{12})\,\alpha_1+(A_{11}-A_{21})\,\alpha_2}. \tag{4.24}$$

From the order conditions we know that $\alpha_1+\alpha_2=1$ which means that no root exists with $Re(z)<0$ if $(A_{22}-A_{12})\,\alpha_1+(A_{11}-A_{21})\,\alpha_2>0$. Thus under these no roots conditions, we can determine A-stability by checking the inequality at $z=1$, which gives $1>(A_{22}-A_{12})\,\alpha_1+(A_{11}-A_{21})\,\alpha_2$. Using the order condition, we have a total of four constraints on the $A^{(0)}$ and α:

$$\begin{aligned}\left(A_{11}+A_{12}\right)\alpha_1+\left(A_{21}+A_{22}\right)\alpha_2 &= \frac{1}{2} \\ \alpha_1+\alpha_2 &= 1 \\ 0<\left(A_{22}-A_{12}\right)\alpha_1+\left(A_{11}-A_{21}\right)\alpha_2 &< 1\end{aligned} \tag{4.25}$$

However, A-stability is not sufficient for most ODE integrators to properly handle stiff equations and thus extra properties generally imposed [42]. One important property we wish to extend to stochastic integrators is L-stability. The straightforward extension of L-stability is the condition

$$\lim_{z\to\infty} G(z)=0. \tag{4.26}$$

This implies that

$$\frac{-A_{11}A_{22}+A_{11}\alpha_2+A_{12}A_{21}-A_{12}\alpha_1-A_{21}\alpha_2+A_{22}\alpha_2\alpha_1}{A_{12}A_{21}-A_{11}A_{22}}=0 \tag{4.27}$$

The denominator is $-\det(A^{(0)})$ which implies $A^{(0)}$ must be non-singular. Next, we attempt to impose B-stability on the drift portion of the method. We use the condition due to Burrage and Butcher that for $B=\operatorname{diag}(\alpha_1,\alpha_2)$ $M=BA^{(0)}+A^{(0)}B-\alpha\alpha^T$ (for ODEs) [11], we require both B and M to be non-negative definite. However, in the supplemental Mathematica notebooks we show computationally that there is no 2-stage SRK method of this form which satisfies all three of these stability conditions. Thus we settle for A-stability and L-stability.

Recalling that $c^{(0)}$ and $c^{(1)}$ are the locations in time where f and g are approximated respectively, we wish to impose

$$\begin{aligned} c_1^{(0)} &= 0 \\ c_2^{(0)} &= 1 \\ c_1^{(1)} &= 0 \\ c_2^{(1)} &= 1 \end{aligned} \tag{4.28}$$

so that the error estimator covers the entire interval of integration. Since $c^{(0)} = A^{(0)}e$, this leads to the condition $A_{21} + A_{22} = 1$. Using the constraint-satisfaction algorithm FindInstance in Mathematica, we look for tableaus which satisfy the previous conditions with the added constraint of semi-implicitness, i.e. $B^{(0)}$ is lower triangular. This assumption is added because the inverse of the normal distribution has unbounded moments, and thus in many cases it mathematically simpler to consider the diffusion term as explicit (though there are recent methods which drop this requirement via truncation or extra assumptions on the solution [75]). However, we find that there is no coefficient set which meets all of these requirements. However, if we relax the interval estimate condition to allow $0 \leq c_2^{(0)} \leq 1$, we find an A-L stable method:

$$A^{(0)} = \begin{pmatrix} 1 & \frac{-41}{64} \\ \frac{32}{41} & \frac{9}{41} \end{pmatrix}, \quad B^{(0)} = \begin{pmatrix} \frac{5}{8} & 0 \\ 0 & \frac{7}{3} \end{pmatrix}, \quad \alpha = \begin{pmatrix} \frac{32}{41} \\ \frac{9}{41} \end{pmatrix} \tag{4.29}$$

$$\beta^{(1)} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad \beta^{(2)} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \quad c^{(0)} = \begin{pmatrix} \frac{23}{64} \\ 1 \end{pmatrix}, \quad c^{(1)} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

which we denote LSRA. If we attempt to look for a 2-stage SDIRK-like method to reduce the complexity of the implicit equation, i.e. $A_{12}^{(0)} = 0$, using FindInstance we find the constraints unsatisfiable. Note that if we drop the semi-implicit assumption we find that the full constraints cannot be satisfied there (we still cannot satisfy $c_1^{(0)} = 0$ and $c_2^{(0)} = 1$), and there does not exist a 2-stage A-L stable SDIRK method in that case.

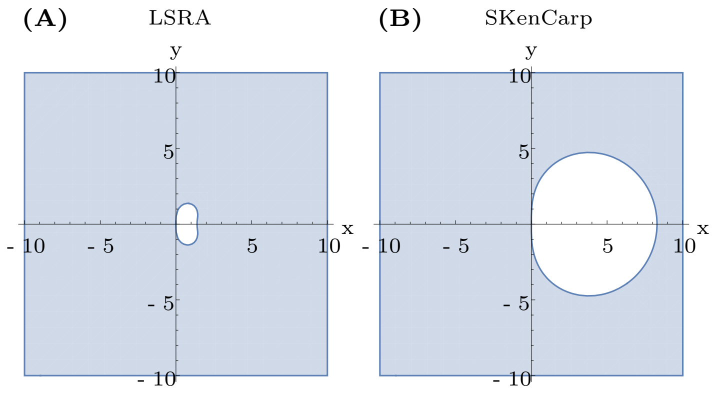


Figure 4.3: **Implicit SRA Stability Regions**. The stability regions ($|G(z)| < 1$) are plotted in the (x, y)-plane for $z = x + iy$. **(A)** LSRA. **(B)** SKenCarp

Extensions of ODE Implicit Runge-Kutta Methods to Implicit SRA Methods

Since the stability region of the SRA methods is completely determined by the deterministic portion $A^{(0)}$, in some cases there may exist a sensible extension of implicit Runge-Kutta methods for ordinary differential equations to high order adaptive methods stochastic differential equations with additive noise which keep the same stability properties. Since the order constraints which only involve the deterministic portions $A^{(0)}$, $c^{(0)}$, and α match the conditions required for ODE integrators, existence is dependent on finding $\beta^{(1)}$, $\beta^{(2)}$, $c^{(1)}$, and $B^{(0)}$ that satisfy the full order constraints. In this case, an adaptive error estimator can be added by using the same estimator as the ODE method (which we call E_D) but adding the absolute size of the stochastic portions

$$E_N = \left| \sum_{i=1}^{s} \left(\beta_i^{(1)} I_{(1)} + \beta_i^{(2)} \frac{I_{(1,0)}}{h} \right) \right| \tag{4.30}$$

leading to the error estimator

$$E = \delta E_D + E_N. \tag{4.31}$$

This can be shown similarly to the construction in [90]. Given the large literature on implicit RK methods for ODEs, this presents a large pool of possibly good methods and heuristically one may believe that these would do very well in the case of small noise.

However, we note that there does not always exist such an extension. Using the constraint-satisfaction algorithm FindInstance in Mathematica, we looked for extensions of the explicit first stage singly-diagonally implicit RK (ESDIRK) method TRBDF2 [49] and could not find values satisfying the constraints. In addition, we could not find values for an extension of the 5th order Radau IIA method [43, 42] which satisfies the constraints. In fact, our computational search could not find any extension of a 3-stage L-stable implicit RK method which satisfies the constraints.

But, the 4-stage 3rd order ODE method due to Kennedy and Carpenter [59] can be extended to the following:

$$A^{(0)} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1767732205903}{4055673282236} & \frac{1767732205903}{4055673282236} & 0 & 0 \\ \frac{2746238789719}{10658868560708} & -\frac{640167445237}{6845629431997} & \frac{1767732205903}{4055673282236} & 0 \\ \frac{1471266399579}{7840856788654} & -\frac{4482444167858}{7529755066697} & \frac{11266239266428}{11593286722821} & \frac{1767732205903}{4055673282236} \end{pmatrix}, \tag{4.32}$$

$$\alpha = \begin{pmatrix} \frac{1471266399579}{7840856788654} \\ -\frac{4482444167858}{7529755066697} \\ \frac{11266239266428}{11593286722821} \\ \frac{1767732205903}{4055673282236} \end{pmatrix}$$

$$\beta^{(1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \ \beta^{(2)} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ -1 \end{pmatrix}, \ c^{(0)} = \begin{pmatrix} 0 \\ \frac{1767732205903}{4055673282236} \\ \frac{3}{5} \\ 1 \end{pmatrix}, \ c^{(1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$B_{2,1}^{(0)} \approx -12.246764387585055918338744103409192607986567514699471403397969732723452087723101$$

$$B_{4,3}^{(0)} \approx -14.432096958608752822047165680776748797565142459789556194474191884258734697161106$$

The exact values for $B_{2,1}$ and $B_{4,3}$ are shown in 4.9.3. (E)SDIRK methods are particularly interesting because these methods can be solved using a single factorization of the function of the Jacobian $I - \gamma dtJ$ where J is the Jacobian. Additionally, explicit handling of the noise term is similar to the Implicit-Explicit (IMEX) form for additive Runge-Kutta methods in that it occurs by adding a single constant term to the Newton iterations in each stage, meaning it does not significantly increase the computational cost. The chosen ESDIRK method has a complimentary explicit tableau to form an IMEX additive Runge-Kutta method, and the chosen values for the stochastic portions are simultaneously compatible with the order conditions for this tableau. In Section 4.7.1 we numerically investigate the order of the IMEX extension of the method and show that it matches the convergence of the other SRA methods on the test equation. One thing to note is that since the problem is additive noise the method is never implicit in the dependent variables in the noise part, so in theory this can also be extended with $B^{(1)}$ implicit as well (with convergence concerns due to the non-finite inverse moments of the Normal distribution [63]).

Note on Implementation and Stage Prediction

One has to be careful with the implementation to avoid accumulation of floating point error for highly stiff equations. For our implementation, we used a method similar to that described in [49]. The implicit stages were defined in terms of

$$z_i = hf\left(t + c_i^{(0)}, H_j^{(0)}\right) \tag{4.33}$$

where X_0 is the previous step, and thus the iterations become

$$H_j^{(0)} = \gamma z_i + \sum_{j=1}^{i-1} A_{ij}^{(0)} f\left(t_n + c_j^{(0)} h, H_j^{(0)}\right) h + \sum_{j=1}^{i-1} B_{ij}^{(0)} g\left(t_n + c_j^{(1)} h\right) \frac{I_{(1,0)}}{h} = \gamma z_i + \alpha_i. \tag{4.34}$$

This gives the implicit system for the residual:

$$G(z_i) = z_i - hf\left(t_n + c_i^{(0)} h, \gamma z_i + \alpha_i\right)$$

which has a Jacobian $I - \gamma hJ$ where J is the Jacobian of f and thus is the same for each stage. For choosing the values to start the Newton iterations, also known as stage prediction, we tested two methods. The first is the trivial stage predictor which is $z_i = z_j$ for the j s.t. $j < i$ and $c_j < c_i$, i.e. using the closest derivative estimate. The other method that was tested is what we denote the stochastic minimal residual estimate given by $H_j^{(0)} = \alpha_i$

or $z_i = 0$. This method takes into account the stochastic bursts at a given step and thus demonstrated much better stability.

Note on Mass Matrices

We note that these methods also apply to solving ODEs with mass-matrices of the form:

$$MdX_t = f(t, X_t)dt + Mg(t, X_t)dW_t.$$

The derivation of the method is the same, except in this case we receive the implicit system

$$G(z_i) = Mz_i - hf\left(t_n + c_i^{(0)}h, \gamma z_i + \alpha_i\right)$$

which has a Jacobian $M - \gamma hJ$. Like in the ODE case, these implicit methods can thus solve DAEs in mass-matrix form (the case where M is singular), though we leave discussion of convergence for future research. One interesting property to note is that a zero row in the mass matrix corresponds to a constraint equation which is only dependent on the output of f since the multiplication of g by M is zero in that same corresponding row. Thus when a singular mass matrix is applied to the noise equation, the corresponding constraints are purely deterministic relations. Thus while this is a constrained form, properties like conservation of energy in physical models can still be placed on the solution using this mass-matrix formulation.

4.5 Optimized-Stability Methods for Affine Noise via Transformation

Given the efficiency of the methods for additive noise, one method for developing efficient methods for more general noise processes is to use a transform of diagonal noise processes to additive noise. This transform is due to Lamperti [79], which states that the SDE of the form

$$dX_t = f(t, X_t)dt + \sigma(t, X_t)R(t)dW_t \tag{4.35}$$

where $\sigma > 0$ is a diagonal matrix with diagonal elements $\sigma_i(t, X_{i,t})$ has the transformation

$$Z_{i,t} = \psi_i(t, X_{i,t}) = \int \frac{1}{\sigma_i(x,t)} dx \mid_{x=X_{i,t}} \tag{4.36}$$

which will result in an Ito process with the ith element given by

$$dZ_{i,t} = \left(\frac{\partial}{\partial t}\psi_i(t,x) \mid_{x=\psi^{-1}(t,Z_{i,t})} + \frac{f_i(t,\psi^{-1}(t,Z_t))}{\sigma_i\left(t,\psi_i^{-1}\left(t,Z_{i,t}\right)\right)} - \frac{1}{2}\frac{\partial}{\partial x}\sigma_i\left(t,\psi_i^{-1}\left(t,Z_{i,t}\right)\right)\right) dt \tag{4.37}$$

$$+ \sum_{j=1}^{n} r_{ij}(t) dw_{j,t} \tag{4.38}$$

with

$$X_t = \psi^{-1}\left(t, Z_t\right). \tag{4.39}$$

This is easily verified using Ito's Lemma. In the case of mixed multiplicative and additive noise (affine noise), the vector equation:

$$dX_t = f(t, X_t)dt + (\sigma_M X_t + \sigma_A)\, dW_t \tag{4.40}$$

with $\sigma_M > 0$ and $\sigma_A > 0$, the transform becomes element-wise in the system. Thus we can consider the one-dimensional case. Since $\psi(t, X_t) = \int \left(\frac{1}{\sigma_M X_t + \sigma_A}\right) dx \mid_{x=X_t} = \frac{\log(\sigma_M X_t + \sigma_A)}{\sigma_M}$, then $X_t = \frac{\exp(\sigma_M Z_t) - \sigma_A}{\sigma_M}$ and

$$\begin{aligned} dZ_t &= \tilde{f}(t, X_t)dt + dW_t \\ \tilde{f}(t, X_t) &= \frac{f(t, X_t)}{\sigma_M X_t + \sigma_A} - \frac{1}{2}\sigma_M \end{aligned} \tag{4.41}$$

provided $\sigma_M X_t$ is guaranteed to be sufficiently different from σ_A to not cause definitional issues. It is common in biological models like chemical reaction networks that $X_t \geq 0$, in which case this is well-defined for any $\sigma_A > 0$ when $\sigma_M > 0$.

For numerical problem solving environments (PSEs), one can make use of this transformation in two ways. Source transformations could transform affine noise SDEs element-wise to solve for the vector Z_t which is the same as X_t if $\sigma_M \neq 0$ and is the transformed X_t otherwise (assuming parameters must be positive). When doing so, references of $X_{i,t}$ must be changed into $\frac{\exp(\sigma_M Z_{i,t}) - \sigma_A}{\sigma_M}$. For example, the affine noise Lotka-Volterra SDE:

$$\begin{aligned} dx &= (ax - bxy)\, dt + (\sigma_M x + \sigma_A)\, dW_t^1 \\ dy &= (-cy + dxy)\, dt + \sigma_{\tilde{A}} dW_t^2 \end{aligned}$$

only has noise on the first term, so this transforms to

$$\begin{aligned} x &= \frac{\exp(\sigma_M z) - \sigma_A}{\sigma_M} \\ dz &= \left(\frac{ax - bxy}{\sigma_M x + \sigma_A} - \frac{1}{2}\sigma_M \right) dt + dW_t^1 \\ dy &= (-cy + dxy)\, dt + \sigma_{\tilde{A}} dW_t^2 \end{aligned}$$

along with the change to the initial condition and can thus be solved with the SRA methods. We note a word of caution that the above transformation only holds when $\sigma_A > 0$ and when $\sigma_A = 0$, the transformation is different, with $X_t = \frac{\exp(Z_t)}{\sigma_M}$ (instead of $\frac{\exp(\sigma_M Z_t)}{\sigma_M}$ which one would get by taking $\sigma_A = 0$).

Instead of performing the transformations directly on the functions themselves, we can modify the SRA algorithm to handle this case as:

$$X_{n+1} = \psi^{-1} \left(\psi\left(X_n\right) + \sum_{i=1}^{s} \alpha_i \tilde{f}\left(t_n + c_i^{(0)} h, H_i^{(0)}\right) + \sum_{i=1}^{s} \left(\beta_i^{(1)} I_{(1)} + \beta_i^{(2)} \frac{I_{(1,0)}}{h} \right) \tilde{g}\left(t_n + c_i^{(1)} h\right) \right) \tag{4.42}$$

with stages

$$H_i^{(0)} = \psi\left(X_n\right) + \sum_{j=1}^{s} A_{ij}^{(0)} \tilde{f}\left(t_n + c_j^{(0)} h, H_j^{(0)}\right) h + \sum_{j=1}^{s} B_{ij}^{(0)} \tilde{g}\left(t_n + c_j^{(1)} h\right) \frac{I_{(1,0)}}{h} \tag{4.43}$$

where ψ is the element-wise function:

$$\psi_i(x) = \begin{cases} \frac{\log(\sigma_{i,M} x + \sigma_{i,A})}{\sigma_{i,M}} & \sigma_{i,M} > 0, \sigma_{i,A} > 0 \\ \frac{\log(x)}{\sigma_{i,M}} & \sigma_{i,M} > 0, \sigma_{i,A} = 0 \\ x & o.w. \end{cases},$$

$$\psi_i^{-1}(z) = \begin{cases} \frac{\exp\left(\sigma_{i,M} z\right)}{\sigma_{i,M}} & \sigma_{i,M} > 0, \sigma_{i,A} > 0 \\ \frac{\exp(z)}{\sigma_{i,M}} & \sigma_{i,M} > 0, \sigma_{i,A} = 0 \\ x & o.w. \end{cases}$$

and

$$\tilde{g}_i(t) = \begin{cases} 1 & \sigma_{i,M} > 0 \\ \sigma_{i,A} & o.w. \end{cases}$$

This can be summarized as performing all internal operations in Z-space (where the equation is additive) but saving each step in X-space.

4.6 Optimized-Stability Order 1.5 SRK Methods with Diagonal Noise

4.6.1 The Stability Equation for Order 1.5 SRK Methods with Diagonal Noise

For diagonal noise, we use the mean-square definition of stability [63]. A method is mean-square stable if $\lim_{n\to\infty} \mathbb{E}\left(|X_n|^2\right) = 0$ on the test equation

$$dX_t = \mu X_t dt + \sigma X_t dW_t. \tag{4.44}$$

In matrix form we can re-write our method as given by

$$X_{n+1} = X_n + \mu h \left(\alpha \cdot H^{(0)}\right) + \sigma I_{(1)} \left(\beta^{(1)} \cdot H^{(1)}\right) + \sigma \frac{I_{(1,1)}}{\sqrt{h}} \left(\beta^{(2)} \cdot H^{(1)}\right) \tag{4.45}$$

$$+ \sigma \frac{I_{(1,0)}}{h} \left(\beta^{(3)} \cdot H^{(1)}\right) + \sigma \frac{I_{(1,1,1)}}{h} \left(\beta^{(4)} \cdot H^{(1)}\right) \tag{4.46}$$

with stages

$$\begin{aligned} H^{(0)} &= X_n + \mu \Delta t A^{(0)} H^{(0)} + \sigma \frac{I_{(1,0)}}{h} B^{(0)} H^{(1)}, \\ H^{(1)} &= X_n + \mu \Delta t A^{(1)} H^{(0)} + \sigma \sqrt{\Delta t} B^{(1)} H^{(1)} \end{aligned} \tag{4.47}$$

where $\hat{X_n}$ is the size s constant vector of X_n.

$$\begin{aligned} H^{(0)} &= \left(I - hA^{(0)}\right)^{-1} \left(\hat{X_n} + \sigma \frac{I_{(1,0)}}{h} B^{(0)} H^{(1)}\right), \\ H^{(1)} &= \left(I - \sigma \sqrt{h} B^{(1)}\right)^{-1} \left(\hat{X_n} + \mu h A^{(1)} H^{(0)}\right) \end{aligned} \tag{4.48}$$

By the derivation in the appendix, we receive the equation

$$\begin{aligned} S = E\left[\frac{U_{n+1}^2}{U_n^2}\right] &= \{1 + \mu h t \left(\alpha \cdot \left[\left(I - \mu \Delta t A^{(0)} - \mu \sigma I_{(1,0)} A^{(1)} B^{(0)} \left(I - \sigma\sqrt{h} B^{(1)}\right)^{-1}\right)^{-1} \left(I + \sigma \frac{I_{(1,0)}}{h} B^{(0)} \left(I - \sigma\sqrt{h} B^{(1)}\right)^{-1}\right)\right]\right) \\ &+\sigma I_{(1)} \left(\beta^{(1)} \cdot \left[\left(I - \sigma\sqrt{h} B^{(1)} - \mu h A^{(1)} \left(I - \mu h A^{(0)}\right)^{-1} \sigma \frac{I_{(1,0)}}{h} B^{(0)}\right)^{-1} \left(I + \mu h A^{(1)} \left(I - \mu h A^{(0)}\right)^{-1}\right)\right]\right) \\ &+\sigma \frac{I_{(1,1)}}{\sqrt{h}} \left(\beta^{(2)} \cdot \left[\left(I - \sigma\sqrt{h} B^{(1)} - \mu h A^{(1)} \left(I - \mu h A^{(0)}\right)^{-1} \sigma \frac{I_{(1,0)}}{h} B^{(0)}\right)^{-1} \left(I + \mu h A^{(1)} \left(I - \mu h A^{(0)}\right)^{-1}\right)\right]\right) \\ &+\sigma \frac{I_{(1,0)}}{h} \left(\beta^{(3)} \cdot \left[\left(I - \sigma\sqrt{h} B^{(1)} - \mu h A^{(1)} \left(I - \mu h A^{(0)}\right)^{-1} \sigma \frac{I_{(1,0)}}{h} B^{(0)}\right)^{-1} \left(I + \mu h A^{(1)} \left(I - \mu h A^{(0)}\right)^{-1}\right)\right]\right) \\ &+\sigma \frac{I_{(1,1,1)}}{h} \left(\beta^{(4)} \cdot \left[\left(I - \sigma\sqrt{h} B^{(1)} - \mu h A^{(1)} \left(I - \mu h A^{(0)}\right)^{-1} \sigma \frac{I_{(1,0)}}{h} B^{(0)}\right)^{-1} \left(I + \mu h A^{(1)} \left(I - \mu h A^{(0)}\right)^{-1}\right)\right]\right)\}^2 \end{aligned} \tag{4.49}$$

We apply the substitutions from the Appendix and let

$$\begin{aligned} z &= \mu h, \\ w &= \sigma\sqrt{h}. \end{aligned} \tag{4.50}$$

In this space, z is the stability variable for the drift term and w is the stability in the diffusion term. Under this scaling $\left(h, \sqrt{h}\right)$, the equation becomes independent of h and thus becomes a function $S(z, w)$ on the coefficients of the SRK method where mean-square stability is achieved when $|S(z, w)| < 1$. The equation $S(z, w)$ in terms of its coefficients for explicit methods ($A^{(i)}$ and $B^{(i)}$ lower diagonal) has millions of terms and is shown in the supplemental Mathematica notebook. Determination of the stability equation for the implicit methods was found to be computationally intractable and is an avenue for further research.

4.6.2 An Optimization Problem for Determination of Coefficients

We wish to determine the coefficients for the diagonal SRK methods which optimize the stability. To do so, we generate an optimization problem which we can numerically solve for the coefficients. To simplify the problem, we let $z, w \in \mathbb{R}$. Define the function

$$f\left(z,w;N,M\right)=\int_{-M}^{M}\int_{-N}^{1}\chi_{S(z,w)\leq1}(z,w)dzdw. \tag{4.51}$$

Notice that for $N, M \to \infty$, f is the area of the stability region. Thus we define the stability-optimized diagonal SRK method as the set of coefficients which achieves

$$\max_{A^{(i)},B^{(i)},\beta^{(i)},\alpha} f(z,w) \tag{4.52}$$

subject to: Order Constraints

However, like with the SRK methods for additive noise, we impose a few extra constraints to add robustness to the error estimator. In all cases we impose $0 < c_i^{(0)}, c_i^{(1)} < 1$. Additionally we can prescribe $c_4^{(0)} = c_4^{(1)} = 1$ which we call the End-C Constraint. Lastly, we can prescribe the ordering constraint $c_1^{(j)} < c_2^{(j)} < c_3^{(j)} < c_4^{(j)}$ which we denote as the Inequality-C Constraint.

The resulting problem is a nonlinear programming problem with 44 variables and 42-48 constraint equations. The objective function is the two-dimensional integral of a discontinuous function which is determined by a polynomial of in z and w with approximately 3 million coefficients. To numerically approximate this function, we calculated the characteristic function on a grid with even spacing dx using a CUDA kernel and found numerical solutions to the optimization problem using the JuMP framework [28] with the NLopt backend [55]. A mixed approach using many solutions of the semi-local optimizer LN_AUGLAG_EQ [23, 6] and fewer solutions from the global optimizer GN_ISRES [96] were used to approximate the optimality of solutions. The optimization was run many times in parallel until many results produced methods with similar optimality, indicating that we likely obtained values near the true minimum.

The parameters N and M are the bounds on the stability region and also represent a trade-off between the stability in the drift and the stability in the diffusion. A method which is optimized when M is small would be highly stable in the case of small noise, but would not be guaranteed to have good stability properties in the presence of large noise. Thus these parameters are knobs for tuning the algorithms for specific situations, and thus we solved the problem for different combinations of N and M to determine different algorithms for the different cases.

4.6.3 Resulting Approximately-Optimal Methods

The coefficients generated for approximately-optimal methods fall into three categories. In one category we have the drift-dominated stability methods where large N and small M was optimized. On the other end we have the diffusion-dominated stability methods where large M and small N was optimized. Then we have the mixed stability methods which used some mixed size choices for N and M. As a baseline, we optimized the objective without constraints on the c_i to see what the "best possible method" would be. When this was done with large N and M, the resulting method, which we name SOSRI, has almost every value of c satisfy the constraints, but with $c_2^{(0)} \approx -0.04$ and $c_4^{(0)} \approx 3.75$. To see if we could produce methods which were more diffusion-stable, we decreased N to optimize more in w but failed to produce methods with substantially enlarged diffusion-stability over SOSRI.

Adding only the inequality constraints on the c_i and looking for methods for drift-dominated stability, we failed to produce methods whose c_i estimators adequately covered the interval. Some of the results did produce stability regions similar to SOSRI but with $c_i^{(0)} < 0.5$ which indicates the method could have problems with error estimation. When placing the equality constraints on the edge c_i, one method, which we label SOSRI2, resulted in similar stability to SOSRI but satisfy the c_i constraints. In addition, this method satisfies $c_3^{(0)} = c_4^{(0)} = 1$ and $c_3^{(1)} = c_4^{(1)} = 1$, a property whose use will be explained in Section 4.6.4. The stability regions for these methods is shown in Figure 4.4.

To look for more diffusion-stable methods, we dropped to $N = 6$ to encourage the methods to expand the stability in the w-plane. However, we could not find a method whose stability region went substantially beyond $[-2, 2]$ in w. This was further decreased to $N = 1$ where methods still could not go substantially beyond $|2|$. Thus we were not able to obtain methods optimized for the diffusion-dominated case. This hard barrier was hit under many different constraint and objective setups and under thousands of optimization runs, indicating there might be a diffusion-stability barrier for explicit methods.

4.6.4 Approximately-Optimal Methods with Stability Detection and Switching Behaviors

In many real-world cases, one may not be able to clearly identify a model as drift-stability bound or diffusion-stability bound, or if the equation is stiff or non-stiff. In fact, many models may switch between such extremes. An example is a model with stochastic switching between different steady states. In this case, we have that the diffusion term $f(t, X_{ss}) \approx 0$ in the area of many stochastic steady states, meaning that while straddling a steady state

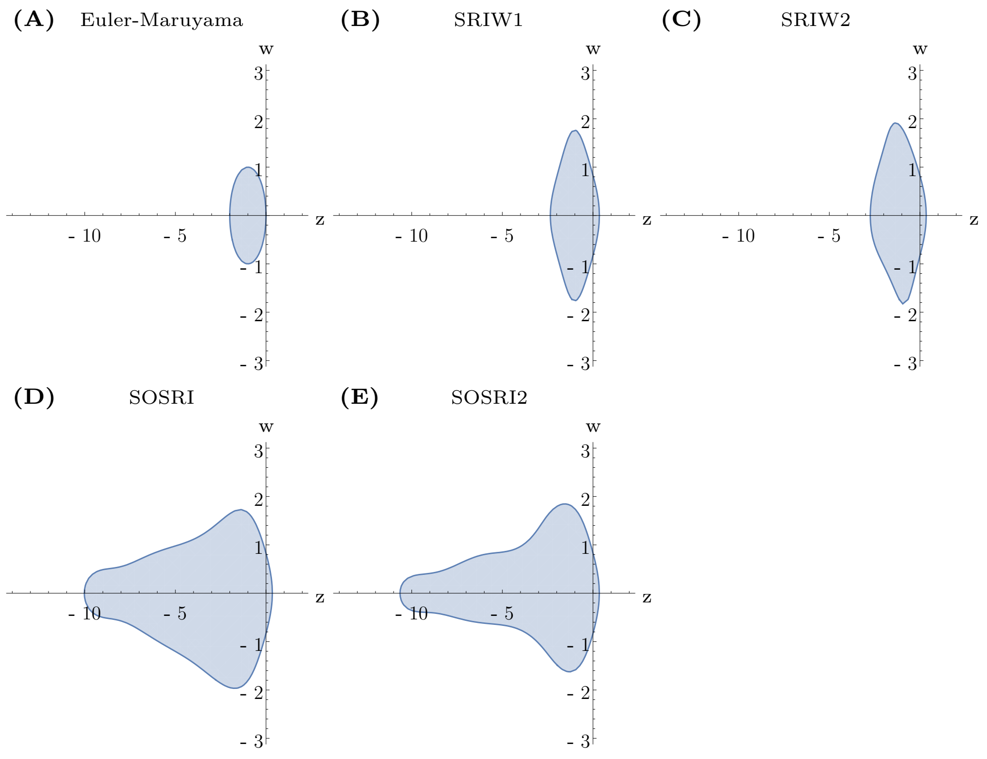


Figure 4.4: **SOSRI Stability Regions**. The stability regions ($S(z, w) \leq 1$)for the previous and SOSRI methods are plotted in the (z, w)-plane. **(A)** Euler-Maruyama. **(B)** SRIW1. **(C)** SRIW2. **(D)** SOSRI. **(E)** SOSRI2

the integration is heavily diffusion-stability dominated and usually non-stiff. However, when switching between steady states, f can be very large and stiff, causing the integration to be heavily drift-stability dominated. Since these switches are random, the ability to adapt between these two behaviors could be key to achieving optimal performance. Given the trade-off, we investigated how our methods allow for switching between methods which optimize for the different situations.

The basis for our method is an extension of a method proposed for deterministic differential equations [102, 105, 42]. The idea is to create a cheap approximation to the dominant eigenvalues of the Jacobians for the drift and diffusion terms. If v is the eigenvector of the respective Jacobian, then for $\|v\|$ sufficiently small,

$$|\lambda_D| \approx \frac{\|f(t,x+v)-f(t,x)\|}{\|v\|}, \tag{4.53}$$

$$|\lambda_N| \approx \frac{\|g(t,x+v)-g(t,x)\|}{\|v\|} \tag{4.54}$$

where $|\lambda_D|$ and $|\lambda_N|$ are the estimates of the dominant eigenvalues for the deterministic and noise functions respectively. We have in approximation that $H_i^{(k)}$ is an approximation for $X_{t+c_i^{(k)}h}$ and thus the difference between two successive approximations at the same time-point, $c_i^{(k)} = c_j^{(k)}$, then the following serves as a local Jacobian estimate:

$$|\lambda_D| \approx \frac{\|f(t+c_i^{(0)}h, H_i^{(0)}) - f(t+c_j^{(0)}h, H_j^{(0)})\|}{\|H_i^{(0)} - H_j^{(0)}\|}, \tag{4.55}$$

$$|\lambda_N| \approx \frac{\|f(t+c_i^{(1)}h, H_i^{(1)}) - f(t+c_j^{(1)}h, H_j^{(1)})\|}{\|H_i^{(1)} - H_j^{(1)}\|} \tag{4.56}$$

If we had already computed a successful step, we would like to know if in the next calculation we should switch methods due to stability. Thus it makes sense to approximate the Jacobian at the end of the interval, meaning $i = s$ and $j = s-1$ where s is the number of stages.

Then if z_{min} is the minimum $z \in \mathbb{R}$ such that z is in the stability region for the method, $\frac{h|\lambda_D|}{z_{min}} > 1$ when the steps are outside the stability region. Because the drift and mixed stability methods do not track the noise axis directly, we instead modify w_{min} to be $\frac{2}{3}$ of the maximum of the stability region in the noise axis.

Hairer noted that, for ODEs, if a RK method has $c_i = c_j = 1$, then it follows that

$$\rho = \frac{\|k_i - k_j\|}{\|g_i - g_j\|} \tag{4.57}$$

where $k_i = f(t + c_i h, g_i)$ is an estimate of the eigenvalues for the Jacobian of f. Given the construction of SOSRI2, a natural extension is

$$|\lambda_D| \approx \frac{\|f\left(t_n + c_4^{(0)}h, H_4^{(0)}\right) - f\left(t_n + c_3^{(0)}h, H_3^{(0)}\right)\|}{\|H_4^{(0)} - H_3^{(0)}\|}, \tag{4.58}$$

$$|\lambda_N| \approx \frac{\|g\left(t_n + c_4^{(0)}h, H_4^{(1)}\right) - g\left(t_n + c_3^{(0)}h, H_3^{(1)}\right)\|}{\|H_4^{(1)} - H_3^{(1)}\|} \tag{4.59}$$

Given that these values are all part of the actual step calculations, this stiffness estimate essentially is free. By comparing these values to the stability plot in Figure 4.2, we use the following heuristic to decide if SOSRI2 is stability-bound in its steps:

1. If $10 > |\lambda_D| > 2.5$, then we check if $h\,|\lambda_N| > \omega$.

2. If $|\lambda_D| < 2.5$, then we check if $h\,|\lambda_N|/2 > \omega$.

The denominator is chosen as a reasonable box approximation to the edge of the stability region. ω is a safety factor: in theory ω is 1 since we divided by the edge of the stability region, but in practice this is only an eigenvalue estimate and thus ω allows for a trade-off between the false positive and false negative rates. If either of those conditions are satisfied, then h is constrained by the stability region. The solver can thus alert the user that the problem is stiff or use this estimate to switch to a method more suitable for stiff equations. In addition, the error estimator gives separate error estimates in the drift and diffusion terms. A scheme could combine these two facts to develop a more robust stiffness detection method, and label the stiffness as either drift or diffusion dominated.

We end by noting that SOSRA2 has the same property, allowing stiffness detection via

$$|\lambda_D| \approx \frac{\|f\left(t_n + c_3^{(0)}h, H_3^{(0)}\right) - f\left(t_n + c_2^{(0)}h, H_2^{(0)}\right)\|}{\|H_3^{(0)} - H_2^{(0)}\|} \tag{4.60}$$

and, employing a similar method as the deterministic case, check for stiffness via the estimate $h\left|\lambda_D\right|/5 > \omega$.

In addition, stiff solvers can measure the maximal eigenvalues directly from the Jacobian. Here we suggest the measure from Shampine [102, 105, 42] of using $\|J\|_\infty$ as a cheap upper bound. For semi-implicit methods like LSRA we only get a stability bound on the drift term, but this should be sufficient since for additive noise diffusive noise instability is not an issue.

4.7 Numerical Results

4.7.1 Convergence Tests

In order to test the efficiency and correctness of the SRA algorithms, we chose to use the additive noise test Equation 4.67. Figure 4.5A demonstrates that the SOSRA and SKenCarp methods achieve the strong order 2.0 on Equation 4.65. To test the convergence of the SRI algorithms, we used the linear test Equation 4.67. Figure 4.5B demonstrates that the SOSRI

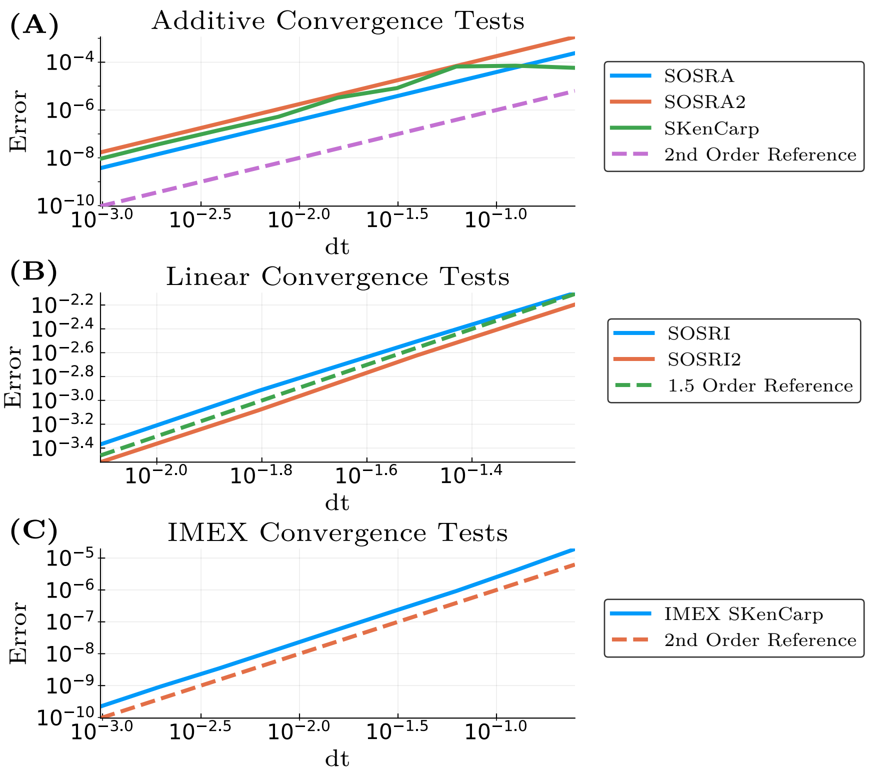


Figure 4.5: **Additive Noise Convergence Tests.** The error is averaged over 1000 trajectories. Shown are the strong l_2 error along the time series of the solution. **(A)** Convergence results on Equation 4.65. The test used a fixed time step $h = 1/2^{-2}$ to $h = 1/2^{-10}$. **(B)** Convergence results on Equation 4.67. The test used a fixed time step $h = 1/2^{-4}$ to $h = 1/2^{-7}$. **(C)** Convergence results on the IMEX Equation 4.69. The test used a fixed time step $h = 1/2^{-2}$ to $h = 1/2^{-10}$.

methods achieve the strong order 1.5 on Equation 4.67. Lastly, we tested the convergence of the IMEX version of the SKenCarp integrator. We defined the split SDE 4.69 as a modification of Equation 4.65 where the f_1 part is solved implicitly and the f_2 part is solved explicitly. Figure 4.5C demonstrates that the IMEX SKenCarp method achieves strong order 2.0 . Note that this does not demonstrate that the method always achieves strong order 1.5 since sufficient conditions for the IMEX pairing are unknown, but it gives numerical evidence that the method can be high order.

4.7.2 SOSRA Numerical Efficiency Experiments

Additive Noise Lotka-Volterra (2 Non-Stiff SDEs)

To test the efficiency we first plotted work-precision [41, 110, 42] diagrams for the SOSRA, SOSRA2, and SKenCarp methods against the SRA1, SRA2, SRA3 [95] methods, and fixed time step Euler-Maruyama method (Milstein is equivalent to Euler-Maruyama in this case [63]). We tested the error and timing on Equation 4.65. In addition, we tested using the Lotka-Volterra equation with additive noise Equation 4.70. Since 4.70 does not have an analytical solution, a reference solution was computed using a low tolerance solution via SOSRA for each Brownian trajectory. The plots show that there is a minimal difference in efficiency between the SRA algorithms for errors in the interval $[10^{-6}, 10^{-2}]$, while these algorithms are all significantly more efficient than the Euler-Maruyama method when the required error is $< 10^{-4}$ (Figure 4.6). The weak error work-precision diagrams show that when using between 100 to 10,000 trajectories, the weak error is less than the sample error in the regime where there is no discernible efficiency difference between the SRA methods. These results show that in the regime of mild accuracy on non-stiff equations, the SOSRA, SOSRA2, and SKenCarp methods are much more efficient than low order methods yet achieve the same efficiency as the non-stability optimized SRA variants. Note that these results also show that the error estimator for adaptivity is highly conservative, generating solutions with around 2 orders of magnitude less error than the tolerance suggests.

Addtive Noise Van Der Pol (2 Stiff SDEs)

To test how efficiently the algorithms could achieve solve stiff equations, we chose to analyze the qualitative results of the driven Van der Pol equation. The driven Van der Pol equation

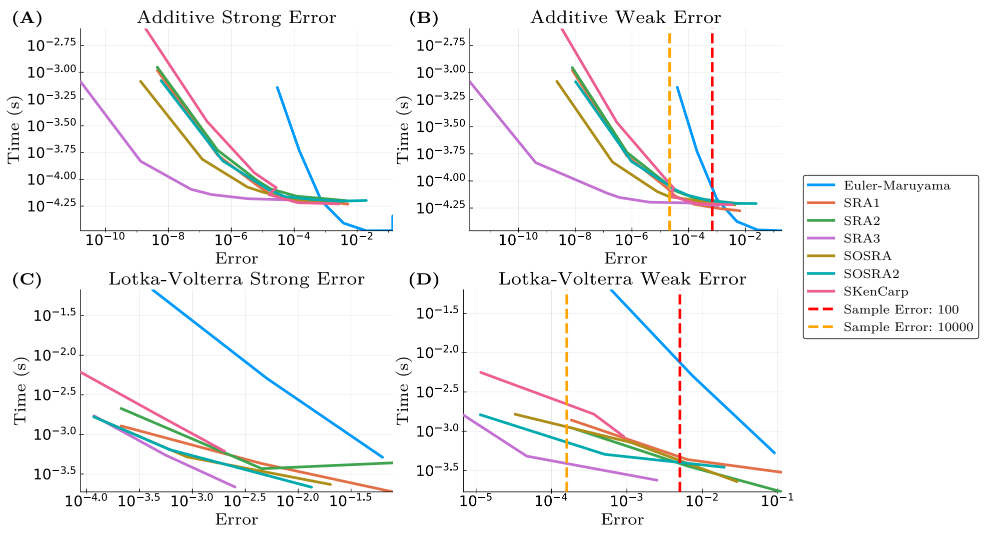


Figure 4.6: **SOSRA Efficiency Tests.** The error was taken as the average of 10,000 trajectories for Equation 4.65 and 100 trajectories for the Lokta-Volterra Equation 4.70. The sample error was determined for the weak error as the normal 95% confidence interval for the mean using the variance of the true solution Equation 4.66 or the variance of the estimated true solutions via low tolerance solutions. The time is the average time to compute a trajectory and is averaged over 1000 runs at the same tolerance or step size. **(A)** Shown are the work-precision plots for the methods on Equation 4.65. Each of the adaptive time-stepping methods solved the problem on the interval using changing values of tolerances, with $tol = abstol = reltol$ starting at 10^2 and ending at 10^{-4} going in increments of 10. The fixed time-stepping methods used time steps of size $h = 1/5^{-1}$ to $h = 1/5^4$, changing the value by factors of 5. The error is the strong l_2 error computed over the time series. **(B)** Same setup as the previous plot but using the weak error at the final time-point. **(C)** Shown are the work-precision plots for the methods on the Equation 4.70. Each of the adaptive time-stepping methods solved the problem on the interval using changing values of tolerances, with $tol = abstol = reltol$ starting at 4^{-2} and ending at 4^{-4} going in increments of 4. The fixed time-stepping methods used time steps of size $h = 1/12^{-2.5}$ to $h = 1/12^{-6.5}$, changing the value by factors of 12. The error is the strong l_2 error computed over the time series. **(D)** Same setup as the previous plot but using the weak error at the final time-point.

is given by Equation 4.71 where μ is the driving factor. As μ increases the equation becomes more stiff. $\mu = 10^6$ is a common test for stiff ODE solvers [43], with lower values used to test the semi-stiff regime for ODEs. For our purposes, we chose $\mu = 10^5$ as a semi-stiff test case. The ODE case, solved using the Tsit5 explicit Runge-Kutta algorithm [116, 92], and demonstrates the mild stiffness which is still well-handled by explicit methods (Figure 4.7A). We extend this model to the driven Van der Pol model with additive noise Equation 4.72 where $\rho = 3.0$ is the noise gain and $dW^{(1)}$ and $dW^{(2)}$ are independent Brownian motions. The solution to this model is interesting because it gives the same qualitative behavior, large bursts when $x(t)$ crosses zero, but in this case the zero crossings are stochastic. Even at high tolerances, ($abstol = 10$,$reltol = 1/2^1$), SOSRA is able to reproduce this qualitative behavior of the low tolerance solutions (Figure 4.7B), and SOSRA2 producing similar results at the same tolerances a factor of two lower. Given the conservativeness of the error estimators shown in previous (and other tests), this case corresponds to roughly two decimal places of accuracy, which is more than sufficient for many phenomenological models. However, even at tolerances of $abstol = 1/2^3$,$reltol = 1/2^3$ SRA3 was unable to reproduce the correct qualitative behavior (Figure 4.7C). Thus we decreased the tolerances by factors of 2 until it was able to reproduce the correct qualitative results (Figure 4.7D). This shows that the SOSRA are more reliable on models with transient stiffness. To test the impact on the run time of the algorithms, each of the algorithms were run 100 times with the tolerance setup that allows them to most efficiently generate correct qualitative results. The run times are shown in Table 4.1, which show that SRA1 takes more than 10 times and SRA3 nearly 4 times as long as the SOSRA methods. In this case the implicit method SKenCarp is the fastest by besting the SOSRA methods by more than 8x while achieving similar qualitative results. This shows that as stiffness comes into play, the SOSRA methods along with the implicit SKenCarp method are more robust and efficient. The fixed time step methods were far less efficient. Adaptive timestepping via rejection sampling was crucial to the success of the SKenCarp method because it required the ability to pull back to a smaller timestep

Algorithm	Run-time (seconds)	Relative Time (vs SKenCarp)
SKenCarp	37.23	1.0x
SOSRA	315.58	8.5x
SOSRA2	394.82	10.6x
SRA3	1385.66	37.2x
SRA1	3397.66	91.3x
Euler-Maruyama	5949.19	159.8x
DISTM $\left(\theta = \frac{1}{2}\right)$	229111.15	6153x

Table 4.1: **SRA Run times on Van der Pol with additive noise.** The additive noise Van der Pol equation was solved 100 times using the respective algorithms at the highest tolerance by powers of two which match the low tolerance solution to plotting accuracy. The fixed time step methods had their Δt determined as the largest Δt in increments of powers of 2 that produced no unstable trajectories. This resulted in $\Delta t = 5e - 8$ for both the Euler-Maruyama the Drift-Implicit Stochastic θ-methods. Note that the total time of the drift-implicit stochastic θ-method and the Euler-Maruyama method were determined by extrapolating the time from a single stable trajectory on $t \in [0, 1]$ due to time constraints. DISTM is the Drift-Implicit Stochastic θ-Method

when Newton iterations diverged, otherwise it resulted in time estimates around 5x slower than SOSRA.

Additive Van Der Pol Stiffness Detection

In addition to testing efficiency, we used this to test the stiffness detection in SOSRA2. Using a safety factor of $\omega = 5$, we added only two lines of code to make the algorithm print out the timings for which the algorithm predicts stiffness. The results on two trajectories were computed and are shown in Figure 4.8. The authors note that the stiffness detection algorithms are surprisingly robust without any tweaking being done and are shown to not give almost any false positives nor false negatives on this test problem. While this safety factor is set somewhat high in comparison to traditional ODE stiffness detection, we note

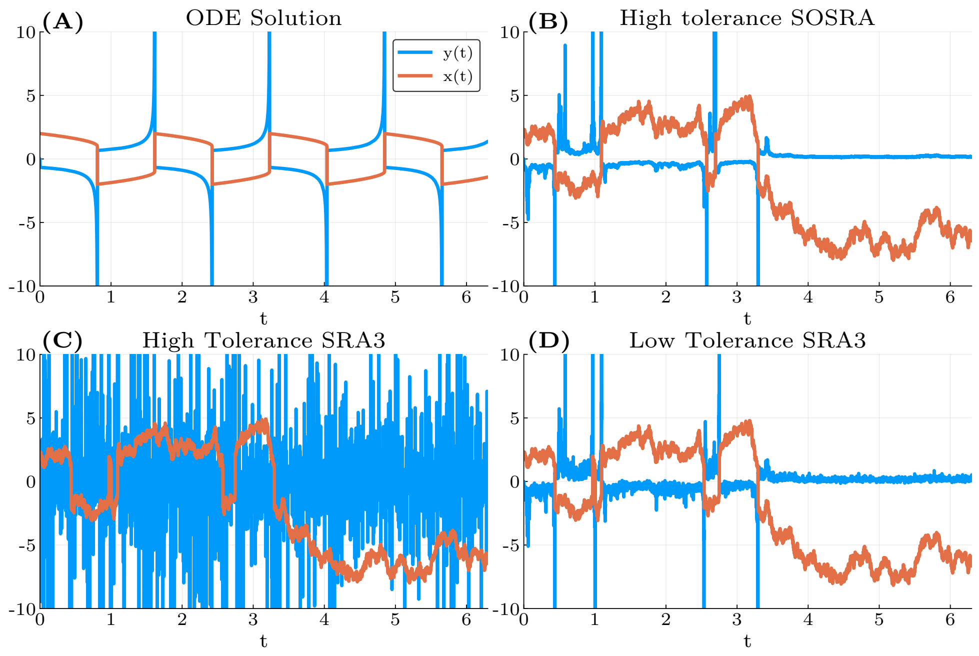


Figure 4.7: **Representative trajectories for solutions to the Van der Pol equations with additive noise.** Each of these trajectories are solved with the same underlying Brownian process. **(A)** The solution to the ODE with the explicit Runge-Kutta method Tsit5. **(B)** The solution to the SDE with tolerance $abstol = 1, reltol = 1/2^1$ from SOSRA. **(C)** Solution to the SDE with tolerances $abstol = 2^{-3}, reltol = 2^{-3}$ with SRA3. **(D)** Solution to the SDE with tolerances $abstol = 2^{-6}, reltol = 2^{-4}$ with SRA3.

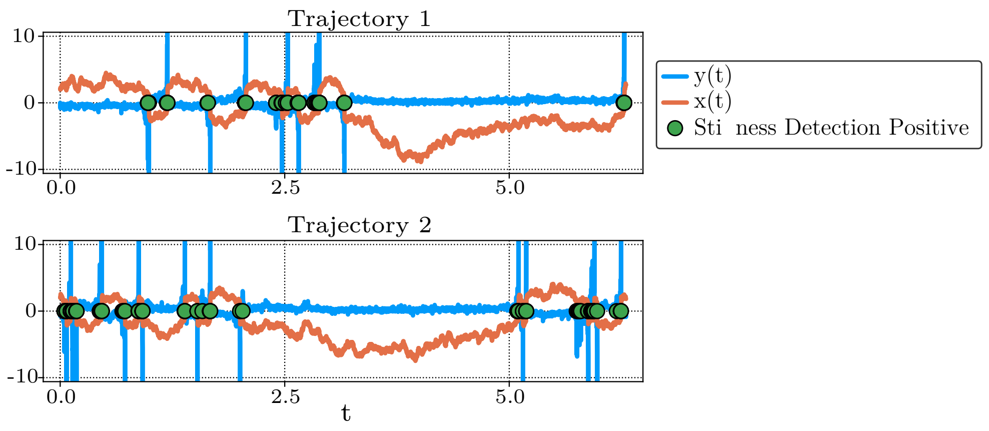


Figure 4.8: **Stiffness detection in the Van der Pol equations with additive noise Equation 4.72.** Two representative trajectories to Equation 4.7 are plotted. The green dots indicate time-points where the stiffness detection algorithm detected stiffness.

that these algorithms were designed to efficiently handle mild stiffness and thus we see it as a benefit that they only declare stiffness when it appears to be in the regime which is more suitable for implicit methods.

4.7.3 SOSRI Numerical Efficiency Experiments

Multiplicative Noise Lotka-Volterra (2 Non-Stiff SDEs)

To test the efficiency we plotted a work-precision diagram with SRIW1, SOSRI, SOSRI2, and the fixed time step Euler-Maruyama and a Runge-Kutta Milstein schemes for Equation 4.67 and the multiplicative noise Lotka-Volterra Equation 4.73. As with Equation 4.70, Equation 4.73 does not have an analytical solution so a reference solution was computed using a low tolerance solution via SOSRI for each Brownian trajectory. The results show that there is a minimal difference in efficiency between the SRI algorithms for errors over the interval $[10^{-6}, 10^{-2}]$, while these algorithms are all significantly more efficient than the lower order algorithms when the required error is $< 10^{-2}$ (Figure 4.9A-D). The weak error work-precision diagrams show that when using between 100 to 10,000 trajectories, the weak error is less than the sample error in the regime where there is no discernible efficiency difference between the SRI methods.These results show that in the regime of mild accuracy on non-stiff equations, these methods are much more efficient than low order methods yet achieve the same efficiency as the non-stability optimized SRI variants. Note that these results also show the conservativeness of the error estimators.

Epithelial-Mesenchymal Transition (EMT) Model (20 Pathwise Stiff SDEs)

To test the real consequences of the enhanced stability, we use the Epithelial-Mesenchymal Transition (EMT) model of 20 pathwise stiff reaction equations introduced in [48], studied as a numerical test in [90], and written in Section 4.9.2. In the previous work it was noted that $t \in [0, 1]$ was a less stiff version of this model. Thus we first tested the speed that the methods could solve for 10,000 trajectories with no failures due to numerical instabilities. The tolerances were tuned for each method by factors of 2 and finding the largest values that were stable. Since SOSRI demonstrated that its stability is much higher than even SOSRI2, we show the effect of tolerance changes on SOSRI as well. The results show that at similar tolerances the SOSRI method takes nearly 5x less time than SRIW1 (Table 4.2). However, there is an upper bound on the tolerances before the adaptivity is no longer able to help keep the method stable. For SRIW1, this bound is much lower, causing it to run more than 15x slower than the fastest SOSRI setup. Interestingly SOSRI2 required a higher tolerance than SRIW1 but was 3x faster than SRIW1's fastest setup. We note that SOSRI's highest relative tolerance $2^{-7} \approx 7 \times 10^{-3}$ is essentially requiring 4 digits of accuracy (in strong error) when considering the conservativeness of the error estimator, which is far beyond the accuracy necessary in many cases. Lastly, we note that the SOSRI method is able to solve for 10,000

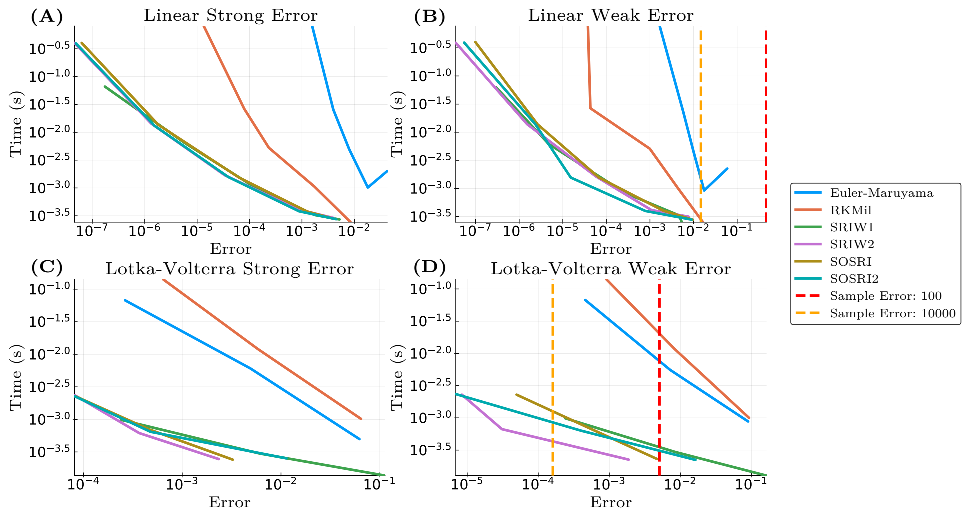


Figure 4.9: **SOSRI efficiency on non-stiff test equations.** The error was taken as the average of 10,000 trajectories for the Equation 4.67 and 100 trajectories for the Lokta-Volterra Equation 4.73. The sample error was determined for the weak error as the normal 95% confidence interval for the mean using the variance of the true solution Equation 4.68 or the variance of the estimated true solutions via low tolerance solutions. The time is the average time to compute a trajectory and is averaged over 1000 runs at the same tolerance or step size.**(A)** Shown are the work-precision plots for the methods on Equation 4.67. Each of the adaptive time-stepping methods solved the problem on the interval using changing values of tolerances, with $tol = abstol = reltol$ starting at 10^{-1} and ending at 10^{-5} going in increments of 10. The fixed time-stepping methods used time steps of size $h = 5^{-2}$ to $h = 5^{-7}$, changing the value by factors of 5. The error is the strong l_2 error computed over the time series. **(B)** Same setup as the previous plot but using the weak error at the final time-point. **(C)** Shown are the work-precision plots for the methods on the multiplicative noise Lotka-Volterra Equation 4.73. Each of the adaptive time-stepping methods solved the problem on the interval using changing values of tolerances, with $tol = abstol = reltol$ starting at 4^{-2} and ending at 4^{-4} going in increments of 4. The fixed time-stepping methods used time steps of size $h = 1/12^{-2.5}$ to $h = 1/12^{-6.5}$, changing the value by factors of 12. The error is the strong l_2 error computed over the time series. **(D)** Same setup as the previous plot but using the weak error at the final time-point.

Algorithm	Abstol	Reltol	Run-time (seconds)	Relative Time (vs SOSRI)
SOSRI	2^{-7}	2^{-4}	2.62	1.0x
SOSRI	2^{-7}	2^{-6}	2.75	1.0x
SOSRI	2^{-12}	2^{-15}	8.78	3.3x
SOSRI	2^{-13}	2^{-7}	3.05	1.2x
SOSRI2	2^{-12}	2^{-15}	8.69	3.3x
SOSRI2	2^{-13}	2^{-11}	5.56	2.2x
SRIW1	2^{-13}	2^{-7}	15.16	5.8x
Euler-Maruyama			169.96	64.8x
Runge-Kutta Milstein			182.59	69.6x
Fixed Time-step SRIW1			424.30	161.7x
DISTM $\left(\theta = \frac{1}{2}\right)$			8912.91	3396x

Table 4.2: **SRI times for the the EMT model on $t \in [0,1]$.** The equations were solved 10,000 times with the given tolerances to completion and the elapsed time was recorded. The fixed time step methods had their Δt determined as the largest Δt in increments of powers of 2 that produced no unstable trajectories, as shown in [90]. DISTM is the Drift-Implicit Stochastic θ-Method

stable trajectories more than 60x faster than any of the tested fixed time step methods.

We then timed the run time to solve 10 trajectories in the $t \in [0, 500]$ case (Table 4.3). This time we found the optimal tolerance in terms of powers of 10. Once again, SRIW1 needed a lower tolerance than is necessary in order to stay stable. SOSRI is able to solve the problem only asking for around $tol = 10^{-2}$, while the others require more (especially in absolute tolerance as there is a stiff reactant whose values travel close to zero). One interesting point to note is that at similar tolerances both SOSRI and SOSRI2 receive similar timings and both over 6 times faster than the fastest SRIW1 tolerance setup. Both are nearly twice as fast as SRIW1 when matching tolerances as well. Given the conservativeness of the error estimators generally being around 2 orders of magnitude more precise than the local error estimate, the low tolerance solutions are accurate enough for many phenomenological experiments and thus present a good speedup over previous methods. The timings for Euler-Maruyama and Runge-Kutta Milstein schemes are omitted since the tests were unable to finish. From the results of [90] we note that the average dt for SRIW1 on the edge of its

Algorithm	Abstol	Reltol	Run-time (seconds)	Relative Time (vs SOSRI)
SOSRI	10^{-2}	10^{-2}	22.47	1.0x
SOSRI	10^{-4}	10^{-4}	73.62	3.3x
SOSRI	10^{-5}	10^{-3}	89.19	4.0x
SOSRI2	10^{-4}	10^{-4}	76.12	3.4x
SOSRI2	10^{-5}	10^{-3}	121.75	5.4x
SRIW1	10^{-5}	10^{-3}	147.89	6.6x
DIRKM $\left(\theta = \frac{1}{2}\right)$			7378.55	328.3x
DIEM $\left(\theta = \frac{1}{2}\right)$			8796.47	391.4x

Table 4.3: **SRI times for the the EMT model on $t \in [0, 500]$.** The equations were solved 10 times with the given tolerances to completion and the elapsed time was recorded. The fixed timestep methods had their Δt chosen by incrementing by 10^{-5} until 10 consecutive trajectories were stable. Drift-Implicit Euler Maruyama (DIEM) had $\Delta t = \frac{1}{60000}$ and Drift-Implicit Runge-Kutta Milstein (DIRKM) had $\Delta t = \frac{1}{50000}$.

stability had that the smallest dt was approximately 10^{-11}. The stability region for fixed step-size Euler-Maruyama is strictly smaller than SRIW1 (Figure 4.4) which suggests that it would require around 5×10^{12} time steps (with Runge-Kutta Milstein being similar) to solve to $t = 500$. Thus, given it takes on our setup extrapolating the time given 170 seconds for 2^{20} steps, this projects to around 1.6×10^{8} seconds, or approximately 5 years.

Retinoic Acid Stochastic Partial Differential Equation Model (6x20x100 Semi-Stiff SDEs)

As another test we applied the methods to a method of lines discretization of a stochastic partial differential equation (SPDE) describing the spatial regulation of the zebrafish hindbrain via retinoic acid signaling (Section 4.9.2) [93]. The discretization results in a system of $6 \times 20 \times 100$ SDEs. Starting from an initial zero state, a concentration gradient emerges over $t \in [0, 500]$. Each of the methods solved the problem at the highest tolerance that was stable giving the results in Table 4.4. Time stepping for this problem is heavily limited by the high diffusion constant which results in a strict CFL condition for the 2nd order finite

Algorithm	Abstol	Reltol	Run-time (seconds)	Relative Time (vs SOSRI)
SOSRI	10^{-1}	10^{-2}	700.76	1.0x
SOSRI2	10^{-3}	10^{-3}	1016.61	1.5x
Euler-Maruyama			1758.85	2.5x
SRIW1	10^{-5}	10^{-3}	4205.52	6.0x

Table 4.4: **SRI times for the the retinoic acid SPDE model on $t \in [0, 500]$.** The equations were solved twice with the given tolerances to completion and the elapsed time was recorded. The tolerances were chosen as the highest pair of tolerances which did not diverge (going up by powers of 10). Note that none of the cases did the two timings vary by more than 1% of the total run time. Euler-Maruyama used time steps of $\Delta t = 1/20000$ since we note that at $\Delta t = 1/10000$ approximately half of the trajectories (simulating 10) were unstable.

difference discretization that is used (in the PDE sense), making this problem's stepping stability-bound for explicit methods. Because of this stiffness in the real axis, we found that the previous high order adaptive method SRIW1 did not perform well on this problem in comparison to Euler-Maruyama because the drift term is expensive and the extra function calls outweighed the slightly larger timesteps. However, the enhanced stability of the SOSRI and SOSRI2 methods allowed for much larger time steps while keeping the same number of f calls per step, resulting in a more efficient solution when high accuracy is not necessary. We note that the drift-implicit stochastic θ-method and drift implicit θRunge-Kutta Milstein methods were too inefficient to estimate since their time steps were constrained to be near that of the Euler-Maruyama equation due to divergence of the Newton iterations. This SPDE could also be solved via SKenCarp by using the transformation of Section 4.5, but from experiments on the PDE we note that efficient solution of the implicit equations would require using a preconditioned Krylov method due to the size of the system and thus it is left for future investigation.

4.8 Discussion

In this work we derived stability-optimized SRK methods for additive and diagonal noise equations, and used a transformation to allow the additive noise methods to solve affine noise problems. Many other equations can be reduced to the additive noise case as well using the same means. Importantly, our derivation methods utilized heavy computational tools in order to approximately optimize otherwise intractable equations. This same method of derivation can easily be scaled up to higher orders, and by incorporating the coefficients for higher conditions, efficiency can be optimized as well by adding the norm of the principle error coefficients to the optimization function. The majority of the search was performed using global optimizers in massive parallel using a hand-optimized CUDA kernel for the numerical integral of the characteristic function, replacing man-hours with core-hours and effectively optimizing the method. The clear next steps are to find SRA and SRI methods with minimal error estimates and sensible stability regions for the cases in which lower strong error matters, and similar optimizations on SRK methods developed for small noise problems. We note that high strong order methods were investigated because of their better trajectory-wise convergence, allowing for a more robust solution and error estimation since our application to transiently pathwise stiff equations requires such properties.

In this work we also derived L-stable methods for additive (and thus multiplicative and affine) noise equations, and computationally could not find an A-B-L stable method. While our method does not prove that no 2-stage A-B-L method exists, we have at least narrowed down its possibility. Additionally an extension of a well-known ESDIRK method to additive noise was developed. These ESDIRK methods have an extension which allows for mass-matrices in the problem formulation. Using singular mass matrices, these methods also present themselves as integrators for a form of SDAEs with deterministic constraints. This method has an implicit-explicit (IMEX) extension and the stochastic extension was compatible with both tableaus. We showed that this IMEX version of the method could numerically converge at order 2.0 on a test problem (matching the other SRA methods), indicating that it may achieve the sufficient condition. As an adaptive high order IMEX method, the ODE version of the method is a common choice for large discretizations of PDEs. Thus this method could present itself as a potentially automatic and efficient option for discretizations of large affine noise SPDEs by being able to use a low number of time steps while minimizing the amount of work required to solve the implicit equation. We note that adaptivity along with efficient stage predictors was required to be more efficient than the common stochastic theta methods since divergence of quasi-Newton steps can be common if care is not taken. After engineering the method with all of the components together, the benchmark results showed large efficiency gains over both the previous drift-implicit and stability-optimized explicit methods. While previous literature questioned the applicability of L-stable integrators to stochastic differential equations due to high error in the slow variables [70], our computations show that this analysis may be mislead by analyzing strong order 0.5 methods. With our higher strong order methods we see sufficiently accurate results on real stiff problems, and this is greatly helped by time stepping adaptivity.

The main caveat for our methods is the restrictions on the form of noise. While we have shown that an enlarged class of problems (affine noise) can handled by the integrators for additive noise problems, this is still a very special case in the scope of possible SDEs. Diagonal noise is a much expanded scope but is still constrained, and our implicit methods were only derived for the additive noise case. Further research should focus on the expansion of this these techniques to high order adaptive ESDIRK diagonal noise integrators. In addition, when g is non-zero a "diagonal noise" problem over the complex plane does not have diagonal noise (due to the mixing of real and complex parts from complex multiplication, and reinterpretation as a $2n$ real system). Thus these methods are not applicable to problems defined in the complex plane with complex Wiener processes. Development of similar integrators for commutative noise problems could allow for similar performance benefits on such problems and is a topic for future research.

Additionally, we were not able to sufficiently improve the stability along the noise axis with our explicit diagonal noise methods. However, this is likely due to explicitness in the noise term. Recent research has shown that step splitting which utilize a predicted step in the diffusion calcuation can significantly improve the stability of a method [62, 121]. Given this, we conjecture that a form of predictor-correction, such as:

$$X_{n+1} = X_n + \sum_{i=1}^{s} \alpha_i f\left(t_n + c_i^{(0)} h, H_i^{(0)}\right) \tag{4.61}$$

$$+ \sum_{i=1}^{s} \left(\beta_i^{(1)} I_{(1)} + \beta_i^{(2)} \frac{I_{(1,1)}}{\sqrt{h}} + \beta_i^{(3)} \frac{I_{(1,0)}}{h} + \beta_i^{(4)} \frac{I_{(1,1,1)}}{h}\right) g\left(t_n + c_i^{(1)} h\right) \tag{4.62}$$

with stages

$$\overline{H_i^{(1)}} = X_n + \sum_{j=1}^{s} A_{ij}^{(1)} f\left(t_n + c_j^{(0)} h, H_j^{(0)}\right) h + \sum_{j=1}^{s} B_{ij}^{(1)} g\left(t_n + c_j^{(1)} h, H_i^{(1)}\right) \sqrt{h} \tag{4.63}$$

$$H_i^{(0)} = X_n + \sum_{j=1}^{s} A_{ij}^{(0)} f\left(t_n + c_j^{(0)} h, H_j^{(0)}\right) h + \sum_{j=1}^{s} B_{ij}^{(0)} g\left(t_n + c_j^{(1)} h, \overline{H_i^{(1)}}\right) \frac{I_{(1,0)}}{h} \tag{4.64}$$

$$H_i^{(1)} = X_n + \sum_{j=1}^{s} A_{ij}^{(1)} f\left(t_n + c_j^{(0)} h, H_j^{(0)}\right) h + \sum_{j=1}^{s} B_{ij}^{(1)} g\left(t_n + c_j^{(1)} h, \overline{H_i^{(1)}}\right) \sqrt{h}$$

could improve the noise stability of the method while keeping explicitness and the same tableau. However, proper convergence and stability analysis would require significant effort.

Our timings show that the current high order SRK methods are stability-bound and that when scientific studies are only looking for small amounts of accuracy in stochastic simulations, most of the computational effort is lost to generating more accurate than necessary solutions in order to satisfy stability constraints. For additive noise problems we were able to obtain solutions about 5x-30x faster and for diagonal noise approximately 6x than the current adaptive methods (SRA1, SRA3, SRIW1), while common methods like Euler-Maruyama and Drift-Implicit θ Runge-Kutta Milstein were in many cases hundreds of times slower or in many cases could not even finish. We have also shown that these methods are very robust even at high tolerances and have a tendency to produce the correct qualitative results on semi-stiff equations (via plots) even when the user chosen accuracy is low. Given that the required user input is minimal and work over a large range of stiffness, we see these as very strong candidates for default general purpose solvers for problem-solving environments such as MATLAB and Julia since they can easily and efficiently produce results which are sufficiently correct. Due to a choice in the optimization, the SOSRA and SOSRA2 methods are not as efficient at low tolerances as SRA3, so SRA3 should be used when high accuracy is necessary (on additive or affine noise problems). However, in many cases like integrating to find steady distributions of bistable parameter regimes or generating trajectories of phonomenological models, this ability to quickly get a more course estimate is valuable.

The stiffness detection in SDEs is a novel addition which we have demonstrated can act very robustly. It has a control parameter ω which can be used to control the false positive and false negative rate as needed. Note that stiff methods can achieve similar largest eigenvalue estimates directly from the Jacobians of f (and g) given that the methods are implicit (or in the case of Rosenbrock methods, the Jacobian must still be computed), and thus this can be paired with a stiff solver to allow for automatic switching between stiff and non-stiff solvers. Given that the cost for such stiffness checks is minimal and the demonstrated efficiency of the implicit methods on stiff equations, we are interested in future studies on the efficiency of such composite method due to the stochastic nature of stiffness in SDEs.

4.9 Extended Information

4.9.1 Test Equations

Additive Noise Test Equation

$$dX_t = \left(\frac{\beta}{\sqrt{1+t}} - \frac{1}{2\left(1+t\right)}X_t\right)dt + \frac{\alpha\beta}{\sqrt{1+t}}dW_t, \;\; X_0 = \frac{1}{2}, \tag{4.65}$$

where $\alpha = \frac{1}{10}$ and $\beta = \frac{1}{20}$ with true solution

$$X_t = \frac{1}{\sqrt{1+t}}X_0 + \frac{\beta}{\sqrt{1+t}}\left(t + \alpha W_t\right). \tag{4.66}$$

Diagonal Noise Test Equation

$$dX_t = \alpha X_t dt + \beta X_t dW_t \;\; X_0 = \frac{1}{2}, \tag{4.67}$$

where $\alpha = \frac{1}{10}$ and $\beta = \frac{1}{20}$ with true solution

$$X_t = X_0 e^{\left(\beta - \frac{\alpha^2}{2}\right)t + \alpha W_t}. \tag{4.68}$$

Split Additive Test Equation

$$dX_t = \left(f_1(t, X_t) + f_2(t, X_t)\right)dt + \frac{\alpha\beta}{\sqrt{1+t}}dW_t, \;\; X_0 = \frac{1}{2}, \tag{4.69}$$

where

$$f_1(t, X_t) = \frac{\beta}{\sqrt{1+t}}$$
$$f_2(t, X_t) = -\frac{1}{2\left(1+t\right)}X_t$$

with true solution Equation 4.66.

4.9.2 Additive Noise Lotka-Volterra

$$\begin{aligned} dx &= (ax - bxy)\,dt + \sigma_A dW_t^1 \\ dy &= (-cy + dxy)\,dt + \sigma_A dW_t^2 \end{aligned} \tag{4.70}$$

where $a = 1.5$, $b = 1$, $c = 3.0$, $d = 1.0$, $\sigma_A = 0.01$.

Additive Noise Van Der Pol

The driven Van Der Pol equation is

$$\begin{aligned} dy &= \mu((1 - x^2)y - x)dt \\ dx &= ydt \end{aligned} \tag{4.71}$$

The additive noise variant is

$$\begin{aligned} dy &= \mu((1 - x^2)y - x)dt + \rho dW_t^{(1)} \\ dx &= y + \rho dW_t^{(2)} \end{aligned} \tag{4.72}$$

Multiplicative Noise Lotka-Volterra

$$\begin{aligned} dx &= (ax - bxy)\,dt + \sigma_A dW_t^1 \\ dy &= (-cy + dxy)\,dt + \sigma_A dW_t^2 \end{aligned} \tag{4.73}$$

where $a = 1.5$, $b = 1$, $c = 3.0$, $d = 1.0$, $\sigma_A = 0.01$.

Epithelial-Mesenchymal Transition Model

The Epithelial-
Mesenchymal Transition (EMT) 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 [48].

$$\begin{aligned}
A &= \left(\left([TGF] + [TGF0](t)\right)/J0_{snail}\right)^{n0_{snail}} + \left([OVOL2]/J1_{snail}\right)^{n1_{snail}} \\
\frac{d\,[snail1]_t}{dt} &= k0_{snail} + k_{snail}\frac{\left(\left([TGF] + [TGF0](t)\right)/J0_{snail}\right)^{n0_{snail}}}{(1+A)\left(1+[SNAIL]/J2_{snail}\right)} \\
&- kd_{snail}\left([snail1] - [SR]\right) - kd_{SR}\,[SR] \\
\frac{d\,[SNAIL]}{dt} &= k_{SNAIL}\left([snail1] - [SR]\right) - kd_{SNAIL}\,[SNAIL] \\
\frac{d\,[miR34]}{dt} &= kO_{34} + \frac{k_{34}}{1 + \left([SNAIL]/J1_{34}\right)^{n1_{34}} + \left([ZEB]/J2_{34}\right)^{n2_{34}}} \\
&- kd_{34}\left([miR34] - [SR]\right) - \left(1-\lambda_{SR}\right)kd_{SR}\,[SR] \\
\frac{d\,[SR]}{dt} &= Tk\left(K_{SR}\left([snail1] - [SR]\right)\left([miR34] - [SR]\right) - [SR]\right) \\
\frac{d\,[zeb]}{dt} &= k0_{zeb} + k_{zeb}\frac{\left([SNAIL]/J1_{zeb}\right)^{n1_{zeb}}}{1 + \left([SNAIL]/J1_{zeb}\right)^{n1_{zeb}} + \left([OVOL2]/J2_{zeb}\right)^{n2_{zeb}}} \\
&- kd_{zeb}\left([zeb] - \sum_{i=1}^{5} C_5^i\,[ZR]\right) - \sum_{i=1}^{5} kd_{ZR_i} C_5^i\,[ZR_i] \\
\frac{d\,[ZEB]}{dt} &= k_{ZEB}\left([zeb] - \sum_{i=1}^{5} C_5^i\,[ZR_i]\right) - kd_{ZEB}\,[ZEB] \\
\frac{d\,[miR200]}{dt} &= k0_{200} + \frac{k_{200}}{1 + \left([SNAIL]/J1_{200}\right)^{n1_{200}} + \left([ZEB]/J2_{200}\right)^{n2_{200}}} \\
&- kd_{200}\left([miR200] - \sum_{i=1}^{5} iC_5^i\,[ZR_i] - [TR]\right) \\
&- \sum_{i=1}^{5}\left(1-\lambda_i\right)kd_{ZR_i} C_5^i i\,[ZR_i] - \left(1-\lambda_{TR}\right)kd_{TR}\,[TR] \\
\frac{d\,[ZR_1]}{dt} &= Tk\left(K_1\left([miR200] - \sum_{i=1}^{5} iC_5^i\,[ZR_i] - [TR]\right)\right.
\end{aligned}$$

$$\left(\left[zeb\right]-\sum_{i=1}^{5}C_5^i\left[ZR_i\right]\right)-\left[ZR_1\right]\Bigg)$$

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

where

$$\begin{aligned}
\sum_{i=1}^{5}iC_5^i\left[ZR_i\right] &= 5\left[ZR_1\right]+20\left[ZR_2\right]++30\left[ZR_3\right]+20\left[ZR_4\right]+5\left[ZR_5\right],\\
\sum_{i=1}^{5}C_5^i\left[ZR_i\right] &= 5\left[ZR_1\right]+10\left[ZR_2\right]+10\left[ZR_3\right]+5\left[ZR_4\right]+\left[ZR_5\right],\\
\left[TGF0\right](t) &= \begin{cases}\frac{1}{2} & t>100\\ 0 & o.w.\end{cases}
\end{aligned}$$

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	k_{Op}	10
						kd_{Op}	10

Table 4.5: **Table of Parameter Values for the EMT Model.**

The parameter values are given in Table 4.5.

Retinoic Acid SPDE Model

$$\begin{aligned}
d\,[RA_{out}] &= \left(\beta(x) + D\Delta\,[RA_{out}] - b\,[RA_{out}] + c\,[RA_{in}]\right) dt + \sigma_{RA_{out}}\, dW_t^{out} \\
d\,[RA_{in}] &= \left(b\,[RA_{out}] + \delta\,[BP]\,[RA-RAR] - \left(\gamma\,[BP] + \eta + \frac{\alpha\,[RA-RAR]}{\omega + [RA-RAR]} - c\right)[RA_{in}]\right) dt \\
d\,[RA-BP] &= \left(\gamma\,[BP]\,[RA_{in}] + \lambda\,[BP]\,[RA-RAR] - (\delta + \nu\,[RAR])\,[RA-BP]\right) dt \\
d\,[RA-RAR] &= \left(\nu\,[RA-BP]\,[RAR] - \lambda\,[BP]\,[RA-RAR]\right) dt + \sigma_{RA-RAR}\,[RA-RAR]\, dW_t^{RA-RAR} \\
d\,[BP] &= \left(a - \lambda\,[BP]\,[RA-RAR] - \gamma\,[BP]\,[RA_{in}] + (\delta + \nu\,[RAR])\,[RA-BP] - u\,[BP] + \frac{d\,[RA-RAR]}{e + [RA-RAR]}\right) dt \\
d\,[RAR] &= \left(\zeta - \nu\,[RA-BP]\,[RAR] + \lambda\,[BP]\,[RA-RAR] - r\,[RAR]\right) dt
\end{aligned}$$

where $\beta(x) = \beta_0 H(x-40)$ with H the Heaviside step function and $x = 40$ is the edge of

Parameter	Value	Parameter	Value	Parameter	Value
$\sigma_{RA_{in}}$,σ_{RA-RAR},$\sigma_{RA_{out}}$	0.1	ω	100	u	0.01
b	0.17	γ	3.0	d	0.1
α	10000	δ	0.0013	e	1
β_0	1	η	0.0001	a	1
c	0.1	r	0.0001	ζ	0.02
ν	0.85	λ	0.85	D	250.46

Table 4.6: **Table of Parameter Values for the RA SPDE Model.**

retinoic acid production [93]. The space was chosen as $[-100, 400] \times [0, 100]$ with $\Delta x = \Delta y = 5$. The boundary conditions were no-flex on every side except the right side which had leaky boundary conditions with parameter $kA = 0.002$, though full no-flux does not noticably change the results. The parameter values are given in Table 4.6.

4.9.3 SKenCarp, SOSRA, and SOSRI Tableaus

All entries not listed are zero.

SKenCarp Exact Values

$$
\begin{aligned}
K_1 &= 87294609440832483406992237 \\
K_2 &= -53983406399371387722712393713535786276 \\
K_3 &= 26826820\sqrt{6853072660943221216270384658311461343029149665543510113394397} \\
K_4 &= \frac{K_1\left(K_2 - K_3\right)}{4868738516734691891458097} \\
B_{2,1}^{(0)} &= \frac{K_4 - 354038415192410790619483213666362001932}{210758174113231167877981435258781706648}, \\
B_{4,3}^{(0)} &= \frac{K_2 - K_3}{8606625878152317177894269252900546591}, \\
B_{i,j}^{(0)} &= 0\, o.w.
\end{aligned}
$$

SOSRA

Coefficient	Value	Coefficient	Value
α_1	0.2889874966892885	$\beta_3^{(1)}$	0.27753845684143835
α_2	0.6859880440839937	$\beta_1^{(2)}$	0.4237535769069274
α_3	0.025024459226717772	$\beta_2^{(2)}$	0.6010381474428539
$c_1^{(0)}$	0	$\beta_3^{(2)}$	-1.0247917243497813
$c_2^{(0)}$	0.6923962376159507	$A_{2,1}^{(0)}$	0.6923962376159507
$c_3^{(0)}$	1	$A_{3,1}^{(0)}$	-3.1609142252828395
$c_1^{(1)}$	0	$A_{3,2}^{(0)}$	4.1609142252828395
$c_2^{(1)}$	0.041248171110700504	$B_{2,1}^{(0)}$	1.3371632704399763
$c_3^{(1)}$	1	$B_{3,1}^{(0)}$	1.442371048468624
$\beta_1^{(1)}$	-16.792534242221663	$B_{3,2}^{(0)}$	1.8632741501139225
$\beta_2^{(1)}$	17.514995785380226		

SOSRA2

Coefficient	Value	Coefficient	Value
α_1	0.4999999999999998	$\beta_3^{(1)}$	0.07561967854316998
α_2	-0.9683897375354181	$\beta_1^{(2)}$	1
α_3	1.4683897375354185	$\beta_2^{(2)}$	-0.8169981105823436
$c_1^{(0)}$	0	$\beta_3^{(2)}$	-0.18300188941765633
$c_2^{(0)}$	1	$A_{2,1}^{(0)}$	1
$c_3^{(0)}$	1	$A_{3,1}^{(0)}$	0.9511849235504364
$c_1^{(1)}$	0	$A_{3,2}^{(0)}$	0.04881507644956362
$c_2^{(1)}$	1	$B_{2,1}^{(0)}$	0.7686101171003622
$c_3^{(1)}$	1	$B_{3,1}^{(0)}$	0.43886792994934987
$\beta_1^{(1)}$	0	$B_{3,2}^{(0)}$	0.7490415909204886
$\beta_2^{(1)}$	0.92438032145683		

SOSRI

Coefficient	Value	Coefficient	Value
$A_{2,1}^{(0)}$	-0.04199224421316468	α_3	0.4736296532772559
$A_{3,1}^{(0)}$	2.842612915017106	α_4	0.026404498125060714
$A_{3,2}^{(0)}$	-2.0527723684000727	$c_2^{(0)}$	-0.04199224421316468
$A_{4,1}^{(0)}$	4.338237071435815	$c_3^{(0)}$	0.7898405466170333
$A_{4,2}^{(0)}$	-2.8895936137439793	$c_4^{(0)}$	3.7504010171562823
$A_{4,3}^{(0)}$	2.3017575594644466	$c_1^{(1)}$	0
$A_{2,1}^{(1)}$	0.26204282091330466	$c_2^{(1)}$	0.26204282091330466
$A_{3,1}^{(1)}$	0.20903646383505375	$c_3^{(1)}$	0.05879875232001766
$A_{3,2}^{(1)}$	-0.1502377115150361	$c_4^{(1)}$	0.758661169101175
$A_{4,1}^{(1)}$	0.05836595312746999	$\beta_1^{(1)}$	-1.8453464565104432
$A_{4,2}^{(1)}$	0.6149440396332373	$\beta_2^{(1)}$	2.688764531100726
$A_{4,3}^{(1)}$	0.08535117634046772	$\beta_3^{(1)}$	-0.2523866501071323
$B_{2,1}^{(0)}$	-0.21641093549612528	$\beta_4^{(1)}$	0.40896857551684956
$B_{3,1}^{(0)}$	1.5336352863679572	$\beta_1^{(2)}$	0.4969658141589478
$B_{3,2}^{(0)}$	0.26066223492647056	$\beta_2^{(2)}$	-0.5771202869753592
$B_{4,1}^{(0)}$	-1.0536037558179159	$\beta_3^{(2)}$	-0.12919702470322217
$B_{4,2}^{(0)}$	1.7015284721089472	$\beta_4^{(2)}$	0.2093514975196336
$B_{4,3}^{(0)}$	-0.20725685784180017	$\beta_1^{(3)}$	2.8453464565104425
$B_{2,1}^{(1)}$	-0.5119011827621657	$\beta_2^{(3)}$	-2.688764531100725
$B_{3,1}^{(1)}$	2.67767339866713	$\beta_3^{(3)}$	0.2523866501071322
$B_{3,2}^{(1)}$	-4.9395031322250995	$\beta_4^{(3)}$	-0.40896857551684945
$B_{4,1}^{(1)}$	0.15580956238299215	$\beta_1^{(4)}$	0.11522663875443433
$B_{4,2}^{(1)}$	3.2361551006624674	$\beta_2^{(4)}$	-0.57877086147738
$B_{4,3}^{(1)}$	-1.4223118283355949	$\beta_3^{(4)}$	0.2857851028163886
α_1	1.140099274172029	$\beta_4^{(4)}$	0.17775911990655704
α_2	-0.6401334255743456		

SOSRI2

Coefficient	Value	Coefficient	Value
$A_{2,1}^{(0)}$	0.13804532298278663	α_3	0.686995463807979
$A_{3,1}^{(0)}$	0.5818361298250374	α_4	-0.2911544680711602
$A_{3,2}^{(0)}$	0.4181638701749618	$c_2^{(0)}$	0.13804532298278663
$A_{4,1}^{(0)}$	0.4670018408674211	$c_3^{(0)}$	1
$A_{4,2}^{(0)}$	0.8046204792187386	$c_4^{(0)}$	1
$A_{4,3}^{(0)}$	-0.27162232008616016	$c_1^{(1)}$	0
$A_{2,1}^{(1)}$	0.45605532163856893	$c_2^{(1)}$	0.45605532163856893
$A_{3,1}^{(1)}$	0.7555807846451692	$c_3^{(1)}$	1
$A_{3,2}^{(1)}$	0.24441921535482677	$c_4^{(1)}$	1
$A_{4,1}^{(1)}$	0.6981181143266059	$\beta_1^{(1)}$	-0.45315689727309133
$A_{4,2}^{(1)}$	0.3453277086024727	$\beta_2^{(1)}$	0.8330937231303951
$A_{4,3}^{(1)}$	-0.04344582292908241	$\beta_3^{(1)}$	0.3792843195533544
$B_{2,1}^{(0)}$	0.08852381537667678	$\beta_4^{(1)}$	0.24077885458934192
$B_{3,1}^{(0)}$	1.0317752458971061	$\beta_1^{(2)}$	-0.4994383733810986
$B_{3,2}^{(0)}$	0.4563552922077882	$\beta_2^{(2)}$	0.9181786186154077
$B_{4,1}^{(0)}$	1.73078280444124	$\beta_3^{(2)}$	-0.25613778661003145
$B_{4,2}^{(0)}$	-0.46089678470929774	$\beta_4^{(2)}$	-0.16260245862427797
$B_{4,3}^{(0)}$	-0.9637509618944188	$\beta_1^{(3)}$	1.4531568972730915
$B_{2,1}^{(1)}$	0.6753186815412179	$\beta_2^{(3)}$	-0.8330937231303933
$B_{3,1}^{(1)}$	-0.07452812525785148	$\beta_3^{(3)}$	-0.3792843195533583
$B_{3,2}^{(1)}$	-0.49783736486149366	$\beta_4^{(3)}$	-0.24077885458934023
$B_{4,1}^{(1)}$	-0.5591906709928903	$\beta_1^{(4)}$	-0.4976090683622265
$B_{4,2}^{(1)}$	0.022696571806569924	$\beta_2^{(4)}$	0.9148155835648892
$B_{4,3}^{(1)}$	-0.8984927888368557	$\beta_3^{(4)}$	-1.4102107084476505
α_1	-0.15036858140642623	$\beta_4^{(4)}$	0.9930041932449877
α_2	0.7545275856696072		

4.9.4 SRK Order Conditions

Order Conditions for Rler-SRI Methods

The coefficients

$\left(A_0, B_0, \beta^{(i)}, \alpha\right)$ 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 $f,g\in C^{1,2}(\mathcal{I}\times\mathbb{R}^d,\mathbb{R}^d)$, $c^{(i)}=A^{(i)}e$, $e=(1,1,1,1)^T$ [95].

Order Conditions for Rler-SRA Methods

The coefficients
$\left(A_0,B_0,\beta^{(i)},\alpha\right)$ must satisfy the conditions for order 1:

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

and the additional conditions for order 1.5:

1. $\alpha^TB^{(0)}e=1$
2. $\alpha^TA^{(0)}e=\frac{1}{2}$
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 $g\in C^1(\mathcal{I},\mathbb{R}^d)$ [95].

4.9.5 Derivation Details

$$\begin{aligned}
\left(I-\mu\Delta tA^{(0)}\right)H^{(0)} &= U_n+\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\left(I-\sigma\sqrt{\Delta t}B^{(1)}\right)^{-1}\left(U_n+\mu\Delta tA^{(1)}H^{(0)}\right),\\
\left(I-\mu\Delta tA^{(0)}\right)H^{(0)}-\left[\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\left(I-\sigma\sqrt{\Delta t}B^{(1)}\right)^{-1}\right]\mu\Delta tA^{(1)}H^{(0)} &= U_n+\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\left(I-\sigma\sqrt{\Delta t}B^{(1)}\right)^{-1}U_n\\
\left(I-\mu\Delta tA^{(0)}-\mu\Delta tA^{(1)}\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\left(I-\sigma\sqrt{\Delta t}B^{(1)}\right)^{-1}\right)H^{(0)} &= \left(I+\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\left(I-\sigma\sqrt{\Delta t}B^{(1)}\right)^{-1}\right)U_n\\
H^{(0)} &= \left(I-\mu\Delta tA^{(0)}-\mu\sigma I_{(1,0)}A^{(1)}B^{(0)}\left(I-\sigma\sqrt{\Delta t}B^{(1)}\right)^{-1}\right)^{-1}\\
&\quad \left(I+\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\left(I-\sigma\sqrt{\Delta t}B^{(1)}\right)^{-1}\right)U_n
\end{aligned}$$

$$\begin{aligned}
\left(I-\sigma\sqrt{\Delta t}B^{(1)}\right)H^{(1)} &= U_n+\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\left(U_n+\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}H^{(1)}\right)\\
\left(I-\sigma\sqrt{\Delta t}B^{(1)}-\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\right)H^{(1)} &= U_n+\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}U_n\\
\left(I-\sigma\sqrt{\Delta t}B^{(1)}-\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\right)H^{(1)} &= \left(I+\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\right)U_n\\
H^{(1)} &= \left(I-\sigma\sqrt{\Delta t}B^{(1)}-\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\right)^{-1}\\
&\quad\left(I+\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\right)U_n
\end{aligned}$$

$$\begin{aligned}
U_{n+1} &= U_n+\mu\Delta t\left(\alpha\cdot\left[\left(I-\mu\Delta tA^{(0)}-\mu\sigma I_{(1,0)}A^{(1)}B^{(0)}\left(I-\sigma\sqrt{\Delta t}B^{(1)}\right)^{-1}\right)^{-1}\left(I+\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\left(I-\sigma\sqrt{\Delta t}B^{(1)}\right)^{-1}\right)\right]U_n\right)\\
&\quad+\sigma I_{(1)}\left(\beta^{(1)}\cdot\left[\left(I-\sigma\sqrt{\Delta t}B^{(1)}-\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\right)^{-1}\left(I+\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\right)\right]U_n\right)\\
&\quad+\sigma\frac{I_{(1,1)}}{\sqrt{\Delta t}}\left(\beta^{(2)}\cdot\left[\left(I-\sigma\sqrt{\Delta t}B^{(1)}-\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\right)^{-1}\left(I+\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\right)\right]U_n\right)\\
&\quad+\sigma\frac{I_{(1,0)}}{\Delta t}\left(\beta^{(3)}\cdot\left[\left(I-\sigma\sqrt{\Delta t}B^{(1)}-\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\right)^{-1}\left(I+\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\right)\right]U_n\right)\\
&\quad+\sigma\frac{I_{(1,1,1)}}{\Delta t}\left(\beta^{(4)}\cdot\left[\left(I-\sigma\sqrt{\Delta t}B^{(1)}-\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\sigma\frac{I_{(1,0)}}{\Delta t}B^{(0)}\right)^{-1}\left(I+\mu\Delta tA^{(1)}\left(I-\mu\Delta tA^{(0)}\right)^{-1}\right)\right]U_n\right)
\end{aligned}$$

Thus we substitute in the Wiktorsson approximations

$$\begin{aligned}
I_{(i,i)} &= \frac{1}{2}\left(\Delta W^2-h\right)\\
I_{(i,i,i)} &= \frac{1}{6}\left(\Delta W^3-3h\Delta W\right)\\
I_{(i.0)} &= \frac{1}{2}h\left(\Delta W+\frac{1}{\sqrt{3}}\Delta Z\right)
\end{aligned}$$

where $\Delta Z\sim N(0,h)$ is independent of $\Delta W\sim N(0,h)$. By the properties of the normal distribution, we have that

$$E\left[(\Delta W)^n\right]=0$$

for any odd n and

$$\begin{aligned}
E\left[(\Delta W)^2\right] &= h \\
E\left[(\Delta W)^4\right] &= 3h^2 \\
E\left[(\Delta W)^6\right] &= 15h^3 \\
E\left[(\Delta W)^8\right] &= 105h^4,
\end{aligned}$$

and similarly for ΔZ.

Chapter 5

DifferentialEquations.jl - A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia

This chapter was published as [92]. It details the software architecture for DifferentialEquations.jl and highlights the algorithms of 3 as a key novelty of the software. DifferentialEquations.jl's performance advantages over previous software are detailed here and were required to make the spatial results of 2 computable.

5.1 Summary

DifferentialEquations.jl is a package for solving differential equations in Julia. It covers ordinary differential equations, stochastic differential equations, and (stochastic) partial dif-

ferential equations. Through extensive use of multiple dispatch, metaprogramming, plot recipes, foreign function interfaces (FFI), and call-overloading, DifferentialEquations.jl offers a unified user interface to solve and analyze various forms of differential equations while not sacrificing features or performance. Many modern features are integrated into the solvers, such as allowing arbitrary user-defined number systems for high-precision and unit-checked arithmetic, built-in multithreading and parallelism, and symbolic calculation of Jacobians. Integrated into the package is an algorithm testing and benchmarking suite to both ensure accuracy and serve as an easy way for researchers to develop and distribute their own methods. Together, these features build a highly extendable suite which is feature-rich and highly performant.

5.2 Introduction

Differential equations are fundamental components of many scientific models; they are used to describe large-scale physical phenomena like planetary systems [45] and the Earth's climate [51, 74], all the way to smaller scale biological phenomena like biochemical reactions [130] and developmental processes [112, 36]. Because of the ubiquity of these equations, standard sets of solvers have been developed, including Shampine's ODE suite for MATLAB [106], Hairer's Fortran codes [41], and the Sundials CVODE solvers [46].

However, these software packages contain many limitations which stem from their implementation and the time when they were developed. Since the time of their inception, many other forms of differential equations have become commonplace tools not only for mathematicians, but throughout the sciences. Other forms of differential equations, such as stochastic differential equations (SDEs), have become more commonplace not only in mathematical finance [99, 25], but also in biochemical [19, 53] and ecological models. Delay differential equations have become a ubiquitous tool for modeling phenomena with natural delays as seen in Neuroscience [17, 98] and control theory [104]. However, a user who is familiar with standard ODE tools has to “leave the box” to find a new specialized package to handle these kinds of differential equations, or write their own solver scripts [44]. Also, when many of these methods were implemented the standard computer was limited by the speed of the processor. These days, most processors are multi-core and many computers contain GPGPU [4] or Xeon Phi [72, 30] acceleration cards and thus taking advantage of the ever-present parallelism is key to achieving good performance.

Other design limitations stem from the programming languages used in the implementation. Many of these algorithms, being developed in early C/Fortran, do not have abstractions for generalized array formats. In order to use these algorithms, one must provide the solver with a vector. In cases where a matrix or a higher dimensional tensor are the natural representation of the differential equation, the user is required to transform their equation into a vector equation for use in these solvers. Also, these solvers are limited to using 64-bit floating point calculations. The numerical precision limits their use in high-precision applications, requiring specialized codes when precision lower than 10^{-16} is required. Lastly, many times these programs are interfaced via a scripting language where looping is not optimized and where “vectorized” codes provide the most efficient solution. However, vectorized coding in the style of MATLAB or NumPy results in temporary allocations and can lack compiler optimizations which require type inference. This increases the computational burden of the user-defined functions which degrades the efficiency of the solver.

The goal of DifferentialEquations.jl is build off of the foundation created by these previous differential equation libraries and modernize them using Julia. Julia is a relatively new programming language which is able to achieve C/Fortran-like speeds with the concise syntax of a scripting language. The language achieves this goal by extensive utilization of multiple dispatch and metaprogramming to design a language that is both easy for a compiler to understand and easy for a programmer to use [5]. DifferentialEquations.jl builds off of these design principles to arrive at a fast, feature-rich, and highly extendable differential equations suite which is easy to use. We start by describing the innovations in usability. In Section 5.3 we show how multiple dispatch is used to consolidate the functions the user needs to into simple descriptive commands like solve and plot. Since these commands are used for all forms of differential equations, the user interface is unified in a manner that makes it easy for a user to explore other types of models. In Section 5.4 we describe the over 100 algorithms currently available to users of DifferentialEquations.jl to solve ODEs, SDEs, and (S)PDEs. Lastly, in Section 5.5 we show how metaprogramming is used to further simplify the user API, allowing the user to define a function in a "mathematical format" which is automatically converted into the computationally-efficient encoding. After that, we describe how the internals were designed in order to be both feature-filled and highly performant. In Section 5.6 we describe how multiple dispatch is used to write a single generic method which compiles into specialized functions dependent on the number types given to the solver. We show how this allows for the solvers to both achieve high performance while being compatible with any Julia-defined number system which implements a few basic mathematical operations, including fast high and intermediate precision numbers and unit-checked arithmetic. In Section 5.7 we describe the experimental within-method multi-threading which is being used to further enhance the performance of the methods, and the multi-node parallelism which is included for performing Monte Carlo simulations of stochastic models. We then discuss some of the tools which allows DifferentialEquations.jl to be a good test suite for the fast development and deployment of new solver algorithms, and the tools provided for performing benchmarks. Lastly, we describe the current limitations and future development plans.

5.3 A Unified API Through Multiple Dispatch

DifferentialEquations.jl uses multiple dispatch on specialized types to arrive at a unified user-API for ODEs, SDEs, and PDEs. To use the package, one follows the steps:

1. Define a problem.
2. Solve the problem.
3. Plot the solution.

This standardization of the API makes complicated solvers accessible to less programming-inclined individuals, giving a good framework for future development and allows for the latest research in numerical differential equations to be utilized without complications.

5.3.1 Solving ODEs

To define a problem, a user must call the constructor for the appropriate problem object. Since ordinary differential equations (ODEs) are represented in the general form as

$$\frac{du}{dt} = f(t,u),\ u(0) = u_0, \tag{5.1}$$

the `ODEProblem` is defined by specifying a function f and an initial condition u_0. For example, we can define the linear ODE using the commands:

```
using DifferentialEquations
f(t,y) = 0.5y
u0 = 1.5
prob = ODEProblem(f,u0)
```

Many other examples are provided in the documentation and the Jupyter notebook tutorials. To solve the ODE, the user can simply call the solve command on the problem over a specified timespan:

```
timespan = [0,1] # Solve from time = 0 to time = 1
sol = solve(prob,timespan) # Solves the ODE
```

By using a dispatch architecture on `AbstractArrays` and using the array-defined indexing functionality provided by Julia (such as `eachindex(A)`), DifferentialEquations.jl accepts problems defined on arrays of any size. For example, one can define and solve a system of equations where the dependent variable u is a matrix as follows:

```
A = [1. 0 0 -5
     4 -2 4 -3
     -4 0 0 1
     5 -2 2 3]
u0 = rand(4,2)
f(t,u) = A*u
prob = ODEProblem(f,u0)
sol = solve(prob,timespan)
```

For most other packages, one would normally have to define u as a vector and rewrite the system of equations in the vector form. However, by allowing arbitrary problem sizes, DifferentialEquations.jl allows the user to specify problems in the natural format and solve directly on any array of numbers. This can be helpful for problems like discretizations of partial differential equations (PDEs) where the matrix format matches some underlying structure, and could result in a denser formulation.

The solver returns a solution object which holds all of the information about the solution. Dispatches to array functions are provided on the `sol` object, allowing for the solution object act like a timeseries array. In addition, high-order efficient interpolations are lazily constructed throughout the solution (by default, a feature which can be turned off) and the `sol` object's call is overloaded with the interpolating function. Thus the solution object can both be used as an array of the solution values, and as a continuous approximation given by the numerical solution. The syntax is as follows:

```
sol[i] # ith solution value
sol.t[i] # ith timepoint
sol(t) # Interpolated solution at time t
```

The solution can be plotted using the provided plot recipes through Plots.jl. The plot recipes use the solver object to build a default plot which is customizable using any of the commands from the Plots.jl package, and can be plotted to any plotting backend provided by Plots.jl. For example, we can by default plot to the PyPlot backend (a Julia wrapper for matplotlib) via the command:

```
plot(sol)
```

These defaults are deliberately made so that a standard user does not need to dig further into the manual and understand the differences between all of the algorithms. However, an extensive set of functionality is available if the user wishes. All of these functions can be modified via keyword arguments. For example, to change the solver algorithm to a highly efficient Order 7 method due to Verner [119], set the line width in the plot to 3 pixels, and add enlarge the tick labels in the plot, one could instead use the commands:

```
sol = solve(prob,alg=:Vern7) # Unrolled Verner 7
plot(sol,xlabel="t",ylabel="u(t)")
```

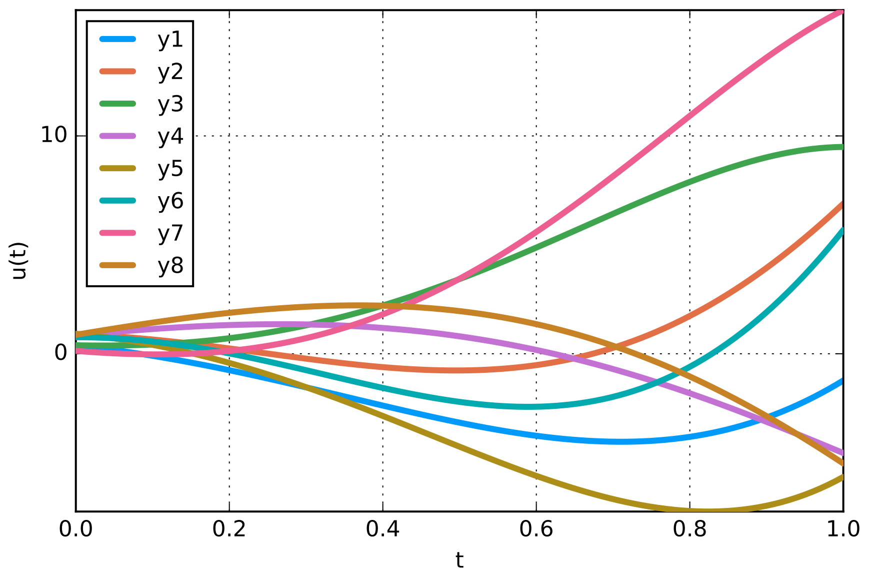


Figure 5.1: **Example of the ODE plot recipe.** This plot was created using the PyPlot backend through Plots.jl. Shown is the solution to the 4x2 ODE with $f(t, u) = Au$ where A is given in the code. Each line corresponds to one component of the matrix over time.

The output of this command is shown in Figure 5.1. Note that the output is smoothed using 100 equally spaced interpolated values through the timespan.

Lastly, these solvers tie into Julia integrated development environments (IDEs) to further enhance the ease of use. Users of the Juno IDE [56] are equipped with a progressbar and time estimates to monitor the progress of the solver. Additionally, all of the DifferentialEquations.jl functions are thoroughly tested and documented with the Jupyter notebook system [86], allowing for reproducible exploration.

5.3.2 Solving SDEs

By using multiple-dispatch, the same user API is offered for other types of equations. For example, if one wishes to solve a stochastic differential equation (SDE):

$$dX_t = f(t, X_t)dt + g(t, X_t)dW_t, \tag{5.2}$$

then one builds an SDEProblem object by specifying the initial condition and now the two functions, f and g. However, the rest of the usage is the same: simply use the solve and plot functions. To extend the previous example to have multiplicative noise, the code would be:

```
g(t,u) = 0.3u
prob = SDEProblem(f,g,u0)
sol = solve(prob,timespan)
plot(sol)
```

While this user interface is simple, the methods these algorithms can call are efficient high-order solvers. These methods tie into the plotting functionality and IDEs in the same manner as the ODE solvers, making it easy for users to explore stochastic modeling without having to learn learn a new interface.

5.3.3 Solving (Stochastic) PDEs

Again, the same user API is offered for the available stochastic PDE solvers. Instead, one builds a `HeatProblem` object which dispatches to algorithms for solving (Stochastic) PDEs. An example using the previously defined functions is:

```
T = 5
dx = 1/2^(1)
dt = 1/2^(7)
fem_mesh = parabolic_squaremesh([0 1 0 1],dx,dt,T,:neumann)
prob = HeatProblem(f,σ=g)
sol = solve(fem_mesh::FEMmesh,prob::HeatProblem)
```

Additional keyword arguments can be supplied to `HeatProblem` to specify boundary data and initial condtions. Notice that the main difference is now we must specify a space-time mesh (and boundary conditions as optional keyword arguments). Again, the same plotting and analysis commands apply to the solution object `sol` (where now the `plot` dispatch is to a `trisurf` plot).

5.4 The Current Library of Algorithms

5.4.1 Deterministic ODE Methods

For ODE integrators, DifferentialEquations.jl offers over 80 different Runge-Kutta algorithms in a tableau form, (what the authors could find as) the largest collection of Runge-Kutta algorithms available in a ready-to-use format. These are implemented in a generic devectorized format which minimizes allocations and allows for optimizations like FSAL and Lund-stabilized adaptive timestepping to be used [41]. In addition, the top performing and most well-known algorithms, such as the Dormand-Prince 4/5 pair, the Bogacki-Shampine 2/3 [7] and 4/5 pairs [103], Verner Efficient pairs [119], Dormand-Prince 8/5/3 [41], and the Feagin Order 10, 12, and 14 methods [32, 31], have an extra optimized implementation where the constants have been declared separately to ensure that the pieces are stack allocated, leading to even better performance. DifferentialEquations.jl also includes Rosenbrock and a few implicit methods (Implicit Euler, Trapezoid) for stiff equations [106].

The ODE suite is rounded out by providing wrappers to other well known libraries. The classic "Hairer" Fortran methods [41], such as dopri5, dop853, odex, and radau, are included via a wrapper to ODEInterface.jl. The installation of these methods only requires two lines of Julia and is thus accessible to those who are not familiar with Fortran. The Sundials ODE solvers [46] are also accessible in DifferentialEquations.jl, allowing one to access CVODE's BDF and Adams-Moulton methods. These implementations are widely regarded as some of the most efficient for stiff equations. Lastly, a wrapper to the other popular Julia package, ODE.jl [82], is included for completeness. After a quick Julia-only installation, all of these algorithms are accessed through the alg keyword, meaning that no code has to be changed in order to use these other libraries. This seamless integration allows for a user to switch between solver libraries and access the strengths of each: the DifferentialEquations,jl native solvers allow for extra features like arbitrary Julia-defined numeric types and have well-developed and highly performant non-stiff solvers, while the radau and CVODE BDF solvers are tried-and-tested solvers for highly stiff equations.

5.4.2 Stochastic ODE Methods

The SDE solver library is one of the most extensive to date. The standard Euler-Maruyama method, a Runge-Kutta version of the Milstein method [63], the Strong Order 1.5 Rossler SRI, and the Strong Order 2.0 Rossler SRA algorithms are provided for efficient solving of SDEs [95]. While most other libraries provide the Kloden-Platen-Schurz Strong Order 1.5 Taylor method [50], the Rossler methods are a Runge-Kutta method which take a reduced number of function evaluations and requires no derivative estimations, leading to a more efficient algorithm. As with the ODE methods, specially optimized versions of the most used tableaus, such as the SRIW1 and SRA1 methods, are provided for maximum efficiency. Also included is an efficient algorithm for adaptive timestepping [91], a first among readily available stochastic differential equation solver libraries. Together these algorithms form a state-of-the-art library for stochastic differential equations.

5.4.3 (Stochastic) Poisson and Heat Methods

While the ODE and SDE libraries can be used to solve (S)PDEs themselves, DifferentialEquations.jl provides PDE solvers for common problems. Currently Finite Element Method (FEM) solvers exist for the (stochastic) Poisson and Heat equations, and a Finite Difference Method (FDM) solver exists for a Stationary Stokes Equation. The Poisson and Heat equation solvers include the semilinear variants, given by the equations

$$u_t = \Delta u + f(u) \tag{5.3}$$

and

$$-\Delta u = f(u) \tag{5.4}$$

which are also known as the Reaction-Diffusion Equation and the stationary Reaction-Diffusion Equation respectively. Standard methods such as the Semi-Implicit Crank-Nicholson [24] and Implicit Euler methods can be used to solve the equations. Many solver options can be given to tailored the method to the specific problem. For example, one can specify the solving of the linear systems via a decomposition like LU/QR/SVD, which incurs an upfront cost to cache but allows for faster solving if the decomposed stiffness matrix can fit in memory, or iterative methods such as conjugate-gradient or GMRES. The implicit equations can be solved using forward-mode auto-differentiation provided via NLSolve.jl and ForwardDiff.jl. The finite element tools also include general FEM tools, such as those for developing and analyzing meshes. In addition to the plot recipes, the HeatSolution object also includes the ability to animate the solution via the command `animate(sol)`.

5.5 Enhanced Performance and Readability Through Macros

5.5.1 A Macro-Based Interface

Most differential equations packages require that the user understands some details about the implementation of the library. For example, while the famous Lorenz system is mathematically defined as

$$\frac{dx}{dt} = \sigma\left(y - x\right) \tag{5.5}$$
$$\frac{dy}{dt} = x\left(\rho - z\right) - y \tag{5.6}$$
$$\frac{dz}{dt} = xy - \beta z \tag{5.7}$$

a user must re-write this function in a “computer friendly format”, defining `u=[x;y;z]` as a vector and writing the equation in terms of this vector. The format for ODE.jl, which is similar to other scripting languages like SciPy or MATLAB, is as follows:

```
f = (t,u,du) -> begin
du[1] = 10*(u[2]-u[1])
du[2] = u[1]*(28-u[3]) - u[2]
du[3] = u[1]*u[2] - 8/3*u[3]
end
```

While this format is accepted by DifferentialEquations.jl, additional usability macros are provided which will automatically translate user input from a more mathematical format. For ODEs, `@ode_def` is provided which allows the user to define the same ODE as follows:

```
f = @ode_def Lorenz begin
dx = σ*(y-x)
dy = x*(ρ-z) - y
dz = x*y - β*z
end σ=>10. ρ=>28. β=(8/3)
```

Since Julia allows for the use of Unicode within code, this format matches the style one would expect to see in a TeX'd publication. The macro takes in this definition, finds the values for the left-hand side of the form "d__", and uses a dictionary in order to find/replace these values to write a function which is in the format of the other scripting language libraries. Thus the translation to a vector system can be done by DifferentialEquations.jl, allowing the users to have more readable scripts while not sacrificing performance. In addition, the macro produces a function which updates an input `du` in-place as the output. This detail can be hard for non-programmers to understand but is required for achieving fast solutions since otherwise every function call requires an array allocation.

5.5.2 Explicit Parameters

A unique feature from this form of function definition is that the parameters are built into the function type itself. The actual implementation involves creating a type `Lorenz` with fields for the parameters (inlining parameters defined with `=` instead of `=>` during compilation). Then the type is set to have its call overloaded by the standard `f(t,u,du)` function signature, effectively acting like the appropriate function. However, the parameters are still accessible via the type fields, for example `f.a` or the overloaded `f[:a]`. This allows for sensitivity analysis, bifurcation diagrams, and parameter estimations to be computed using the same function, allowing for this infrastructure to extend far beyond the domain of differential equations solvers.

5.5.3 Enhanced Performance Through Symbolic Calculations

Also, since the code is analyzed by the program at the expression level, silent optimizations are able to be performed. For example, during the construction of the function, the code is transformed into a symbolic form for use in the high-performance CAS SymEngine [113], where the Jacobian is calculated and an in-place function for its computation is created. In addition, the symbolic expression is inverted, allowing for stiff solvers which require inverting Jacobians to be written as directly computed matrices and matrix multiplications.

5.5.4 Finite Element Extensions

An additional macro is provided for use in the Finite Element Methods (FEM) part of the library. The internal library is written in a dimension-invariant form, where space is determined by a variable x with coordinates $(x_1, x_2, \ldots, x_n)$. These are encoded as `x=x[:,1]`, `y=x[:,2]`. Also, systems of equations internally held as a matrix where each column `u[:,i]` denotes the values of u_i at each element node. While suitable for the computations, this format does not allow for readability of function definitions. The `@fem_def` macro allows a user to specify a PDE in a mathematical format using space variables x, y, z and a tuple of variable names, and produces an output which is suitable for DifferentialEquations.jl.

For example, take the Gierer-Meinhardt equations [36] with a linear forcing term:

$$u_t = \Delta u + \frac{au}{v^2} + \zeta - \alpha u + \cos(xy)t^2 \tag{5.8}$$

$$v_t = \Delta v + au^2 + \gamma - \beta v \tag{5.9}$$

Using the usability macro `@fem_def`, the user can define the data for an FEMProblem by

```
f = @fem_def (t,x,u) GiererMeinhardt begin
du = (a*u/v^2)+ζ-α*u+cos(x*y)*t^2
dv = a*u^2+γ-β*v
end a=1 α=1 ζ=2 γ=3 β=10
```

which will translate into

```
f = (t,x,u) -> begin
[(u[:,1]/u[:,2]^2)+2-u[:,1]+cos(x[:,1]*x[:,2])*t^2 u[:,1]^2+3-10*u[:,2]]
end
```

To really appreciate the the convenience of the macro, notice that the computer friendly output is actually a two column vector with two expression separated by a space. Using characters to concatenate the arrays, such as `;`, would cause them to incorrectly concatenate as one large vector. However, this causes a delicate situation because then this syntax is spacing sensitive, meaning that the location of spaces determines the output. For an extreme example, notice that for:

```
[cos(2*pi.*x[:,1]).*cos(2*pi.*x[:,2])./(4*pi) -sin(2.*x[:,1]).*sin(2.*x[:,2])./(4)]
[cos(2*pi.*x[:,1]).*cos(2*pi.*x[:,2])./(4*pi) - sin(2.*x[:,1]).*sin(2.*x[:,2])./(4)]
```

the first definition is two expressions and thus creates a two column array, whereas the second incorrectly would parse as a one column vector. However, the user is effectively shielded from such issues when using the `@fem_def` macro, making FEM solver more accessible.

5.6 Multiple Dispatch as a tool for Arbitrary Numerics

Julia's base library defines its standard numeric types, `Float64`, `Int64`, etc., as concrete subtypes of the abstract type `Number`. The implementation is contained within Julia: using the concrete `BitsType` as a way to store numbers, and defining the operations such as +,-, etc. for each pair of numbers using dispatch on subtypes of Number. The result is that each number type receives its own compiled function for each operation, resulting in performance which is 1x with C (as can be investigated via the `@code_llvm` and `@code_native` macros). This design allows for users to develop packages which are new number systems.

DifferentialEquations.jl utilizes Julia's multiple dispatch architecture to allow for fast performance over these arbitrary numerical types. The design of the integration schemes includes a wrapper over the integration loops which matches types (to ensure type-stability), choosing the types for the problem by the user defined u_0 (the initial condition) and Δt (the initial timestep)(this can be overruled). It then calls a type-dependent integration function which is optimized via JIT compilation for the numeric types given to the function. Different dispatches are given for subtypes of Number and AbstractArray since arrays are mutable and heap allocated, meaning that when numbers are treated natively instead of as arrays of size 1 a large speedup can occur. This allows for the internal integration algorithms to achieve C/Fortran speeds, while allowing for the generic numerical types and the readability of being in Julia itself.

For a current list of number types which are compatible with DifferentialEquations.jl, check out the "Solving Equations with Julia-Defined Types" notebook. The following subsections highlight two important examples.

5.6.1 Case 1: Arbitrary Precision Numerics

One advantage of this design beyond speed is that it allows the user of DifferentialEquations.jl to use any type which is a subclass of Number as the number type for the equations. This includes not only the basic types like Float64 and Int64, but also Rational and arbitrary precision BigFloats (based off of GNU MPFR). However, even numeric types defined in packages (which implement +,-,/, and for optionally for adaptive timestepping, sqrt) can be used within DifferentialEquations.jl. Some examples which have been shown to work are ArbFloats (a library for faster high-precision numbers than MPFR floats between 64 and 512 bits based on the Arb library of Fredrik Johansson [54]) and DecFP (an implementation of IEEE 754-2008 Decimal Floating-Point Arithmetic).

The combination of high-performance number systems with high order Runge-Kutta methods such as the Order 14 methods due to Feagin allows for fast solving with high accuracy. For an example showing this combination, see the "Feagin's Order 10, 12, and 14 methods" notebook in the examples folder[1].

5.6.2 Case 2: Unitful Numbers

This design also allows DifferentialEquations.jl to be compatible with unit-checked arithmetic. SIUnits.jl and Unitful.jl are packages which have developed number implementations which have units. Numbers defined by these packages automatically constrain the equations to satisfy dimensional constraints. For example, if one tries to add a quantity with units of seconds with a quantity with units of Newtons, it will throw an error. This is useful in fields like physics where these dimensional analysis tools are used to check for correctness in equations. DifferentialEquations.jl was developed such that the internal solvers satisfy dimensional constraints. Thus one can use unitful numbers like other arbitrary number systems. The "Unit Checked Arithmetic via SIUnits" notebook in the examples folder[2] describes the usage of this feature. For example, we can solve an ODE where the dependent variable is in terms of seconds and the independent variable is in terms of Newtons via the following equation:

[1]https://github.com/JuliaDiffEq/DifferentialEquations.jl/blob/master/examples/Feagin's%20Order%2010%2C%2012%2C%20and%2014%20methods.ipynb

[2]https://github.com/JuliaDiffEq/DifferentialEquations.jl/blob/master/examples/Unit%20Checked%20Arithmetic%20via%20SIUnits.ipynb

```
using DifferentialEquations, SIUnits, SIUnits.ShortUnits
f = (t,y) -> 0.5*y
u = 1.5N
prob = ODEProblem(f,u)
sol =solve(prob,[0,1],Δt=(1/2^4)s)
```

The attentive reader should realize that this will correctly throw an error: the output of the function in an ODE must be a rate, and therefore must have units of N/s in this example. DifferentialEquations.jl will thus return an error notifying the user that the dimensions are off by a unit of seconds. Instead, the pleased physicists would modify the previous code by a rate constant and use the following code instead:

```
f = (t,y) -> 0.5*y/3.0s
u = 1.5N
prob = ODEProblem(f,u)
sol =solve(prob,[0,1],Δt=(1/2^4)s)
```

This will produce an output whose units are in terms of Newtons, and with time in terms of seconds.

5.7 Integrated Parallelism

5.7.1 Within-Method Multithreading

DifferentialEquations.jl also includes parallelism whenever possible. One area where parallelism is currently being employed is via "within-method" parallelism for Runge-Kutta methods. Using Julia's experimental multithreading, DifferentialEquations.jl provides a multithreaded version of the `DP5` solver. Benchmarks using the tools from Section 5.8 show that this can give a 30% speedup over the non-multithreaded algorithm for problem sizes ranging from 75x75 matrices to 200x200 matrices. For larger problems this trails off as more time is spent within the function evaluations, thus reducing the difference between the methods. See the "Multithreaded Runge-Kutta Methods" notebook[3] in the benchmarks folder for the most up-to-date results as this may change rapidly along with Julia's threading implementation.

5.7.2 Multi-Node Monte Carlo Simulations

Also, DifferentialEquations.jl provides methods for performing parallel Monte Carlo simulations for SDEs. Using Julia's pmap construct, one is able to specify for a problem to be solved N times, and DifferentialEquations.jl will distribute this automatically across multiple nodes of a cluster. A vector of results along with summary statistics is returned for the solution. This functionality has been tested on the local UC Irvine cluster (using SGE) and the XSEDE Comet cluster (using Slurm).

5.8 Development, Testing, and Benchmarking

DifferentialEquations.jl includes a suite specifically designed for researchers interested in developing new methods for differential equations (like the authors themselves). This includes functionality for easy integration of new methods, extensive testing, and a benchmarking suite.

[3]https://github.com/JuliaDiffEq/DifferentialEquations.jl/blob/master/benchmarks/Multithreaded%20Runge-Kutta%20Methods.ipynb

5.8.1 Development

The design of DifferentialEquations.jl allows for users to add new integration methods by adding new dispatches. Let's take as an example the ODE suite. The ODE solver first works by setting up options and fixing types, and then calls the `ode_solve` function with the first type parameter being the algorithm. Thus to define a new algorithm, one defines a new dispatch for `ode_solve` where the first parameter is the symbol for the algorithm name. For writing the internal loop, DifferentialEquations.jl provides convenience macros to inline commonly used functionality. For example, `@ode_preamble` declares all of the major arrays and the components for the adaptive timestepping, while `@ode_loopheader` performs iteration tests and `@ode_numberloopfooter` does the adaptive timestepping (if enabled), saving of the values, and extra features like updating the progressbar. Thus to develop the algorithm one only needs to specify the inner loop. The following is the inner loop for the dispatch of the classic Runge-Kutta order 4 algorithm on Numbers:

```
while t < T
@ode_loopheader
k1 = f(t,u)
ttmp = t+halfdt
k2 = f(ttmp,u+halft*k1)
k3 = f(ttmp,u+halft*k2)
k4 = f(t+t,u+t*k3)
u = u + dt*(k1 + 2(k2 + k3) + k4)/6
@ode_numberloopfooter
end
```

This allows for quick development of new methods with the resulting code being easy to check against the mathematical formulation. Lastly, due to the dispatch nature of the internal solvers, one can add their own algorithms to DifferentialEquations.jl without modifying the source code by adding their own dispatch to the integrator type, and pushing the symbol for their algorithm into the algorithm dictionaries. This makes DifferentialEquations.jl an easily extendable library which allows algorithm developers to take full advantage of the convenience features provided by DifferentialEquations.jl.

5.8.2 Testing

The DifferentialEquations.jl suite includes a large number of testing functions to ensure correctness of all of the algorithms. Many premade problems with analytical solutions are provided and convergence testing functionality is included to be able to test the order of accuracy and plot the results. All of the DifferentialEquations.jl algorithms are tested using the Travis and AppVoyer Continuous Integration (CI) testing services to ensure correctness.

5.8.3 Benchmarking

Lastly, a benchmarking suite is included to test the efficiency of different algorithms. Two forms of benchmarking are included: the `Shootout` and the `WorkPrecision`. A `Shootout` solves using all of the algorithms in a given setup and calculates an average time (over a user-chosen number of runs) and error for each algorithm. The `WorkPrecision` and `WorkPrecisionSet` additionally take in vectors of tolerances and draw work-precision diagrams to compare algorithms. Up to date benchmarks can be found in the repository's benchmarks folder[4]. These notebooks can be opened to be run locally via the commands

```julia
using IJulia
notebook(dir=Pkg.dir("DifferentialEquations")*"/benchmarks")
```

[4]https://github.com/JuliaDiffEq/DifferentialEquations.jl/tree/master/benchmarks

As of this publication, the benchmarks show an order of magnitude speedup on nonstiff problems when achieving the same error over the classic Hairer Runge-Kutta implementations and the ODE.jl implementations.

5.9 Limitations and Future Development Plans

While DifferentialEquations.jl already offers many new features and high performance, the package is still under heavy development and will be for the foreseeable future. Currently, most of the methods for stiff equations are wrapped methods (only the Rosenbrock with-/without local extrapolation, and the order 1/2 BDF methods exist in native forms). While these methods, such as CVODE and radau, are widely regarded standards for stiff ODEs, by not being native Julia functions these algorithms choices do not allow for the extra functionality such as arbitrary precision and unit-checked arithmetic (these features require a pure-Julia implementation). Also, DifferentialEquations.jl is currently limited on the types of PDEs it natively supports, and the mesh generation tools are still in their infancy.

To address these issues and more, planned functionality includes (but is not limited to):

- Native and wrapped solvers for Differential Algebraic Equations (DAEs) and Differential Delay Equations (DDEs)
- Finite Difference Methods for common elliptic, parabolic, and hyperbolic PDEs, including high order methods for SPDEs
- Highly parallel accelerated solvers using GPGPUs and Xeon Phi cards (prototypes have already been developed [89])
- A web interface to allow for easy numerical solving of differential equations for non-programmers

- Parameter sensitivity checking
- High order methods for stiff SDEs

For the greater DifferentialEquations ecosystem, the existence of mutable parameters inside of functioned declared by the function definition macros allows for bifurcation plotting and parameter inference to occur on the same functions/types, a convenience which will be exploited within DifferentialEquations or by other packages in the Julia ecosystem. Check the repository issues for the most up to date roadmap.

5.10 Quality Control

Continuous Integration testing with the latest versions of Julia on Mac, Linux, and Windows are provided via Travis and AppVoyer. These tests check most of the features of DifferentialEquations.jl, including the convergence of each algorithm, the ability to plot, the number types used in the computations, and more. Coveralls and Coverage badges are provided on the repository for test coverage analysis, and currently these show a 91% test coverage. As with other Julia packages, a user can check to see if these functionalities are working on their local machine via the command Pkg.test("DifferentialEquations"). Benchmarks in Jupyter notebooks are provided to test the differences between the integrator implementations.

5.11 Reuse Potential

Ordinary differential equations, stochastic differential equations, and partial differential equations form the bedrock of many scientific fields. Therefore, there is no question as to whether numerical differential equation solvers will be used, rather which ones will be used. Julia is a relatively young language which is seeing rapid adoption in the fields of data science and scientific computing due to the performance and productivity that it offers. Because of this, many scientists using Julia will need these tools either as a means to analyze models themselves, or as intermediate tools in more complex methods. With its ease of extendability, its already vast capabilities, and the rapid pace at which this software is being developed, DifferentialEquations.jl looks to be a viable choice for many Julians looking for a differential equations library.

Bibliography

[1] Numerical Simulation of Stochastic Differential Equations. In R. Toral and P. Colet, editors, *Stochastic Numerical Methods*, pages 191–234. Wiley-VCH Verlag GmbH & Co. KGaA, Weinheim, Germany, June 2014.

[2] A. Abdulle and S. Cirilli. S-rock: Chebyshev methods for stiff stochastic differential equations. *SIAM Journal on Scientific Computing*, 30(2):997–1014, 2008.

[3] D. I. K. Ahmadia and A. J. Optimal stability polynomials for numerical integration of initial value problems. *CAMCOS*, 7(2):247–271, 2012.

[4] K. Ahnert, D. Demidov, and M. Mulansky. *Solving Ordinary Differential Equations on GPUs*, pages 125–157. Springer International Publishing, Cham, 2014.

[5] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. 2015.

[6] E. G. Birgin and J. M. Martinez. Improving ultimate convergence of an augmented lagrangian method. *Optimization Methods and Software*, 23(2):177–195, 2008.

[7] P. Bogacki and L. Shampine. A 3(2) pair of runge - kutta formulas. *Applied Mathematics Letters*, 2(4):321–325, 1989.

[8] R. L. Burden and J. D. Faires. *Numerical analysis.* Brooks/Cole, Cengage Learning, Boston, MA, 9th edition, 2011.

[9] 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(2-4):161–177, 1998.

[10] K. Burrage and P. M. Burrage. High strong order explicit runge-kutta methods for stochastic ordinary differential equations. *Applied Numerical Mathematics*, 20:1–21, 1996.

[11] K. Burrage and J. C. Butcher. Non-linear stability of a general class of differential equation methods. *BIT Numerical Mathematics*, 20(2):185–203, 1980.

[12] P. M. Burrage. *Runge-Kutta methods for stochastic differential equations.* Thesis, 1999.

[13] P. M. Burrage and K. Burrage. A variable stepsize implementation for stochastic differential equations. *SIAM Journal on Scientific Computing*, 24(3):848–864, 2002.

[14] J. C. Butcher. A history of runge-kutta methods. *Applied Numerical Mathematics*, 20(3):247–260, 1996.

[15] J. C. Butcher. Numerical methods for ordinary differential equations in the 20th century. *Journal of Computational and Applied Mathematics*, 125(1-2):1–29, 2000.

[16] A. Q. Cai, K. Radtke, A. Linville, A. D. Lander, Q. Nie, and T. F. Schilling. Cellular retinoic acid-binding proteins are essential for hindbrain patterning and signal robustness in zebrafish. *Development (Cambridge, England)*, 139:2150–5, June 2012.

[17] S. A. Campbell. *Time Delays in Neural Systems*, pages 65–90. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[18] L. Cardelli, A. Csiksz-Nagy, N. Dalchau, M. Tribastone, and M. Tschaikowski. Noise Reduction in Complex Biological Switches. 6:20214, 2016.

[19] M. Carletti. Numerical solution of stochastic differential problems in the biosciences. *Journal of Computational and Applied Mathematics*, 185(2):422–440, 2006.

[20] 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(3):201–222, 1990.

[21] F. Ceschino. Modification de la longueur du pas dans l'integration numerique parles methodes a pas lies. *Chiffres*, 2:101–106, 1961.

[22] G. Chalancon, C. N. Ravarani, S. Balaji, A. Martinez-Arias, L. Aravind, R. Jothi, and M. M. Babu. Interplay between gene expression noise and regulatory network architecture. *Trends Genet*, 28:221–232, 2012.

[23] A. Conn, N. Gould, and P. Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572, 1991.

[24] J. Crank and P. Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Advances in Computational Mathematics*, 6(1):207–226, 1996.

[25] R. DeSantiago, J.-P. Fouque, and K. Slna. *Bond Markets with Stochastic Volatility.* 2007.

[26] J. R. Dormand and P. J. Prince. A family of embedded runge kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.

[27] J. R. Dormand and P. J. Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.

[28] I. Dunning, J. Huchette, and M. Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[29] M. B. Elowitz, A. J. Levine, E. D. Siggia, and P. S. Swain. Stochastic Gene Expression in a Single Cell. *Science*, 297:1183, 2002.

[30] J. Fang, A. L. Varbanescu, H. J. Sips, L. Zhang, Y. Che, and C. Xu. An empirical study of intel xeon phi. *CoRR*, abs/1310.5842, 2013.

[31] T. Feagin. High-order explicit runge-kutta methods using m-symmetry.

[32] T. Feagin. A tenth-order runge-kutta method with error estimate.

[33] S. Filippi, C. P. Barnes, P. D. W. Kirk, T. Kudo, K. Kunida, S. S. McMahon, T. Tsuchiya, T. Wada, S. Kuroda, and M. P. H. Stumpf. Robustness of MEK-ERK Dynamics and Origins of Cell-to-Cell Variability in MAPK Signaling. *Cell Reports*, 15:2524–2535, 2016.

[34] 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(5):1455–1484, 1997.

[35] A. Ghasemi and S. Zahediasl. Normality tests for statistical analysis: a guide for non-statisticians. *Int J Endocrinol Metab*, 10(2):486–9, 2012.

[36] A. Gierer and H. Meinhardt. A theory of biological pattern formation. *Kybernetik*, 12(1):30–39, 1972.

[37] H. Gilsing and T. Shardlow. Sdelab: A package for solving stochastic differential equations in matlab. *Journal of Computational and Applied Mathematics*, 205(2):1002–1018, 2007.

[38] T. Gregor, D. W. Tank, E. F. Wieschaus, and W. Bialek. Probing the limits to positional information. *Cell*, 130:153–64, July 2007.

[39] A. Gupta, B. Hepp, and M. Khammash. Noise Induces the Population-Level Entrainment of Incoherent, Uncoupled Intracellular Oscillators. *Cell Systems*, 3:521–531.e13, 2016.

[40] E. Hairer, S. P. Nrsett, and G. Wanner. *Solving ordinary differential equations I*. Springer series in computational mathematics. Springer-Verlag, Berlin ; New York, 2nd rev. edition, 1993.

[41] E. Hairer, S. P. Nrsett, and G. Wanner. *Solving ordinary differential equations I : nonstiff problems*. Springer series in computational mathematics,. Springer, Heidelberg ; London, 2nd rev. edition, 2009.

[42] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems*. Springer, 1991.

[43] E. Hairer and G. Wanner. Stiff differential equations solved by radau methods. *Journal of Computational and Applied Mathematics*, 111(1):93–111, 1999.

[44] D. J. Higham. An algorithmic introduction to numerical simulation of stochastic differential equations.

[45] G. W. Hill. On the extension of delaunay's method in the lunar theory to the general problem of planetary motion. *Transactions of the American Mathematical Society*, 1(2):205–242, 1900.

[46] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.*, 31(3):363–396, 2005.

[47] D. M. Holloway, F. J. Lopes, L. da Fontoura Costa, B. A. Travencolo, N. Golyandina, K. Usevich, and A. V. Spirov. Gene expression noise in spatial patterning: hunchback promoter structure affects noise amplitude and distribution in Drosophila segmentation. *PLoS Computational Biology*, 7:e1001069, Feb. 2011.

[48] 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(11):e1004569, 2015.

[49] M. E. Hosea and L. F. Shampine. Analysis and implementation of tr-bdf2. *Applied Numerical Mathematics*, 20(1):21–37, 1996.

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

[51] W. James, W. Esther, H. Jonathan, and M. Richard. Periodic orbits for a discontinuous vector field arising from a conceptual model of glacial cycles. *Nonlinearity*, 29(6):1843, 2016.

[52] A. Janicki, A. Izydorczyk, and P. Gradalski. *Computer Simulation of Stochastic Models with SDE-Solver Software Package*, pages 361–370. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[53] S. K. Jha and C. J. Langmead. Exploring behaviors of stochastic differential equation models of biological systems using change of measures. *BMC Bioinformatics*, 13(Suppl 5):S8–S8, 2012.

[54] F. Johansson. Efficient implementation of elementary functions in the medium-precision range. *CoRR*, abs/1410.7176, 2014.

[55] S. G. Johnson. The nlopt nonlinear-optimization package.

[56] Juno. 9/21/2016. *v0.2.1*, https://github.com/JunoLab/uber-juno.

[57] M. Kaern, T. C. Elston, W. J. Blake, and J. J. Collins. Stochasticity in gene expression: from theories to phenotypes. *Nat Rev Genet*, 6:451–464, 2005.

[58] J. Kaneko. Explicit order 1.5 runge kutta scheme for solutions of ito stochastic differential equations.

[59] C. A. Kennedy and M. H. Carpenter. Additive runge-kutta schemes for convection-diffusion-reaction equations. *Applied Numerical Mathematics*, 44(1):139–181, 2003.

[60] T. B. Kepler and T. C. Elston. Stochasticity in transcriptional regulation: origins, consequences, and mathematical representations. *Biophysical journal*, 81:3116–36, Dec. 2001.

[61] C. B. Kimmel, R. M. Warga, and D. A. Kane. Cell cycles and clonal strings during formation of the zebrafish central nervous system. *Development*, 120:265–276, 1994.

[62] P. Kloeden and A. Neuenkirch. *Convergence of numerical methods for stochastic differential equations in mathematical finance.* 2012.

[63] P. E. Kloeden and E. Platen. *Numerical Solution of Stochastic Differential Equations.* Springer Berlin Heidelberg, 2011.

[64] Y. Komori and K. Burrage. Weak second order s-rock methods for stratonovich stochastic differential equations. *Journal of Computational and Applied Mathematics*, 236(11):2895–2908, 2012.

[65] Y. Komori and K. Burrage. Strong first order s-rock methods for stochastic differential equations. *Journal of Computational and Applied Mathematics*, 242(Supplement C):261–274, 2013.

[66] A. Koseska, A. Zaikin, J. Kurths, and J. Garca-Ojalvo. Timing Cellular Decision Making Under Noise via CellCell Communication. *PLoS ONE*, 4:e4872, 2009.

[67] H. Lamba. An adaptive timestepping algorithm for stochastic differential equations. *Journal of Computational and Applied Mathematics*, 161(2):417–430, 2003.

[68] F. S. Lawrence. Some practical runge-kutta formulas. *Math. Comput.*, 46(173):135–150, 1986.

[69] J. Lawson. An order five runge-kutta process with extended region of stability. *SIAM Journal on Numerical Analysis*, 3(4):593–597, 1966.

[70] T. Li, A. Abdulle, and W. E. Effectiveness of implicit methods for stiff stochastic differential equations. *Commun. Comput. Phys*, 3(2):295–307, 2008.

[71] T. Liggett. *Continuous Time Markov Processes: An Introduction.* American Mathematical Society, 2010.

[72] J. V. F. Lima, N. Maillard, F. Broquedis, T. Gautier, and B. Raffin. Performance evaluation of intel xeon phi coprocessor using xkaapi. *Workshop on Parallel and Distributed Processing*, 2013.

[73] P. Liu, R. Song, G. L. Elison, W. Peng, and M. Acar. Noise reduction as an emergent property of single-cell aging. *Nature communications*, 8:680, Sept. 2017.

[74] S. Manabe and K. Bryan. Climate calculations with a combined ocean-atmosphere model. *Journal of the Atmospheric Sciences*, 26(4):786–789, 1969.

[75] X. Mao. The truncated eulermaruyama method for stochastic differential equations. *Journal of Computational and Applied Mathematics*, 290(Supplement C):370–384, 2015.

[76] X. Mao and C. Yuan. *Stochastic differential equations with Markovian switching.* Imperial College Press, London, 2006.

[77] G. K. Marinov, B. A. Williams, K. McCue, G. P. Schroth, J. Gertz, R. M. Myers, and B. J. Wold. From single-cell to cell-pool transcriptomes: stochasticity in gene expression and RNA splicing. *Genome Res*, 24:496–510, Mar. 2014.

[78] H. Meinhardt. Models for the Generation and Interpretation of Gradients. *Cold Spring Harbor Perspectives in Biology*, 1:a001362, 2009.

[79] J. K. Moller and H. Madsen. From state dependent diffusion to constant diffusion in stochastic differential equations by the lamperti transform. Report, Technical University of Denmark, DTU Informatics, Building 321, 2010.

[80] K. Niederreither and P. Dolle. Retinoic acid in development: towards an integrated view. *Nature reviews. Genetics*, 9:541–53, July 2008.

[81] K. R. Norbert Hofmann, Thomas Muller-Gronbach. Optimal approximation of stochastic differential equations by adaptive step-size control. *Mathematics of Computation*, 69(231):1017–1034, 2000.

[82] ODE.jl. 9/21/2016. https://github.com/JuliaDiffEq/ODE.jl.

[83] B. Oksendal. *Stochastic Differential Equations: An Introduction with Applications.* Springer, 2003.

[84] C. B. Paul. Stochastic switching in biology: from genotype to phenotype. *Journal of Physics A: Mathematical and Theoretical*, 50:133001, 2017.

[85] J. Paulsson. Summing up the noise in gene networks. *Nature*, 427:415–418, 2004.

[86] F. Pérez and B. E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.

[87] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In S. Gomez and J.-P. Hennart, editors, *Advances in Optimization and Numerical Analysis, Proceedings of the 6th Workshop on Optimization and Numerical Analysis, Oaxaca, Mexico*, volume 275, pages 51–67. Kluwer Academic Publishers.

[88] 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.

[89] C. Rackauckas. Interfacing with a xeon phi via julia. *StochasticLifestyle.com*, 2016.

[90] C. Rackauckas and Q. Nie. Adaptive methods for stochastic differential equations via natural embeddings and rejection sampling with memory. *Discrete and Continuous Dynamical Systems - Series B*, 22(7):2731–2761, 2016.

[91] C. Rackauckas and Q. Nie. Adaptive methods for stochastic differential equations via natural embeddings and rejection sampling with memory. *Submitted*, 2016.

[92] C. Rackauckas and Q. Nie. Differentialequations.jl - a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, 5(1):15, 2017.

[93] C. Rackauckas, T. Schilling, and Q. Nie. Mean-independent noise control of cell fates via intermediate states. *iScience*, 3:11–20, 2018.

[94] A. Raj and A. van Oudenaarden. Nature, nurture, or chance: stochastic gene expression and its consequences. *Cell*, 135:216–26, 2008.

[95] A. Rossler. Runge kutta methods for the strong approximation of solutions of stochastic differential equations. *SIAM Journal on Numerical Analysis*, 48(3):922–952, 2010.

[96] T. P. Runarsson and Y. Xin. Search biases in constrained evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(2):233–243, 2005.

[97] T. Ryden and M. Wiktorsson. On the simulation of iterated ito integrals. *Stochastic Processes and their Applications*, 91(1):151–168, 2001.

[98] A. Saarinen, M.-L. Linne, and O. Yli-Harja. Stochastic differential equation model for cerebellar granule cell excitability. *PLoS Comput Biol*, 4(2):e1000004, 2008.

[99] N. Safarov and C. Atkinson. Natural gas storage valuation and optimisation under time-inhomogeneous exponential lvy processes. *International Journal of Computer Mathematics*, 2015.

[100] T. Schaffter. *From genes to organisms: Bioinformatics System Models and Software.* Thesis, 2014.

[101] T. F. Schilling, Q. Nie, and A. D. Lander. Dynamics and precision in retinoic acid morphogen gradients. *Curr Opin Genet Dev*, 22:562–9, Dec. 2012.

[102] L. F. Shampine. Stiffness and nonstiff differential equation solvers, ii: Detecting stiffness with runge-kutta methods. *ACM Trans. Math. Softw.*, 3(1):44–53, 1977.

[103] L. F. Shampine. Some practical runge-kutta formulas. *Mathematics of Computation*, 46(173):135–150, 1986.

[104] L. F. Shampine and P. Gahinet. Delay-differential-algebraic equations in control theory. *Appl. Numer. Math.*, 56(3-4):574–588, 2006.

[105] L. F. Shampine and K. L. Hiebert. Detecting stiffness with the fehlberg (4, 5) formulas. *Computers & Mathematics with Applications*, 3(1):41–46, 1977.

[106] L. F. Shampine and M. W. Reichelt. The matlab ode suite.

[107] W. H. E. Sharp, D. J. Higham, B. Owren, and P. W. A survey of the explicit runge-kutta method. 1995.

[108] A. Singh, B. S. Razooky, R. D. Dar, and L. S. Weinberger. Dynamics of protein noise can distinguish between alternate sources of gene-expression variability. *Mol Syst Biol*, 8:607, 2012.

[109] I. O. Sirbu, L. Gresh, J. Barra, and G. Duester. Shifting boundaries of retinoic acid activity control hindbrain segmental gene expression. *Development*, 132:2611, 2005.

[110] G. Soderlind and L. Wang. Evaluating numerical ode/dae methods, algorithms and software. *Journal of Computational and Applied Mathematics*, 185(2):244–260, 2006.

[111] J. Sosnik, L. Zheng, C. V. Rackauckas, M. Digman, E. Gratton, Q. Nie, and T. F. Schilling. Noise modulation in retinoic acid signaling sharpens segmental boundaries of gene expression in the embryonic zebrafish hindbrain. *eLife*, 5, 2016.

[112] J. Sosnik, L. Zheng, C. V. Rackauckas, M. Digman, E. Gratton, Q. Nie, and T. F. Schilling. Noise modulation in retinoic acid signaling sharpens segmental boundaries of gene expression in the embryonic zebrafish hindbrain. *eLife*, 5:e14034, 2016.

[113] SymEngine. 9/21/2016. *v0.2.0*, https://github.com/symengine/symengine.

[114] C. H. Ta, Q. Nie, and T. Hong. CONTROLLING STOCHASTICITY IN EPITHELIAL-MESENCHYMAL TRANSITION THROUGH MULTIPLE INTERMEDIATE CELLULAR STATES, 2016.

[115] M. Thattai and A. van Oudenaarden. Intrinsic noise in gene regulatory networks. *Proceedings of the National Academy of Sciences*, 98:8614–8619, 2001.

[116] C. Tsitouras. Runge-kutta pairs of order 5(4) satisfying only the first column simplifying assumption. *Computers & Mathematics with Applications*, 62(2):770–775, 2011.

[117] P. J. van der Houwen. Explicit runge-kutta formulas with increased stability boundaries. *Numerische Mathematik*, 20(2):149–164, 1972.

[118] P. van Galen, A. Kreso, E. Wienholds, E. Laurenti, K. Eppert, E. R. Lechman, N. Mbong, K. Hermans, S. Dobson, C. April, J. B. Fan, and J. E. Dick. Reduced lymphoid lineage priming promotes human hematopoietic stem cell expansion. *Cell Stem Cell*, 14:94–106, Jan. 2014.

[119] J. H. Verner. Explicit runge kutta pairs with lower stage-order. *Numerical Algorithms*, 65(3):555–577, 2013.

[120] B. Wang and Q. Qi. Modeling the lake eutrophication stochastic ecosystem and the research of its stability. *Mathematical Biosciences*.

[121] P. Wang and Y. Li. Split-step forward methods for stochastic differential equations. *Journal of Computational and Applied Mathematics*, 233(10):2641–2651, 2010.

[122] Q. Wang, W. R. Holmes, J. Sosnik, T. Schilling, and Q. Nie. Cell Sorting and Noise-Induced Cell Plasticity Coordinate to Sharpen Boundaries between Gene Expression Domains. *PLoS Computational Biology*, 13:e1005307, Jan. 2017.

[123] X. Wang, B. Errede, and T. C. Elston. Mathematical Analysis and Quantification of Fluorescent Proteins as Transcriptional Reporters. *Biophysical Journal*, 94(6):2017–2026, Mar. 2008.

[124] O. Wartlick, A. Kicheva, and M. Gonzalez-Gaitan. Morphogen gradient formation. *Cold Spring Harbor Perspectives in Biology*, 1:a001255, Sept. 2009.

[125] R. J. White, Q. Nie, A. D. Lander, and T. F. Schilling. Complex regulation of cyp26a1 creates a robust retinoic acid gradient in the zebrafish embryo. *PLoS Biology*, 5:e304, Nov. 2007.

[126] R. J. White and T. F. Schilling. How degrading: Cyp26s in hindbrain development. *Developmental dynamics : an official publication of the American Association of Anatomists*, 237:2775–2790, 2008.

[127] M. Wiktorsson. Joint characteristic function and simultaneous simulation of iterated ito integrals for multiple independent brownian motions. *The Annals of Applied Probability*, pages 470–487, 2001.

[128] F. Wu, R.-Q. Su, Y.-C. Lai, and X. Wang. Engineering of a synthetic quadrastable gene network to approach Waddington landscape and cell fate determination. *eLife*, 6, Apr. 2017.

[129] M. Wu, R.-Q. Su, X. Li, T. Ellis, Y.-C. Lai, and X. Wang. Engineering of regulated stochastic cell fate determination. *Proceedings of the National Academy of Sciences*, 110(26):10610–10615, June 2013.

[130] L. Xiao, Q. Cai, Z. Li, H. Zhao, and R. Luo. A multi-scale method for dynamics simulation in continuum solvent models. i: Finite-difference algorithm for navierstokes equation. *Chemical Physics Letters*, 616617:67–74, 2014.

[131] L. Zhang, K. Radtke, L. Zheng, A. Q. Cai, T. F. Schilling, and Q. Nie. Noise drives sharpening of gene expression boundaries in the zebrafish hindbrain. *Mol Syst Biol*, 8:613, 2012.