

The Analysis and Implementation of Exponential Almost Runge-Kutta Methods for Semilinear Problems

Eóin O'Callaghan

B.Sc.

`eoin.ocallaghan3@mail.dcu.ie`

A Dissertation Submitted for the Degree of Doctor of Philosophy
Dublin City University

Supervisor

Prof. John Carroll

School of Mathematical Sciences

Dublin City University

`john.carroll@dcu.ie`

September 2011

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy in Mathematics is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: _____
Eóin O'Callaghan Date

ID Number: 51711288

Date: 19 September 2011

Contents

List of Tables	vii
List of Figures	viii
Abstract	x
Acknowledgements	xi
1 Introduction	1
1.1 Stiff Differential Equations	2
1.2 Exponential Time-Integrator (EI) History	3
1.3 Exact Solution and φ -functions	6
2 ERK and EGLM Families of EIs	9
2.1 Exponential Runge-Kutta Methods	10
2.1.1 Order Conditions	11
2.1.2 1 st Order Consistency Conditions	12
2.1.3 2 nd Order Conditions	13
2.1.4 4-Stage, 4 th Order Conditions	15
2.2 Exponential General Linear Methods	25
2.2.1 3-Stage 4 th Order Conditions	26
2.2.2 Examples	27
3 Multi-value Families of EIs	29
3.1 Exponential Almost Runge-Kutta Methods	30
3.1.1 Deriving EARKs	32

3.1.2	3-Stage 4 th Order Conditions	32
3.1.3	Outgoing Approximations	33
3.1.4	Examples	34
3.2	Exponential Almost General Linear Methods	35
3.2.1	3-Stage 6 th Order Conditions	35
3.2.2	Outgoing Approximations	46
3.2.3	Example	47
3.3	Convergence Bounds	49
3.3.1	Order Conditions	49
3.3.2	Proof of Convergence Bound	53
3.3.3	Equivalence between EGLMs and EAGLMs	55
3.3.4	Summary	56
4	Stability	58
4.1	Stability Analysis	59
4.2	Comparisons with mixed Explicit-Implicit Schemes	59
4.3	Stability of ERKs	60
4.4	Unconditional Stability of ERKs	62
4.5	Stability of EGLMs	65
4.6	Stability of EARKs	66
4.7	Summary	67
5	Numerical Experiments	69
5.1	Test Problems	70
5.1.1	The Kuramoto-Sivashinsky Equation	70
5.1.2	Brusselator System	71
5.1.3	The Allen-Cahn Equation	71
5.1.4	Hochbruck & Ostermann Parabolic PDE	72
5.1.5	The Reaction Diffusion Advection (RDA) 2D Equation	73
5.1.6	The Gray-Scott Equation	73
5.2	Computing φ -functions	75
5.2.1	Padé approximations	75

5.2.2	Krylov Subspace Methods	77
5.2.3	Real Leja Points Method	79
5.2.4	Contour Integration	81
5.2.5	Conclusions	84
5.3	Convergence Order	90
5.3.1	Two-Stage ERKs	90
5.3.2	Three-Stage ERKs	91
5.3.3	Four-Stage ERKs	94
5.3.4	EGLMs	97
5.3.5	EARKs and EAGLMs	98
5.3.6	Summary	101
5.4	Variable Step Size	102
5.4.1	Basic Requirements	102
5.4.2	Truncation Error Estimation	105
5.4.3	Stepsize Adjustment	109
5.4.4	Initial Stepsize and Starting Points	113
5.5	Benchmarks	115
5.5.1	Brusselator System	115
5.5.2	Allen-Cahn Equation	116
5.5.3	The Kuramoto-Sivashinsky Equation	118
5.5.4	The RDA 2D Equation	119
5.5.5	The Gray-Scott Equation	122
5.6	ODE15s CPU Cost Scaling	126
6	Conclusions	127
A	Definitions	129
A.1	Parabolic Evolution Equations	129
A.1.1	Analytical Framework	129
A.2	B-Convergence	130

B Complete Results	132
B.1 Brusselator System Tables	132
B.2 Allen-Cahn Equation Tables	134
B.2.1 Chebyshev Differentiation Matrix	134
B.2.2 Finite-Differences	135
B.3 1D Kuramoto-Sivashinsky Equation Tables	136
B.4 RDA 2D Equation Tables	137
B.5 Gray-Scott Equation Tables	139
B.5.1 1-Dimensional Problem	139
B.5.2 2-Dimensional Problem	141
B.5.3 3-Dimensional Problem	142
C Alternative Order Condition Proofs	143
C.1 3-Stage, 6 th Order EAGLMs	143
D Acronyms	149
Bibliography	150

List of Tables

2.1	Computational inspection of Scheme 2.1.42	22
2.2	Computational inspection of Scheme (2.1.43)	23
2.3	Computational inspection of Scheme (2.1.46)	24
5.1	Padé timings in seconds using PHIPADE from the EXPINT package, $h = 1 \times 10^{-2}$	76
5.2	PHIPM at tolerance 1×10^{-13} with $h = 1 \times 10^{-2}$	79
5.3	Real Leja Points Method (ReLPM) at tolerance 1×10^{-13} with $h = 1 \times 10^{-2}$. . .	81
5.4	Computational timings for $e^L v$ in seconds	83
5.5	Rational Approximations with $h = 1 \times 10^{-2}$	84
5.6	Relative performance measure for the 2 and 3-Stage Exponential Runge-Kutta Methods (ERKs)	94
5.7	Timings for 256 Timesteps, with Exponential Time Differencing Method (ETD) Euler and the 2 & 3-Stage ERKs	94
5.8	Relative performance measure for the 4-Stage ERKs	97
5.9	Relative performance measure for the 4-Stage Exponential General Linear Methods (EGLMs)	98
5.10	Relative performance measure for the 4-Stage Exponential Almost General Linear Methods (EAGLMs)	99
5.11	Scheme (3.2.30) with all Controllers for Problem (5.1.3)	112
5.12	1D Brusselator System $N = 256$, Step Counts	115
5.13	Allen-Cahn, Global Error = 2×10^7 , Step Counts	118
5.14	Kuramoto-Sivashinsky Equation, $N = 256$, Step Counts	119
5.15	2D RDA Equation, Step Counts, $\varepsilon = 0.05$, $\alpha = -1$	121
5.16	Memory Usage in Megabytes (MBs)	123
5.17	Gray-Scott Equation, Global Error = 1×10^{-6}	125

List of Figures

4.1	Mixed explicit-implicit schemes stability boundaries for $y = -2$	60
4.2	ERK ₂ (2.1.19) stability graphs	61
4.3	ETD2 (1.3.14) stability boundaries	62
4.4	EGLM _{322c2} (2.2.6) Stability Plots	66
4.5	Exponential Almost Runge-Kutta Method (EARK) _{321c2} (3.2.30) Stability Plots .	66
4.6	EGLM _{322c2} (2.2.6) and EARK _{321c2} (3.2.30) side-by-side stability regions	67
4.7	EGLM _{322c2} (2.2.6) with scaled EARK _{321c2} (3.2.30) stability region	67
5.1	The 1D Kuramoto-Sivashinsky Equation Solution (5.1.1)	70
5.2	The Brusselator System Solution (??)	71
5.3	The Allen-Cahn Equation (5.1.5)	72
5.4	Rendering of 2D RDA Equation with $\varepsilon = 0.05, \alpha = -1, \rho = 100$	73
5.5	The 1D Gray-Scott Equation (5.1.13) for $u(t, x)$	74
5.6	φ Timings for the Hochbruck & Ostermann Parabolic PDE, $h = 0.001$	86
5.7	φ Timings for the 2D RDA Equation, $h = 0.001$	87
5.8	φ Timings for the Kuramoto-Sivashinsky Equation, $h = 0.001$	88
5.9	φ Timings for the Allen-Cahn Problem, $h = 0.001$	89
5.10	Fixed step integration of Problem 5.1.9	91
5.11	CPU Timings against Global Error for Problem 5.1.9	94
5.12	Fixed stepsize ERKs for Problem 5.1.9	96
5.13	Fixed stepsize EGLMs for Problem 5.1.9	98
5.14	Fixed stepsize EAGLMs for Problem 5.1.9	99
5.15	Allen-Cahn Equation, fixed stepsize, $N = 50$	100
5.16	Kuramoto-Sivashinsky Equation, fixed stepsize	101

5.17 Illustration of Local Error Profiles	108
5.18 Scheme (3.2.30) with the Stepping Controller for Problem (5.1.3)	110
5.19 Scheme (3.2.30) with the Guided Controller for Problem (5.1.3)	111
5.20 Scheme (3.2.30) with the I-controller for Problem (5.1.3)	112
5.21 Illustration of Different β Exponents with $l = 16$ (5.4.19)	114
5.22 Brusselator System, adaptive stepsize, $N = 64$	115
5.23 Brusselator System, Global Error against CPU Time without ODE15s	116
5.24 Allen-Cahn Equation, adaptive stepsize, $N = 50, \varepsilon = 0.001$	116
5.25 Allen-Cahn Equation, adaptive stepsize without ODE15s, $N = 50, \varepsilon = 0.001$	117
5.26 Allen-Cahn Equation “Type 1”, $N = 512, \varepsilon = 0.001$	117
5.27 Allen-Cahn Equation “Type 2”, $N = 512, \varepsilon = 0.001$	118
5.28 Kuramoto-Sivashinsky Equation, $N = 256$	119
5.29 2D RDA Equation, $N = 20 \times 20, \varepsilon = 0.05, \alpha = -1, \rho = 1$	120
5.30 2D RDA Equation, $N = 64 \times 64, \varepsilon = 0.05, \alpha = -1, \rho = 1$	120
5.31 2D RDA Equation without ODE15s, $\varepsilon = 0.05, \alpha = -1, \rho = 1$	120
5.32 2D RDA Equation, CPU Timings, $\varepsilon = 0.05, \alpha = -1, \rho = 1$	121
5.33 2D RDA Equation, $N = 32 \times 32, \varepsilon = 0.05, \alpha = -1, \rho = 100$	122
5.34 2D RDA Equation without ODE15s, $\varepsilon = 0.05, \alpha = -1, \rho = 100$	122
5.35 2D RDA Equation, CPU Timings, $\varepsilon = 0.05, \alpha = -1, \rho = 100$	122
5.36 1D Gray-Scott, $N = 512$	123
5.37 1D Gray-Scott, $N = 1024$	123
5.38 1D Gray-Scott, CPU Timings against Problem Size	124
5.39 2D Gray-Scott, $N = 32 \times 32$	124
5.40 2D Gray-Scott, CPU Timings against Problem Dimension	125
5.41 3D Gray-Scott Equation, $n = 8 \times 8 \times 8$	125
5.42 Effect of matrix dimension on ODE15s computational cost	126

Abstract

Recently there has been a great deal of interest in the construction of Exponential Time-Integrators (EIs) for semilinear problems. EIs in particular are well suited to the numerical integration of stiff ODEs arising from the spatial discretisation of PDEs. For such problems, stability issues rather than accuracy requirements dominate the choice of stepsize. Established methods for solving stiff problems have revolved around *implicit* schemes, for example, implicit Runge-Kutta Methods (RKs). EIs offer an *explicit* alternative.

We study and test the performance of numerical schemes derived from EIs, for the integration of large stiff systems of non-linear initial value problems, typically PDEs,

$$y_t = \mathcal{L}y + \mathcal{N}(y, t), \quad y(t_0) = y_0$$

where \mathcal{L} is an unbounded linear operator and \mathcal{N} is a non-linear operator. Our field of interest is primarily with the system of semi-linear ODEs,

$$y'(t) = Ly(t) + N(t, y(t)), \quad y(t_n) = y_n$$

obtained after a space discretisation of the PDE. The linear part, L , of the equation is stiff, while the non-linear part, N , is assumed to be non-stiff, in the sense that it can be approximated by an explicit method.

EIs, as their name suggests, use the exponential function, and related functions known as φ -functions, of L , or an approximation to it, inside the numerical method. Though not a new idea, it is only with recent developments in computing the exponential of a matrix have EIs become practical. Now, with these new methods, renewed interest in Exponential Integrators has been observed, and EIs are now proving themselves to be competitive against existing algorithms.

Our work contains an overview of the main established families of EIs, those being Exponential Time Differencing Methods (ETDs), Exponential Runge-Kutta Methods (ERKs) and Exponential General Linear Methods (EGLMs). ETDs are a class of multi-step integrators, while ERKs are 1-step integrators using lower order intermediate steps as in classical Runge-Kutta schemes. EGLMs represent a framework that can combine multi-step and multi-stage approaches and within EGLMs the ETD and ERK families become a special case.

We introduce a new family of EIs, namely Exponential Almost Runge-Kutta Methods (EARKs). Like EGLMs, they retain the ERK's concept of multiple stages but, instead of the multi-step nature of EGLMs, the input and output values passed from step-to-step are function evaluations of the approximate solution, $N(y_{n+1})$, followed by increasing derivatives, $N_n^{(i)}$. In this sense, the schemes are multi-value rather than multi-step. Also a broader family, named Exponential Almost General Linear Methods (EAGLMs), is introduced, which represent a unifying framework within which all the families of EIs become special cases. The convergence properties of EIs are studied with particular interest in our EARKs. A number of the methods, which had been selected for further investigation, have their stability properties studied.

While carrying out a survey of the new methods for calculating the φ -functions we perform a comprehensive suite of numerical experiments. By concentrating initially on fixed stepsize experiments we confirm the order convergence properties of the various schemes involved in our earlier work. This allows us to plot order graphs and observe the accuracy performances of the schemes.

Our work culminates in the construction of a complete and robust variable-stepsize integrator. All the necessary aspects for a general purpose implementation are addressed. This includes initial stepsize selection and reliable truncation error estimation with minimal computational overhead. We also include intelligent stepsize control to both minimise rejected steps and manage the global error. At the heart of this engine is our most efficient EARK and we show that it is the most optimal scheme for this purpose. Our integrator is run on a number of test problems, representative of the PDEs we are interested in, and is benchmarked against the built in Matlab ODE solvers.

Acknowledgements

I would like to thank Prof. John Carroll, my supervisor, for his constant support during this research. I am indebted to him for his ongoing guidance, enthusiasm and encouragement.

I would like to thank the Dublin City University School of Mathematical Sciences for their financial support during the four years.

Many thanks also to my family and to my friends in DCU and elsewhere. Their support has been greatly appreciated.

Chapter 1

Introduction

In this chapter we will look at:

- The classes of problems we are interested in. Those problems exhibit a property known as *stiffness* and we will show why this property has motivated the development of Exponential Time-Integrators (EIs).
- The history behind the development of EIs. By investigating the work of various authors in this field, we will see how the formulation of EIs has evolved over the past few decades.
- How the model PDE for a scalar function y , defined in a spatial domain $\Omega \subset \mathbf{R}^d$ for $t > 0$,

$$y_t = \mathcal{L}y + \mathcal{N}(y, x, t) \tag{1.0.1}$$

- \mathcal{L} is linear elliptic operator
- \mathcal{N} is a generic nonlinear term

with suitable initial and boundary conditions [15], is represented within the EI framework and the key role which the so called φ -functions play in the approximation to the exact solution of (1.0.1).

1.1 Stiff Differential Equations

Stiff problems are a type of differential equation which are extremely difficult to solve with explicit schemes. A pragmatic definition attributed [21] to Curtiss & Hirschfelder [14], is

“Stiff equations are equations where certain implicit methods, in particular BDFs [Backward Differentiation Methods], perform better, usually tremendously better, than explicit ones.”

Stiff problems are usually characterised locally at a point (t, y) , by the spectrum of the Jacobian, $J(t, y) = \frac{\partial f}{\partial y}(t, y)$. A problem is usually called stiff if there exist eigenvalues, $\lambda_i(t, y)$, of $J(t, y)$ with $\text{Re}(\lambda_i(t, y)) \ll 0$, together with moderately sized eigenvalues [16].

When integrating stiff equations with an explicit classical scheme, the choice of stepsize, h , is governed by stability requirements, rather than accuracy. The standard solution to this problem is to use implicit schemes.

To demonstrate this, we look at a stability analysis of the explicit and implicit Euler’s methods. Following the format of [21], we start with

$$y' = f(t, y) \tag{1.1.1}$$

where $\phi(t)$ is a smooth solution of (1.1.1). We look at the linearisation of f in its neighbourhood about $\phi(t)$

$$f(y(t) - \phi(t)) = f(t, \phi(t)) + \frac{\partial f}{\partial y}(t, \phi(t))(y(t) - \phi(t)) + \dots \tag{1.1.2}$$

and introduce $y(t) - \phi(t) = \bar{y}(t)$ to obtain

$$\begin{aligned} \bar{y}'(t) &= \frac{\partial f}{\partial y}(t, \phi(t)) \bar{y}(t) + \dots \\ &= J(t) \bar{y}(t) + \dots \end{aligned} \tag{1.1.3}$$

We consider $J = -\lambda$, where $\lambda > 0$ is a constant scalar, and as an approximation, we neglect terms above 1st order. Simplifying notation, we arrive at

$$y' = -\lambda y \tag{1.1.4}$$

Explicit Euler. Applying the explicit Euler’s method

$$y_{n+1} = y_n + hf(t_n, y_n) \tag{1.1.5}$$

to (1.1.4) gives

$$y_{n+1} = R(h\lambda)y_n \tag{1.1.6}$$

with $R(z) = 1 - z$.

We study the behaviour of (1.1.4) by looking at the equation

$$y_n = (R(h\lambda))^n y_0 \quad (1.1.7)$$

For y_n to remain bounded as $n \rightarrow \infty$, we require that the complex number $z = h\lambda$ satisfy

$$|1 - z| \leq 1 \quad (1.1.8)$$

From this analysis we can work out that, for the explicit Euler to remain stable, we must limit the stepsize to $0 \leq h \leq \frac{2}{\lambda}$.

Implicit Euler. To perform a similar stability analysis with an implicit method, we look at the implicit Euler method

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) \quad (1.1.9)$$

applied to (1.1.4). This gives us

$$y_{n+1} = R(h\lambda)y_n \quad (1.1.10)$$

with $R(z) = \frac{1}{1+z}$.

Looking at (1.1.7) with $R(z) = \frac{1}{1+z}$ we can see that y_m remains bounded as $m \rightarrow \infty$ for all $h \geq 0$. This tells us implicit Euler is unconditionally stable.

Implicit schemes require the solution of systems of equations at each timestep [29], which means the total work complexity required to perform the integration is $O(N^3)$, N^2 for the number of steps, and N for the work per step [21].

1.2 EI History

EIs offer an explicit alternative for the integration of stiff problems. Recently, there has been a great deal of interest in the construction of EIs. These integrators use the exponential function (and related functions) of the linear part of the problem, inside the numerical method.

Using the matrix exponential within a numerical integrator is not new. These exponential integrators were regarded as impractical until recent advances in evaluating the matrix exponential and, in particular, its operation upon a vector. For quite some time prior to these developments, EIs did not play a prominent role in applications. With the development of new methods for computing or approximating the exponential of a matrix (for example with Krylov subspace techniques), interest in EIs has increased.

Lawson Methods [28] An early approach for solving stiff problems was introduced by Lawson in 1967. Lawson looked at stiff problems of the form

$$y'(t) = f(t, y(t)), \quad y(0) = y_0 \quad (1.2.1)$$

with large Lipschitz constants. He worked with a related problem which could be solved by a traditional Runge-Kutta Method (RK) solver, and from which the solution of (1.2.1) could be deduced. By combining the integration of the related problem with the recovery of the original, an efficient solution could be obtained. He called this combination, a generalised Runge-Kutta process.

The idea of Lawson was to consider, as a related function,

$$z(t) = e^{-tA}y(t) \quad (1.2.2)$$

such that,

$$\begin{aligned} z'(t) &= e^{-tA} [f(t, e^{tA}z(t)) - Ae^{tA}z(t)] \\ z(0) &= y_0 \end{aligned} \quad (1.2.3)$$

Viewing (1.2.3) as $z' = g(t, z)$, it can be shown that the eigenvalues of the Jacobian $\left(\frac{\partial g}{\partial z}\right)$ are the same as those of $\left(\frac{\partial f}{\partial y}\right) - A$. One therefore tries to select A , such that the eigenvalues of $\left(\frac{\partial g}{\partial z}\right)$ are small enough to allow (1.2.3) to be solved with a traditional method. Lawson methods are more commonly referred to as Integrating Factor Methods (IFs) [34].

The IF approach can also be applied to semi-linear equations, the class of problems we are most interested in,

$$y'(t) = Ly(t) + N(t, y(t)), \quad y(t_n) = y_n \quad (1.2.4)$$

Considering a related function, $z(t) = e^{-tL}y(t)$, of (1.2.4), we get

$$\begin{aligned} z'(t) &= e^{-tL}N(e^{tL}z(t), t) = g(z, t) \\ z(0) &= y_0. \end{aligned} \quad (1.2.5)$$

Since

$$\frac{\partial g}{\partial z} = e^{-tL} \frac{\partial N}{\partial y} e^{tL} \quad \text{and} \quad e^{-tL} = (e^{tL})^{-1} \quad (1.2.6)$$

we would once again hope that (1.2.5) can be solved by a traditional method [33].

Ultimately, the main issue with Lawson-type methods is that they only achieve stiff order of 1, and are best suited to moderately stiff problems for which the solution is periodic, or tends towards zero. Furthermore, they do not preserve the fixed points of the original problem [34, 13].

Hochbruck, Lubich & Selhofer [24] looked at using the exponential of the Jacobian and related functions within a numerical integrator. By combining this with Krylov approximation techniques, they were able to compute

$$\varphi(\tau A)v$$

where A is the Jacobian of (1.2.1), and

$$\varphi(z) = \frac{e^z - 1}{z}$$

They formulated a number of methods based on this technique and presented a Matlab code, `exp4` for semilinear problems of the type (1.2.4), which was the first actual implementation of an EI. A number of numerical tests on stiff and oscillatory problems showed that the code performed well when compared with the Matlab ODE codes and some classical solvers [24, 34].

Beylkin, Keiser & Vozovoi [5] investigated the stability of classes of implicit and explicit Exponential Time Differencing Methods (ETDs), referring to them as *Exact* treatment of the *Linear Part*, L , Methods (ELPs) and presented a derivation for formulating such methods. They performed the stability analysis by establishing a baseline against an existing 3rd Order Adams-Moulton / Adams-Bashforth Method (AMAB) and a 3rd Order mixed Implicit-Explicit Method (I-EM).

By comparing both an implicit and explicit example of their ELPs against this baseline, they were able to draw conclusions about their relative stability. The comparison showed that the implicit ELP was “super-stable”, having a much larger stability region than the I-EM. The stability region of the explicit ELP was smaller than that of the implicit schemes, which meant that it would require timesteps of about half the length of the implicit stepsizes to remain stable. When compared with the classical schemes, the explicit ELP was comparable to the I-EM and was significantly more stable than the AMAB.

Cox & Matthews [13] presented an alternative formulation of ETDs, providing a more elegant derivation for constructing schemes of arbitrary order. Their approach was based on a polynomial approximation to $N(t_n + \tau, y(t_n + \tau))$.

Alongside this, they developed a 1-step RK type extension to ETDs which they referred to as ETD Runge-Kutta Methods (ETDRKs). These schemes are now described as belonging to the family of Exponential Runge-Kutta Methods (ERKs). Cox & Matthews provided schemes of 2nd, 3rd and 4th Order, and subjected them to a number of numerical tests. The results of these tests demonstrated that both ETDs and ERKs consistently outperformed IFs in terms of

accuracy. The 4th order scheme appears in a number of our own numerical experiments, where we refer to it as ERK₄ Cox-Matthews [13, Equations (26-29)] .

Krogstad [27] looked at overcoming the difficulties associated with IFs. For problems where the norm of the linear term is large, IFs produce large error coefficients. Krogstad proposed a generalisation of IFs displaying significantly improved accuracy.

Krogstad also looked at the construction of ERKs, and like Cox and Matthews, he referred to them as ETDRKs. In particular he developed the 4th Order ERK, that we will refer to as ERK₄ Krogstad [27, Equation (51)] throughout this thesis. This scheme will be included in many of our numerical experiments in Section 5.3.

Ostermann, Thalhammer & Wright [38] introduced the Exponential General Linear Method (EGLM) family of EIs. EGLMs are an extension of classical General Linear Methods (GLMs), which were introduced by Butcher in [7], into the EI setting. EGLMs represent a framework of multi-stage multi-step schemes, within which, the earlier families of ETDs and ERKs become special cases. By combining the advantages of both families, it is possible to achieve high stage order which facilitates the construction of high-order methods with favorable stability properties for stiff problems [38].

1.3 Exact Solution and φ -functions

We study and test the performance of numerical schemes derived from EIs, for the integration of large stiff systems of non-linear initial value problems, typically PDEs.

$$y_t = \mathcal{L}y + \mathcal{N}(y, t), \quad y(t_0) = y_0 \quad (1.3.1)$$

- \mathcal{L} is an unbounded linear operator independent of y ,
- \mathcal{N} is a non-linear operator

There are several well known examples of this type of problem, such as Allen-Cahn, Kuramoto-Sivashinsky and the Brusselator System. Our field of interest is primarily with the system of semi-linear ODEs,

$$y'(t) = Ly(t) + N(t, y(t)), \quad y(t_n) = y_n \quad (1.3.2)$$

obtained after a space discretisation of (1.3.1). The linear part, L , of the equation is stiff, while the non-linear part, N , is assumed to be non-stiff, in the sense that it can be approximated by an explicit method.

More formally, we consider ODEs of the type 1.3.2, where $L \in \mathbb{R}^{d \times d}$ has a large norm together with a non-positive or moderately positive logarithmic norm and $N : [t_0, +\infty) \times \mathbb{R}^d \rightarrow \mathbb{R}^d$ has a moderate Lipschitz constant with respect to the second argument [32].

Stiffness in the L part is inherent to the PDE. The system of ODEs from the discretisation of the PDE retain this stiffness. Finite differences and spectral methods are examples of discretisation approaches which can be used to solve stiff PDEs where the system of ODEs produced is generally stiff [13]. EIs solve the linear part exactly through the matrix exponential.

We derive the exact solution of (1.3.2) via the variation-of-constants formula. Pre-multiply with e^{-tL} (integrating factor) to obtain

$$\begin{aligned} e^{-tL}y' &= e^{-tL}Ly + e^{-tL}N(t, y), \\ (e^{-tL}y(t))' &= e^{-tL}N(t, y)dt. \end{aligned} \quad (1.3.3)$$

Now integrate,

$$e^{-tL}y_p(t) = \int e^{-tL}N(t, y)dt \quad (1.3.4)$$

add the homogeneous solution to (1.3.2), to the particular solution $y_p(t)$ and simplify

$$y(t) = e^{(t-a)L}y_a + e^{tL} \int_a^t e^{-\tau L}N(\tau, y(\tau))d\tau \quad (1.3.5)$$

$$\begin{aligned} y(t_n + h) &= e^{(t_n+h-t_n)L}y_n + e^{(t_n+h)L} \int_0^h e^{-(t_n+\tau)L}N(t_n + \tau, y(t_n + \tau))d\tau \\ &= e^{hL}y_n + \int_0^h e^{(h-\tau)L}N(t_n + \tau, y(t_n + \tau))d\tau. \end{aligned} \quad (1.3.6)$$

Substituting the Taylor expansion for $N(t_n + \tau, y(t_n + \tau))$

$$\sum_{m=0}^{\infty} \frac{\tau^m N^{(m)}(y(t_n))}{m!} \quad (1.3.7)$$

into (1.3.6), and by defining

$$\varphi_k(hL) = \frac{1}{h^k} \int_0^h e^{(h-\tau)L} \frac{\tau^{k-1}}{(k-1)!} d\tau \quad (1.3.8)$$

we can represent the exact solution for (1.3.2) as

$$y(t_{n+1}) = e^{hL}y_n + h\varphi_1N + h^2\varphi_2N' + h^3\varphi_3N'' + \dots \quad (1.3.9)$$

where $y(t_{n+1})$ denotes $y(t_n + h)$. It is also common to make reference to the exact solution at an intermediate point $t_n + c_i h$, which is represented as,

$$y(t_n + c_i h) = e^{c_i hL}y_n + c_i h\varphi_{1i}N + c_i^2 h^2\varphi_{2i}N' + c_i^3 h^3\varphi_{3i}N'' + \dots \quad (1.3.10)$$

where

$$\varphi_{ji} = \varphi_j(c_i hL) \quad (1.3.11)$$

A key element in the implementation of exponential integrators is the evaluation of the matrix exponential and exponential like functions, (1.3.8), commonly referred to as φ -functions in literature.

Integration by parts reveals that (1.3.8) obeys the recurrence relation

$$\begin{aligned}
\varphi_k(hL) &= \frac{1}{h^k} \int_0^h e^{(x-\tau)L} \frac{\tau^{k-1}}{(k-1)!} d\tau \\
&= \frac{1}{h^k} \left[\frac{-e^{(h-\tau)L} \tau^{k-1}}{(k-1)!L} \Big|_0^h - \int_0^h \frac{-e^{(x-\tau)L}}{L} \frac{\tau^{k-2}}{(k-2)!} d\tau \right] \\
&= \frac{1}{h(k-1)!L} + \frac{1}{hL} \frac{1}{h^{k-1}} \int_0^h e^{(x-\tau)L} \frac{\tau^{k-2}}{(k-2)!} d\tau \\
&= \frac{\varphi_{k-1}(hL) - \frac{1}{(k-1)!}}{hL}
\end{aligned} \tag{1.3.12}$$

where, for convenience it is common to define $\varphi_0(z) = e^z$. When $j = 1, 2$ and 3 the respective φ -functions are

$$\varphi_1(z) = \frac{e^z - 1}{z} \quad \varphi_2(z) = \frac{e^z - z - 1}{z^2} \quad \varphi_3(z) = \frac{e^z - z^2/2 - z - 1}{z^3}$$

When deriving ETDs, we approximate the function $N(t_n + \tau, u(t_n + \tau))$ within (1.3.6), by a Newton backward-difference interpolation polynomial. The simplest case is to approximate by a constant N_n . This gives us the ETD1 method.

$$u_{n+1} = e^{hL} u_n + h\varphi_1(hL)N_n, \quad \varphi_1(hL)(z) = \frac{e^z - 1}{z} \tag{1.3.13}$$

It is known as the ETD Euler method as it reduces to the classical Euler method when $L = 0$.

As a second example, $N(t_n + \tau, y(t_n + \tau))$ is approximated by the linear polynomial

$$N_n + \frac{\tau}{h}(N_n - N_{n-1})$$

giving us the two-step method known as ETD2.

$$y_{n+1} = e^{hL} y_n + h[\varphi_1(hL) + \varphi_2(hL)] N_n - h\varphi_2(hL)N_{n-1} \tag{1.3.14}$$

Chapter 2

ERK and EGLM Families of EIs

In this chapter we look at two different families of EIs, namely

- Exponential Runge-Kutta Methods (ERKs)
- Exponential General Linear Methods (EGLMs)

We derive order conditions allowing us to construct new schemes and look at the difficulties which surround the design of higher order schemes.

2.1 Exponential Runge-Kutta Methods

We saw in Section 1.2 that some early examples of ERKs were presented by Cox & Matthews [13]. ERKs are 1-step methods using lower order intermediate steps as in classical RKs. If we start with the general representation of our problem (1.3.2) we can then define a general explicit ERK for solving this class of problems

$$y_{n+1} = e^{hL}y_n + h \sum_{n=1}^q b_n(hL)K_n$$

$$K_i = N \left(e^{c_i hL} y_n + h \sum_{j=1}^{i-1} a_{ij}(c_i hL) K_j \right)$$

Classical (non-exponential) RK integrators are often written in a tableau format

$$\begin{array}{c|c} c & A \\ \hline & b \end{array} \qquad \begin{array}{c|c} 0 & \\ \hline \frac{1}{2} & \frac{1}{2} \\ \hline & 0 \quad 1 \end{array}$$

A modified form of this tableau will also be used to represent ERKs, for example the tableau

$$\begin{array}{c|cc|c} 0 & & & I \\ \hline \frac{1}{2} & \frac{1}{2}\varphi_{1,2} & & e^{\frac{1}{2}hL} \\ \hline & \varphi_1 - 2\varphi_2 & 2\varphi_2 & \end{array} \quad (2.1.1)$$

represents the method

$$K_1 = N_n$$

$$K_2 = N \left(e^{\frac{hL}{2}} y_n + \frac{h}{2} \varphi_{1,2} \left(\frac{hL}{2} \right) N_n \right) \quad (2.1.2)$$

$$y_{n+1} = e^{hL} y_n + h ((\varphi_1 - 2\varphi_2) K_1 + 2\varphi_2 K_2)$$

More generally, we can see here how the coefficients within 3-stage ERKs relate to the entries in the tableau representation

$$\begin{array}{c|cc|c} 0 & & & \\ \hline c_2 & a_{21} & & e^{c_2 hL} \\ \hline c_3 & a_{31} & a_{32} & e^{c_3 hL} \\ \hline & b_1 & b_2 & b_3 & e^{hL} \end{array} \quad (2.1.3)$$

$$K_1 = N_n$$

$$K_2 = N(t_n + c_2 h, e^{c_2 hL} y_n + h a_{21} K_1)$$

$$K_3 = N(t_n + c_3 h, e^{c_3 hL} y_n + h [a_{31} K_1 + a_{32} K_2])$$

$$y_{n+1} = e^{hL} y_n + h (b_1 K_1 + b_2 K_2 + b_3 K_3)$$

2.1.1 Order Conditions

For classical RKs, the order of a scheme is defined based on a Taylor expansion of the exact solution for $y(t_0 + h)$ and the approximation y_1 :

Definition 1. [20, Definition 1.2] A Runge-Kutta method has *order* p if for sufficiently smooth problems (1.1.1),

$$\|y(t_0 + h) - y_1\| \leq Kh^{p+1} \quad (2.1.4)$$

i.e., if the Taylor series for the exact solution $y(t_0 + h)$ and for y_1 coincide up to (and including) the term h_p .

The above definition defines order in a classical (nonstiff) sense. We are interested in working with stiff equations of the form (1.3.2) and will construct schemes based on the concept of stiff order,

Definition 2. [32, Definition 1] An exponential method is said to be of *stiff order* p if the local error has order $p + 1$ with respect to h_n when the method is applied to the general semilinear ODE (1.3.2) in which $N(t, y(t))$ is a sufficiently smooth function of t . Stiff convergence order describes the behaviour of the local error independently of the norm of L .

It is not possible in general, to obtain order by simply using a RK for the nonlinear part N of the problem. There are coupling conditions between the nonlinear and linear parts of the problem despite the fact that the linear part has been solved exactly [33].

When deriving order conditions for constructing schemes of higher order, we will see that it is sometimes not possible to satisfy all the conditions unless we *weaken* some of them. We will therefore make a distinction between schemes of strong and weak stiff order.

Definition 3. An exponential method is said to be of *strong order* p if it satisfies the p stiff order conditions for the general semi-linear ODE (1.3.2).

Definition 4. An exponential method is said to be of *weak order* p if it has strong order $p - 1$ and satisfies the p stiff order conditions only in the weakened case where $L = 0$.

Hochbruck & Ostermann developed an approach based on trees for deriving stiff order conditions. Their approach represents the elementary differentials of $F(u) = Lu + N(u)$ which come from a Taylor expansion of the exact solution [25]. This allowed them to develop conditions for arbitrary orders in a way similar to classical RKs.

We adopt the approach whereby we view the Taylor expansions of the method's internal stages, K_i , on 2-dimensional grids. The columns of the grids group terms by the order of their

h coefficient, while the rows group terms relative to successive derivatives of N . Within this interpretation, operations upon the entries in the grid become simple shifts in locations.

To illustrate this we will view a general K_i in the following format where R_i is some residual term.

$$K_i = N(t_n + c_i h, y(t_n + c_i h) + R_i) \quad (2.1.5)$$

Then the Taylor expansion of (2.1.5) can be viewed more naturally in a 2-dimensional sense

$$K_i = \begin{cases} N & + & R_i N_y & + & R_i^2 N_{yy} & + & R_i^3 N_{yyy} & \cdots \\ + & & + & & + & & \vdots \\ c_i h N' & + & R_i c_i h N'_y & + & R_i^2 c_i h N'_{yy} & \cdots \\ + & & + & & \vdots \\ \frac{1}{2} c_i^2 h^2 N'' & + & \frac{1}{2} R_i c_i^2 h^2 N''_y & \cdots \\ + & & \vdots \\ \frac{1}{6} c_i^3 h^3 N''' & \cdots \\ \vdots \end{cases} \quad (2.1.6)$$

2.1.2 1st Order Consistency Conditions

We will consider an ERK of the form

$$\begin{aligned} y_{n+1} &= e^{hL} y_n + h \sum_{i=1}^q b_i(hL) K_i \\ K_1 &= N(t_n, y_n) \\ K_i &= N\left(t_n + c_i h, e^{c_i hL} y_n + \sum_{j=1}^{i-1} h a_{ij} K_j\right) \end{aligned} \quad (2.1.7)$$

when applied to approximate the solution of problem (1.3.2). The conditions for a method to be of order 1 are usually referred to as the *consistency conditions* [20]. To derive these conditions we will require that the method be exact for the case $N(y(t)) = N$, a constant. This means, that the internal stages,

$$y(t_n + c_i h) = e^{c_i hL} y_n + \sum_{j=1}^{i-1} h a_{ij} K_j \quad (2.1.8)$$

and y_{n+1} are exact. Let Y_{ni} denote the ERK approximation to $y(t_n + c_i h)$,

$$Y_{ni} = e^{c_i hL} y_n + \sum_{j=1}^{i-1} h a_{ij} N$$

where the exact solution (1.3.10) is

$$y(t_n + c_i h) = e^{c_i hL} y_n + c_i h \varphi_{1,i} N$$

for $Y_{ni} = y(t_n + c_i h)$, we must have

$$\sum_{j=1}^{i-1} a_{ij} = c_i \varphi_{1,i} \quad (2.1.9a)$$

For ERK, the approximation y_{n+1} ,

$$y(t_{n+1}) = e^{hL} y_n + \sum_{i=1}^S h b_i N$$

to equal the exact solution (1.3.9)

$$e^{hL} y_n + h \varphi_1 N$$

where S is the number of stages, we require the condition

$$\sum_{i=1}^S b_i = \varphi_1 \quad (2.1.9b)$$

For non-constant N , a scheme satisfying these conditions will have an error of order h^2 or more and so will be of, at least, stiff-order 1 for the general problem (1.3.2).

2.1.3 2nd Order Conditions

Theorem 1. *A 2-stage Exponential Runge-Kutta of the form*

$$\begin{array}{c|cc|c} 0 & 0 & 0 & 1 \\ c_2 & a_{21} & 0 & e^{c_2 h L} \\ \hline & b_1 & b_2 & e^{h L} \end{array} \quad (2.1.10)$$

$$K_1 = N(t_n, y_n) \quad (2.1.11)$$

$$K_2 = N(t_n + c_2 h, e^{c_2 h L} y_n + h a_{21} K_1)$$

$$y_{n+1} = e^{hL} y_n + h(b_1 K_1 + b_2 K_2) \quad (2.1.12)$$

will be of 2nd order if it satisfies the condition (2.1.9), and if

$$b_2 c_2 = \varphi_2 \quad (2.1.13)$$

Proof. Using the notation $N \equiv N(t_n, y(t_n))$, from (2.1.11) we have,

$$K_2 = N(t_n + c_2 h, e^{c_2 h L} y_n + h a_{21} N) \quad (2.1.14)$$

and require,

$$N(t_n + c_2 h, e^{c_2 h L} y_n + h a_{21} N) = N(t_n + c_2 h, y(t_n + c_2 h) + R_2) \quad (2.1.15)$$

where R_2 is a residual term of order h^3 . Replacing $y(t_n + c_2h)$ with the exact solution expansion (1.3.10) we get,

$$R_2 = e^{c_2hL}y_n + ha_{21}N - e^{c_2hL}y_n - c_2h\varphi_{1,2}N + O(h^3) \quad (2.1.16)$$

This implies that $c_2\varphi_{1,2} = a_{21}$, which recovers the consistency condition (2.1.9a).

Substituting for (2.1.16) into K_2 this process is then repeated using the Taylor expansion about $y(t_n + c_2h)$

$$\begin{aligned} K_2 &= N(t_n + c_2h, e^{c_2hL}y_n + ha_{21}N) \\ &= N(t_n + c_2h, e^{c_2hL}y_n + hc_2\varphi_{1,2}N) \\ &= N(t_n + c_2h, y(t_n + c_2h) + R_2) \\ &= N(t_n + c_2h, y(t_n + c_2h)) + R_2N_y(t_n + c_2h, y(t_n + c_2h)) + O(h^4). \end{aligned}$$

Then, expanding K_2 about t_n ,

$$\begin{aligned} K_2 &= N + c_2hN' + \frac{1}{2}c_2^2h^2N'' + \frac{1}{6}c_2^3h^3N''' \\ &\quad + R_2N_y + c_2hR_2N'_y + O(h^4) \\ &= N + c_2hN' + \frac{1}{2}c_2^2h^2N'' + R_2N_y + O(h^3). \end{aligned} \quad (2.1.17)$$

Here, the pattern key to our grid approach of representing the expansions is beginning to emerge. When deriving higher order conditions, we are required to retain higher powers of h from the Taylor expansion. For those derivations, the grid representation will then be used.

Returning then to the general form of the numerical method (2.1.10), we substitute K_1 and (2.1.17) into (2.1.11) to obtain

$$\begin{aligned} y_{n+1} &= e^{hL}y_n + h(b_1K_1 + b_2K_2) \\ &= e^{hL}y_n + hb_1N + hb_2N + b_2c_2h^2N' + \frac{1}{2}b_2c_2^2h^3N'' + hb_2R_1N_y + O(h^4) \\ &= e^{hL}y_n + h(b_1 + b_2)N + b_2c_2h^2N' + O(h^3) \end{aligned} \quad (2.1.18)$$

Finally, by matching terms to ensure y_{n+1} coincides with the exact solution (1.3.9) up to $O(h^3)$, we recover the remaining conditions (2.1.9b) and (2.1.13). \square

Theorem 1 demonstrates that two-stage ERKs can achieve 2nd order, constrained by 3 conditions, (2.1.9a), (2.1.9b) and (2.1.13), for 4 unknowns.

We can view c_2 as a free parameter giving the following family of ERK₂ methods,

0		I
c_2	$c_2\varphi_{1,2}$	e^{c_2hL}
	$\varphi_1 - \frac{1}{c_2}\varphi_2 \quad \frac{1}{c_2}\varphi_2$	

(2.1.19)

Here, we have derived the same order conditions as presented by Hochbruck & Ostermann [25, Equations 5.1 & 5.2]. ERK₂ (2.1.19) is also presented in that same paper [25, Method 5.3].

This method requires 2 distinct φ -functions if $c_2 = 1$ and 3 if $c_2 \neq 1$. There is a possibility here to reduce this computational cost to 1 and 2 distinct φ -functions for $c_2 = 1$ and $c_2 \neq 1$ respectively if we weaken the condition (2.1.13) to

$$b_2(0)c_2 = \varphi_2(0) = \frac{1}{2} \quad (2.1.20)$$

This gives us a second 1-parameter family of ERK₂ [25, Scheme 5.4] schemes

$$\begin{array}{c|cc|c} 0 & & & I \\ c_2 & c_2\varphi_{1,2} & & e^{c_2 hL} \\ \hline & (1 - \frac{1}{2c_2})\varphi_1 & \frac{1}{2c_2}\varphi_1 & \end{array} \quad (2.1.21)$$

With condition (2.1.13) only satisfied weakly, this family of methods is not *strongly* 2nd order.

2-stage ERKs can achieve at most 2nd order. To achieve higher order it is necessary to consider methods with additional internal stages.

We will see that, as we aim for higher order, the number of conditions to be satisfied grows rapidly. This makes it increasingly difficult to achieve higher orders, and this difficulty makes the notion of weak order, as defined in Definition 4, very important. For example, we will see that there are no strongly 3rd order 3-stage ERKs, only weakly 3rd order methods. Still numerical comparisons of the relative performances of methods shows that weak order can be a desirable property. For example, the weakly 3rd order methods perform somewhere in between the strongly 2nd and strongly 3rd order methods.

2.1.4 4-Stage, 4th Order Conditions

We will now extend our formulation of (2.1.2) to include 3-stage and 4-stage schemes, and will derive the conditions for constructing schemes up to 4th order.

Theorem 2. *4-stage ERKs of the form*

$$\begin{array}{c|cccc|c} 0 & 0 & & & & I \\ c_2 & a_{21} & 0 & & & e^{c_2 hL} \\ c_3 & a_{31} & a_{32} & 0 & & e^{c_3 hL} \\ c_4 & a_{41} & a_{42} & a_{43} & 0 & e^{c_4 hL} \\ \hline & b_1 & b_2 & b_3 & b_4 & \end{array}$$

with the general format of the scheme being

$$\begin{aligned}
y_{n+1} &= e^{hL} y_n + h (b_1 K_1 + b_2 K_2 + b_3 K_3 + b_4 K_4) \\
K_1 &= N_n \\
K_2 &= N (t_n + c_1 h, e^{c_2 h L} + h a_{21} K_1) \\
K_3 &= N (t_n + c_1 h, e^{c_3 h L} + h a_{31} K_1 + h a_{32} K_2) \\
K_4 &= N (t_n + c_1 h, e^{c_4 h L} + h a_{41} K_1 + h a_{42} K_2 + h a_{43} K_3)
\end{aligned} \tag{2.1.22}$$

that satisfy (2.1.9) will achieve 2nd order if they also satisfy

$$b_2 c_2 + b_3 c_3 + b_4 c_4 = \varphi_2, \tag{2.1.23}$$

will achieve 3rd order if, in addition, they satisfy

$$\sum_{i=2}^4 b_i \left(\sum_{j=2}^{i-1} a_{ij} c_j - c_i^2 \varphi_{2,i} \right) = 0, \tag{2.1.24a}$$

$$b_2 c_2^2 + b_3 c_3^2 + b_4 c_4^2 = \varphi_3, \tag{2.1.24b}$$

and 4th order if

$$\sum_{i=3}^4 b_i \left[\sum_{j=2}^{i-1} a_{i2} \left(c_2^2 \varphi_{2,j} - \sum_{k=2}^{j-1} a_{jk} c_k \right) \right] = 0, \tag{2.1.25a}$$

$$\sum_{i=2}^4 b_i c_i \left(c_i^2 \varphi_{2,i} - \sum_{j=2}^{i-1} a_{ij} c_j \right) = 0, \tag{2.1.25b}$$

$$\sum_{i=2}^4 b_i \left(c_i^3 \varphi_{3,i} - \frac{1}{2} \sum_{j=2}^{i-1} a_{ij} c_j^2 \right) = 0, \tag{2.1.25c}$$

$$b_2 c_2^3 + b_3 c_3^3 + b_4 c_4^3 = 6\varphi_4, \tag{2.1.25d}$$

are also satisfied.

Proof. As in the earlier derivations we initially wish to express $e^{c_2 h L} + h a_{21} K_1$, from the K_2 stage (2.1.22), in the form $y(t_n + c_2 h) + R_2$. Using the exact solution (1.3.10) for $i = 2$,

$$\begin{aligned}
\Rightarrow e^{c_2 h L} + h a_{21} K_1 &= y(t_n + c_2 h) + R_2 \\
&= e^{c_2 h L} y_n + c_2 h \varphi_{12} N + c_2^2 h^2 \varphi_{2,2} N' + \dots + R_2
\end{aligned}$$

when $a_{21} = c_2 \varphi_{1,2}$

$$\Leftrightarrow R_2 = -c_2^2 h^2 \varphi_{2,2} N' - c_2^3 h^3 \varphi_{3,2} N'' + O(h^4)$$

We will now introduce the grid representation to ease calculation with these increasingly large terms. The first two terms of the residual R_2 can be written down in the grid format

R_2	1	h	h^2	h^3
N				
N'			$-c_2^2 h^2 \varphi_{2,2} N'$	
N''				$-c_2^3 h^3 \varphi_{3,2} N''$
N'''				

(2.1.26)

This tabular grid format will serve to make it easier to identify the relevant terms within our expressions for K_i 's and R_i 's. As such, it will be used as the primary representation for the K_i and R_i expressions.

In Taylor expanding K_2 we follow the pattern of (2.1.6) taking into account all terms of order h^3 and less. It is clear that R_2 is of order h^2 . We will see that this applies to R_3 and R_4 also. As a result we will not need to take the $R_i^2 N_{yy}$ term from K_i into account.

Using (2.1.6) we can construct a table for K_2 . The 1st column of the expansion leads to the main diagonal of the grid. The $R_2 N_y$ term from the expansion (2.1.6) contributes the $h^2 N'$ and $h^3 N'''$ entries, while the $h^3 N'$ term comes from $R_2 c_2 h N'_y$ in the expansion.

K_2	1	h	h^2	h^3
N	N			
N'		$c_2 h N'$	$-c_2^2 h^2 \varphi_{2,2} N' N_y$	$-c_2^3 h^3 \varphi_{2,2} N' N'_y$
N''			$\frac{1}{2} c_2^2 h^2 N''$	$-c_2^3 h^3 \varphi_{3,2} N'' N_y$
N'''				$\frac{1}{6} c_2^3 h^3 N'''$

(2.1.27)

Progressing in a similar vein for K_3 we will seek that

$$\begin{aligned}
y(t_n + c_3 h) + R_3 &= e^{c_3 h L} + h a_{31} K_1 + h a_{32} K_2 \\
c_3^2 h^2 \varphi_{2,3} N' + c_2^3 h^3 \varphi_{3,3} N'' + O(h^4) + R_3 &= h a_{31} K_1 + h a_{32} K_2 \\
&= h a_{31} N + \\
&\quad h a_{32} [N + c_2 h N' + -c_2^2 h^2 \varphi_{2,2} N' N_y + \dots] \\
&= h(a_{31} + a_{32})N + \\
&\quad h a_{32} [c_2 h N' + -c_2^2 h^2 \varphi_{2,2} N' N_y + \dots]
\end{aligned}$$

when $a_{31} + a_{32} = c_3 \varphi_{1,3}$,

$$\begin{aligned}
R_3 &= h a_{32} [c_2 h N' + -c_2^2 h^2 \varphi_{2,2} N' N_y + \dots] \\
&\quad - c_2^2 h^2 \varphi_{2,2} N' - c_2^3 h^3 \varphi_{3,2} N'' + O(h^4)
\end{aligned}$$

The superficially complex residual R_3 is actually just a combination of some of the $ha_{32} \times K_2$ terms minus some of the terms from (1.3.10) for $i = 3$.

We can tabulate R_3 as shown below. Here the {bracketed} entries are simplifications of the terms in the adjacent cells.

R_3	1	h	h^2	h^3
N				
N'		$\{h^2 N' [c_2 a_{32} - c_3^2 \varphi_2]\}$	$-c_3^2 h^2 \varphi_{2,3} N' + c_2 h^2 a_{32} N'$	$-c_2^2 h^3 a_{32} \varphi_{2,2} N' N'_y$
N''		$\{h^3 N'' [\frac{1}{2} c_2^2 a_{32} - c_3^3 \varphi_{3,3}]\}$		$-c_3^3 h^3 \varphi_{3,3} N'' + \frac{1}{2} c_2^2 h^3 a_{32} N''$
N'''				

(2.1.28)

As mentioned earlier, R_3 is indeed of order h^2 . This means we can proceed to tabulate K_3 in exactly the same fashion as we did for K_2 , with the exception that the $h^3 N'$ term in the table is now a combination of contributions from both the $R_3 N_y$ and $R_3 c_3 h N'_y$ terms in the expansion of K_3 .

K_3	1	h	h^2	h^3
N	N			
N'		$c_3 h N'$	$h^2 N' (c_2 a_{32} - c_3^2 \varphi_{2,3}) N_y$	$h^3 N' c_3 (c_2 a_{32} - c_3^2 \varphi_{2,3}) N'_y - c_2^2 h^3 a_{32} \varphi_{2,2} N' (N_y)^2$
N''			$\frac{1}{2} c_3^2 h^2 N''$	$h^3 N'' (\frac{1}{2} c_2^2 a_{32} - c_3^3 \varphi_{3,3}) N_y$
N'''				$\frac{1}{6} c_3^3 h^3 N'''$

(2.1.29)

For the K_4 stage, we require

$$\begin{aligned}
y(t_n + c_4 h) + R_4 &= e^{c_4 h L} + h a_{41} K_1 + h a_{42} K_2 + h a_{43} K_3 \\
&= h a_{41} N \\
&\quad + h a_{42} [N + c_2 h N' + -c_2^2 h^2 \varphi_{2,2} N' N_y + \dots] \\
&\quad + h a_{43} [N + c_3 h N' + h^2 N' (c_2 a_{32} - c_3^2 \varphi_{2,3}) N_y + \dots] \\
c_4^2 h^2 \varphi_{2,4} N' + c_4^3 h^3 \varphi_{3,4} N'' + O(h^4) + R_4 &= \\
&\quad h(a_{41} + a_{42} + a_{43})N + h a_{42} [c_2 h N' + -c_2^2 h^2 \varphi_{2,2} N' N_y + \dots] \\
&\quad + h a_{43} [c_3 h N' + h^2 N' (c_2 a_{32} - c_3^2 \varphi_{2,3}) N_y + \dots]
\end{aligned}$$

$$\begin{aligned}
&= y(t_n + c_3 h) + h a_{42} [c_2 h N' - c_2^2 h^2 \varphi_{2,2} N' N_y + \dots] \\
&\quad + h a_{43} [c_3 h N' + h^2 N' (c_2 a_{32} - c_3^2 \varphi_{2,3}) N_y + \dots] \\
&\quad - c_4^2 h^2 \varphi_{2,4} N' - c_4^3 h^3 \varphi_{3,4} N'' - \dots
\end{aligned}$$

when $a_{41} + a_{42} + a_{43} = c_3 \varphi_{1,3}$

$$\begin{aligned}
\Leftrightarrow R_4 &= h a_{42} [c_2 h N' + -c_2^2 h^2 \varphi_{2,2} N' N_y + \dots] \\
&\quad + h a_{43} [c_3 h N' + h^2 N' (c_2 a_{32} - c_3^2 \varphi_{2,3}) N_y + \dots] \\
&\quad - c_4^2 h^2 \varphi_{2,4} N' - c_4^3 h^3 \varphi_{3,4} N'' - \dots
\end{aligned}$$

For R_4 , as with R_3 , we tabulate the order h^2 and h^3 terms from $a_{42} K_2$ and $a_{43} K_3$ minus the order h^2 and h^3 terms of (1.3.10) for $i = 4$.

R_4	1	h	h^2	h^3
N			$\{h^3 N' N_y [-c_2^2 a_{42} \varphi_{2,2} + a_{43} (c_2 a_{32} - c_3^2 \varphi_{2,3})]\}$	
N'		$\{h^2 N' [\sum_{j=2}^3 c_j a_{4j} - c_4^2 \varphi_{2,4}]\}$	$-c_4^2 h^2 \varphi_{2,4} N'$ $+c_2 h^2 a_{42} N'$ $+c_3 h^2 a_{43} N'$	$h^3 a_{4,3} N' (c_2 a_{32} - c_3^2 \varphi_{2,3}) N_y$ $-c_2^2 h^3 a_{42} \varphi_{2,2} N' N_y$
N''		$\{h^3 N'' [\frac{1}{2} \sum_{j=2}^3 c_j^2 a_{4j} - c_4^3 \varphi_{3,4}]\}$		$-c_4^3 h^3 \varphi_{3,4} N''$ $+\frac{1}{2} c_2^2 h^3 a_{42} N''$ $+\frac{1}{2} c_3^2 h^3 a_{43} N''$
N'''				

(2.1.30)

Likewise we can tabulate the expression for K_4 . Again note that R_4 is of order h^2 .

K_4	1	h	h^2	h^3
N	N			
N'		$c_4 h N'$	$h^2 N' (\sum_{j=2}^3 c_j a_{4j} - c_4^2 \varphi_{2,4}) N_y$	$h^3 N' [-c_2^2 a_{42} \varphi_{2,2}$ $+a_{43} (c_2 a_{32} - c_3^2 \varphi_{2,3})] (N_y)^2$ $+h^3 N' c_4 [\sum_{j=2}^3 c_j^2 a_{4j} - c_4^2 \varphi_{2,4}] N'_y$
N''			$\frac{1}{2} c_4^2 h^2 N''$	$h^3 N'' (\frac{1}{2} \sum_{j=2}^3 c_j^2 a_{4j} - c_4^3 \varphi_{3,4}) N_y$
N'''				$\frac{1}{6} c_4^3 h^3 N'''$

(2.1.31)

Returning to the exact solution

$$y(t_{n+1}) = e^{hL} y_n + h \varphi_1 N + h^2 \varphi_2 N' + h^3 \varphi_3 N'' + h^4 \varphi_4 N''' + \dots \quad (2.1.32)$$

we will try to determine the conditions necessary to attain a scheme of 4th order.

$$\begin{aligned}
y_{n+1} &= e^{hL} y_n + h(b_1 K_1 + b_2 K_2 + b_3 K_3 + b_4 K_4) \\
&= e^{hL} y_n + h b_1 N + \\
&\quad h b_2 (N + c_2 h N' - c_2^2 h^2 \varphi_{2,2} N' N_y - c_2^3 h^3 \varphi_{2,2} N' N'_y + \dots) + \\
&\quad h b_3 (N + c_3 h N' + h^2 N' [c_2 a_{32} - c_3^2 \varphi_{2,3}] N_y + \\
&\quad h^3 N' \{c_3 [c_2 a_{32} - c_3^2 \varphi_{2,3}] N'_y + \dots\} + \dots) + \\
&\quad h b_4 \left(N + c_4 h N' + h^2 N' \left[\sum_{j=2}^3 c_j a_{4j} - c_4^2 \varphi_{2,4} \right] N_y + \dots \right)
\end{aligned} \tag{2.1.33}$$

$$\begin{aligned}
y_{n+1} &= e^{hL} y_n + h N (b_1 + b_2 + b_3 + b_4) + \\
&\quad h^2 N' (b_2 c_2 + b_3 c_3 + b_4 c_4 + N_y \{-c_2^2 \varphi_{2,2} + [c_2 a_{32} - c_3^2 \varphi_{2,3}] + \dots\} + \dots) + \\
&\quad h^3 N' \left(N_y \left\{ b_3 [c_2 a_{32} - c_3^2 \varphi_{2,3}] + b_4 \left[\sum_{j=2}^3 c_j a_{4j} - c_4^2 \varphi_{2,4} \right] + \dots \right\} + \right. \\
&\quad \left. N'_y \{-c_2^3 \varphi_{2,2}\} + \dots \right) + \\
&\quad \dots
\end{aligned} \tag{2.1.34}$$

Due to the complexity of these expressions, we will return to the grid forms for the K_i 's and proceed to collect expressions from terms that occur in the same grid cells.

The Conditions

We can recover the familiar consistency conditions (2.1.9a) and (2.1.9b) by requiring the cells $(1, N) = 0$. The (h, N') cells recover condition (2.1.23).

The (h^2, N') cells give

$$\begin{aligned}
0 &= b_3 c_2 a_{32} + b_4 \sum_{j=2}^3 c_j a_{4j} - b_2 c_2^2 \varphi_{2,2} - b_3 c_3^2 \varphi_{2,3} + b_4 c_4^2 \varphi_{2,4} \\
&\Rightarrow \text{condition (2.1.24a)}
\end{aligned} \tag{2.1.35}$$

While from the (h^2, N'') cells we recover condition (2.1.24b).

From the (h^3, N') cells.

$$\begin{aligned}
0 &= b_4 [-c_2^2 a_{42} \varphi_{2,2} + a_{43} (c_2 a_{32} - c_3^2 \varphi_{2,3})] - b_3 c_2^2 a_{32} \varphi_{2,2} \\
&\Rightarrow \text{condition (2.1.25a)}
\end{aligned} \tag{2.1.36}$$

The (h^3, N') cells.

$$0 = b_2 c_2^3 \varphi_{1,1} + b_3 c_3 (c_2 a_{32} - c_3^2 \varphi_{2,3}) + b_4 c_4 \left(\sum_{j=2}^3 c_j^2 a_{4j} - c_4^2 \varphi_{2,4} \right) \quad (2.1.37)$$

\Rightarrow condition (2.1.25b)

The (h^3, N'') cells.

$$0 = b_2 c_2^3 \varphi_{3,2} + b_3 \left(\frac{1}{2} c_2^2 a_{32} - c_3^3 \varphi_{3,3} \right) + b_4 \left(\frac{1}{2} \sum_{j=2}^3 c_j^2 a_{4j} - c_4^3 \varphi_{3,4} \right) \quad (2.1.38)$$

\Rightarrow condition (2.1.25c)

Finally, the (h^3, N''') cells give us condition (2.1.25d). \square

With these conditions it is possible to derive strongly 3rd and weakly 4th Order 4-stage schemes. Note that no strongly 4th Order 4-stage scheme is possible.

We can compare these conditions with those presented by Hochbruck & Ostermann [25, Equations (5.14a)-(5.14i)] and see that they are equivalent.

Weakly 3rd Order 3-Stage Schemes

For 3-stage ERKs, conditions (2.1.24a) and (2.1.24b) are impossible to satisfy simultaneously, when the earlier conditions are satisfied. Here, we can see how it is advantageous to introduce the concept of weakly satisfying an order condition.

Hochbruck & Ostermann presented a number of weakly 3rd order 3-stage schemes. As an example the following tableau represents one such scheme [25, Equation (5.8)],

0				1
c_2	$c_2 \varphi_{1,2}$			$e^{c_2 h L}$
$\frac{2}{3}$	$\frac{2}{3} \varphi_{1,3} - \frac{4}{9c_2} \varphi_{2,3}$	$\frac{4}{9c_2} \varphi_{2,3}$		$e^{\frac{2}{3} h L}$
	$\varphi_1 - \frac{3}{2} \varphi_2$	0	$\frac{3}{2} \varphi_2$	

(2.1.39)

This scheme only satisfies condition (2.1.24b) in a weakened form

$$b_2(0)c_2^2 + b_3(0)c_3^2 = 2\varphi_3(0) = \frac{1}{3}, \quad (2.1.40)$$

yet it still displays superior performance to the earlier strong 2nd schemes. Alternatively, one

can derive a weakly 3rd scheme using a weakened form of (2.1.24a) to

$$\begin{aligned}
b_3 c_2 a_{32} &= b_3 c_3^2 \varphi_{2,3} + b_2 c_2^2 \varphi_{2,2} \\
&= (b_2(0) c_2^2 + b_3(0) c_3^2) \varphi_2(0) \\
&= 2\varphi_3(0) \varphi_2(0) \\
&= \frac{1}{6}
\end{aligned} \tag{2.1.41}$$

Strongly 3rd Order 4-Stage Schemes

$$\begin{array}{c|ccc}
0 & & & \\
\frac{1}{3} & \frac{1}{3}\varphi_{1,2} & & \\
\frac{2}{3} & \frac{2}{3}\varphi_{1,3} - \frac{4}{3}\varphi_{2,3} & \frac{4}{3}\varphi_{2,3} & \\
1 & \varphi_{1,4} - \varphi_{2,4} & -\varphi_{2,4} & -\varphi_{2,4} \\
\hline
& \varphi_1 - \frac{5}{2}\varphi_2 + 3\varphi_3 & 0 & \frac{9}{2}\varphi_2 - 9\varphi_3 \quad -2\varphi_2 + 6\varphi_3
\end{array} \tag{2.1.42}$$

Scheme (2.1.42) is a 4-stage scheme with fixed c_i coefficients, it requires a total of 8 distinct φ -functions.

Order 2	$L \neq 0$	$L = 0$
2,1	5.55×10^{-17}	
2,2	3.47×10^{-17}	
2,3	3.58×10^{-18}	
Order 3		
3,4	2.78×10^{-17}	0
3,5	2.61×10^{-18}	0
Order 4		
4,6	0.00182	0.0278
4,7	0.00138	0.00926
4,8	0.00225	0
4,9	1.88×10^{-18}	0

Table 2.1: Computational inspection of Scheme 2.1.42

Table 2.1 shows computational test of the scheme coefficients against the order conditions for a sample $L \neq 0$ and $L = 0$. The test is performed by explicitly evaluating the tableau entries of the scheme and subjecting them to the order conditions. For non-zero L we used a discretisation matrix from a standard three-point finite differences discretisation.

If a scheme satisfies a condition, the corresponding value in the table should be 0. This approach clearly demonstrates that Scheme (2.1.42) is strongly third order, but it fails to satisfy

most of the fourth order conditions, even weakly.

$$\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2}\varphi_{1,2} & & & \\
\frac{2}{3} & \frac{2}{3}\varphi_{1,3} - \frac{8}{9}\varphi_{2,3} & \frac{8}{9}\varphi_{2,3} & & \\
1 & \varphi_{1,4} - \varphi_{2,4} & -2\varphi_{2,4} & 3\varphi_{2,4} & \\
\hline
& \varphi_1 - \frac{5}{2}\varphi_2 + 3\varphi_3 & 0 & -\frac{9}{2}\varphi_2 - 9\varphi_3 & -2\varphi_2 + 6\varphi_3
\end{array} \tag{2.1.43}$$

Like Scheme (2.1.42), Scheme (2.1.43) is a 4-stage scheme with fixed c_i coefficients again requiring a total of 8 distinct φ -functions. Table 2.2 demonstrates that the method is strongly third order but it fails to satisfy all but the final 4th order condition.

Order 2	$L \neq 0$	$L = 0$
2,1	5.55×10^{-17}	
2,2	3.47×10^{-17}	
2,3	2.95×10^{-18}	
Order 3		
3,4	2.78×10^{-17}	0
3,5	1.34×10^{-18}	0
Order 4		
4,6	0.00182	0.0278
4,7	0.000349	0.00463
4,8	0.00374	0
4,9	1.34×10^{-18}	0

Table 2.2: Computational inspection of Scheme (2.1.43)

Parametrised 3rd Order Families

It is also possible to maintain some of the c_i coefficients as free parameters, as in the 2-stage scheme (2.1.19). If we do so we can derive families of methods. Numerical experiments can then be performed with varying parameters to identify schemes with desirable properties. Tableau (2.1.44) shows one such family of parametrised schemes.

$$\begin{array}{c|cccc}
0 & & & & \\
c_2 & c_2\varphi_{1,2} & & & \\
c_3 & c_3\varphi_{1,3} - \frac{c_3^2}{c_2}\varphi_{2,3} & \frac{c_3^2}{c_2}\varphi_{2,3} & & \\
1 & \varphi_{1,4} - \frac{1}{c_2}\varphi_{2,4} & \frac{1}{c_2}\varphi_{2,4} & 0 & \\
\hline
& \varphi_1 - \frac{c_3+1}{c_3}\varphi_2 + \frac{2}{c_3}\varphi_3 & 0 & -\frac{1}{c_3^2-c_3}\varphi_2 + \frac{2}{c_3^2-c_3}\varphi_3 & \frac{c_3}{c_3-1}\varphi_2 + \frac{2}{c_3-1}\varphi_3
\end{array} \tag{2.1.44}$$

The following tableau describes another family of strongly 3rd order schemes with two free parameters,

$$\begin{array}{c|cccc}
0 & & & & \\
c_2 & c_2 \varphi_{1,2} & & & \\
c_3 & c_3 \varphi_{1,3} - \frac{c_3^2}{c_2} \varphi_{2,3} & \frac{c_3^2}{c_2} \varphi_{2,3} & & \\
c_2 & c_2 \varphi_{1,4} - 2c_2 \varphi_{2,4} & 2c_2 \varphi_{2,4} & 0 & \\
\hline
& \varphi_1 - \frac{c_3+c_2}{c_2 c_3} \varphi_2 + \frac{2}{c_2 c_3} \varphi_3 & \beta_2 & -\frac{c_2}{c_3^2 - c_2 c_3} \varphi_2 + \frac{2}{c_3^2 - c_2 c_3} \varphi_3 & \beta_2
\end{array} \tag{2.1.45}$$

with

$$\beta_2 = \frac{c_3}{2c_2 c_3 - 2c_2^2} \varphi_2 - \frac{1}{c_2 c_3 - c_2^2} \varphi_3$$

Here the flexibility in the free c_i parameters allows us to achieve weak 4th order with the choices $c_2 = \frac{1}{2}$ and $c_3 = 1$. Table 2.3 shows the results of an order inspection of the scheme highlighting the weakly satisfied 4th order conditions.

$$\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} \varphi_{1,2} & & & \\
1 & 1\varphi_{1,3} - 2\varphi_{2,3} & 2\varphi_{2,3} & & \\
\frac{1}{2} & \frac{1}{2} \varphi_{1,4} - \varphi_{2,4} & \varphi_{2,4} & 0 & \\
\hline
& \varphi_1 - 3\varphi_2 + 4\varphi_3 & 2\varphi_2 - 4\varphi_3 & -\varphi_2 + 4\varphi_3 & 2\varphi_2 - 4\varphi_3
\end{array} \tag{2.1.46}$$

In Section 5.3, where we run a number of numerical experiments, we will see how different

Order 2	$L \neq 0$	$L = 0$
2,1	2.78×10^{-17}	
2,2	4.34×10^{-19}	
2,3	3.25×10^{-19}	
Order 3		
3,4	6.94×10^{-18}	0
3,5	0	0
Order 4		
4,6	0.00164	0
4,7	0.000885	5.2×10^{-18}
4,8	0.00352	0
4,9	0	0

Table 2.3: Computational inspection of Scheme (2.1.46)

choices of c_2 and c_3 produce schemes with different accuracy performance.

2.2 Exponential General Linear Methods

General Linear Methods (GLMs) are a class of multi-step, multi-stage explicit methods introduced by Butcher in [7]. They are a generalisation of multi-step Adams-Bashforth methods with the multi-stage nature of Runge-Kutta methods. The advantage they offer is that they allow one to easily construct higher-order schemes while retaining an inherent Runge-Kutta stability [8]. Stability is a key property for a method and we will study it in detail in Section 4.1.

Exponential General Linear Methods (EGLMs) are an extension of GLMs into the exponential framework. These methods contain as special cases the ETD and ERK families. Ostermann, Talhammer & Wright investigated EGLMs and presented a number of example methods. They concluded that EGLMs, like their classical counterparts, combine the ease of construction of high-order methods with the superior stability properties of ERKs [38].

For problems of type (1.1.1), a 3-stage, 3-step GLMs has the following format

$$\begin{aligned} K_1 &= f(t_n, y_n) \\ K_2 &= f(t_n + c_2 h, y_n + h[a_{21}K_1 + u_{21}y_{n-1} + u_{22}y_{n-2}]) \\ K_3 &= f(t_n + c_3 h, y_n + h[a_{31}K_1 + a_{32}K_2 + u_{31}y_{n-1} + u_{32}y_{n-2}]) \\ y_{n+1} &= y_n + h(b_1K_1 + b_2K_2 + b_3K_3 + v_1y_{n-1} + v_2y_{n-2}) \end{aligned}$$

written in tableau layout as

c	A	U	0		
c_2			a_{21}		$u_{21} \quad u_{22}$
c_3			$a_{31} \quad a_{32}$		$u_{31} \quad u_{32}$
	B	V	$b_1 \quad b_2 \quad b_3$		$v_1 \quad v_2$

Ostermann, Talhammer & Wright [38] introduced a class of explicit exponential general linear methods based on the Adams-Bashforth schemes. For given start values, y_0, y_1, \dots, y_{q-1} the internal stages K_n are defined through

$$K_i = N \left(e^{hL} y_n + h \sum_{j=1}^{i-1} a_{ij}(hL) K_j + h \sum_{j=1}^q u_{ij}(hL) N(y_{n-j}) \right) \quad (2.2.1)$$

and the numerical approximation y_{n+1} at time t_{n+1} is given by the recurrence formula.

$$y_{n+1} = e^{hL} y_n + h \sum_{i=1}^s b_i(hL) K_i + h \sum_{j=1}^q v_i(hL) N(y_{n-j}). \quad (2.2.2)$$

From this general representation, we can proceed to derive order conditions for various combinations of the number of stages and the number of previous timesteps used in a scheme.

Following the format of [38], we used the naming notation EGLM_{psq} to denote a p -order s -stage q -step method. In addition we will postfix the name with c_2 when referring to families of methods with c_2 as a free parameter.

Note that the ERKs seen previously are contained within the EGLM tableau when $U = V = 0$. As with the ERKs, we have limited our research to explicit schemes. Therefore, the A matrix in the EGLM tableau is strictly lower-triangular.

2.2.1 3-Stage 4th Order Conditions

We can see more clearly the similarities between GLMs and their exponential counterparts by looking once again at a 3-stage 2-step example. An EGLM of such type is written generally in the following format,

$$\begin{aligned}
K_1 &= N_n \\
K_2 &= N(t_n + c_2 h, e^{c_2 h L} y_n + h[a_{21} K_1 + u_{21} N_{t_{n-1}} + u_{22} N_{t_{n-2}}]) \\
K_3 &= N(t_n + c_3 h, e^{c_3 h L} y_n + h[a_{31} K_1 + a_{32} K_2 + u_{31} N_{t_{n-1}} + u_{32} N_{t_{n-2}}]) \\
y_{n+1} &= e^{hL} y_n + h(b_1 K_1 + b_2 K_2 + b_3 K_3 + v_1 N_{t_{n-1}} + v_2 N_{t_{n-2}})
\end{aligned} \tag{2.2.3}$$

and can be represented compactly in tableau form,

c	A	U	0						
			c_2	a_{21}		$e^{c_2 h L}$	u_{21}	u_{22}	
			c_3	a_{31}	a_{32}	$e^{c_3 h L}$	u_{31}	u_{32}	
				b_1	b_2	b_3	e^{hL}	v_1	v_2

As with ERKs, we must derive conditions before we can begin constructing EGLMs of particular orders. For the moment, it will suffice to summarise the 3-stage, 4th order conditions here and look at examples of specific schemes. In Theorem 12, we will look at the full derivation of the following EGLM order conditions.

First Order (Consistency)

$$\sum_{j=1}^{i-1} a_{ij} + \sum_{k=1}^q u_{ik} = c_i \varphi_{1,i} \tag{2.2.4a}$$

$$\sum_i^s b_i + \sum_{j=1}^q v_j = \varphi_1 \tag{2.2.4b}$$

Second Order

$$\sum_{i=2}^s b_i c_i - \sum_{j=1}^q j v_j = \varphi_2 \tag{2.2.5a}$$

Third Order

$$\sum_{i=2}^s b_i c_i^2 + \sum_{j=1}^q j^2 v_j = 2\varphi_3 \quad (2.2.5b)$$

$$\sum_{i=2}^s b_i \left(\sum_{j=2}^{i-1} c_j a_{ij} - \sum_{j=1}^q j u_{ij} - c_i^2 \varphi_{2,i} \right) = 0 \quad (2.2.5c)$$

Fourth Order

$$\sum_{i=2}^s b_i c_i^3 - \sum_{j=1}^q j^3 v_j = 6\varphi_4 \quad (2.2.5d)$$

$$\sum_{i=2}^s b_i \left(\sum_{j=2}^{i-1} c_j^2 \frac{1}{2} a_{ij} + \sum_{j=1}^q \frac{j^2}{2} u_{ij} - c_i^3 \varphi_{3,i} \right) = 0 \quad (2.2.5e)$$

$$\sum_{j=2}^{i-1} c_j a_{ij} - \sum_{j=1}^q j u_{ij} - c_i^2 \varphi_{2,i} = 0 \quad (2.2.5f)$$

2.2.2 Examples

We will show two example EGLM schemes, a 3rd order scheme followed by a 4th order one.

3rd Order

This family of schemes is referred to as EGLM_{322c₂}

$$\begin{array}{c|cc|cc} c_2 & a_{2,1} & & e^{c_2 h L} & u_{2,1} \\ \hline & b_1 & b_2 & e^{h L} & v_1 \end{array} \quad (2.2.6)$$

$$a_{2,1} = c_2 \varphi_{1,2} + c_2^2 \varphi_{2,2}$$

$$b_1 = \varphi_1 + \frac{c_2 - 1}{c_2} \varphi_2 + \frac{-2}{c_2} \varphi_3$$

$$u_{2,1} = -c_2^2 \varphi_{2,2}$$

$$b_2 = \frac{1}{c_2^2 + c_2} \varphi_2 + \frac{2}{c_2^2 + c_2} \varphi_3$$

$$v_1 = \frac{-c_2}{c_2 + 1} \varphi_2 - \frac{2}{c_2 + 1} \varphi_3$$

It has 2 matrix exponentials and 6 distinct φ 's and as a consequence will require at least 8 matrix-vector products to implement.

Ostermann, Thalhammer & Wright made reference to the particular case of (2.2.6) where $c_2 = 1$ [38]. We refer to this scheme as EGLM₃₂₂ [38, Table 4.1]

$$\begin{array}{c|cc|cc} 1 & \varphi_1 + \varphi_2 & & -\varphi_2 & \\ \hline & \varphi_1 - 2\varphi_3 & \frac{1}{2}\varphi_2 + \varphi_3 & -\frac{1}{2}\varphi_2 - \varphi_3 & \end{array} \quad (2.2.7)$$

Here, the number of matrix exponentials has been reduced to just 1, and the number of distinct φ 's to 3. In the implementation of the scheme, the number of matrix vector products is reduced to just 5 in comparison to the more general case when $c_2 \neq 1$, where 8 matrix vector products are necessary.

4th Order

This family of schemes is referred to as EGLM₄₂₃ c_2

$$\begin{array}{c|cc|cc} c_2 & a_{21} & & e^{c_2 h L} & u_{21} & u_{22} \\ \hline & b_1 & b_2 & e^{h L} & v_1 & v_2 \end{array} \quad (2.2.8)$$

$$\begin{aligned} a_{2,1} &= c_2 \varphi_{1,2} + \frac{3c_2^2}{2} \varphi_{2,2} + c_2^3 \varphi_{3,2} \\ u_{2,1} &= -2c_2^2 \varphi_{2,2} - 2c_2^3 \varphi_{3,2} & u_{2,2} &= \frac{c_2^2}{2} \varphi_{2,2} + c_2^3 \varphi_{3,2} \\ b_1 &= \varphi_1 + \frac{\frac{3c_2-2}{2} \varphi_2 + c_2 - 3\varphi_3 - 3\varphi_4}{c_2} & b_2 &= \frac{2\varphi_2 + 6\varphi_3 + 6\varphi_4}{c_2^3 + 3c_2^2 + 2c_2} \\ v_1 &= \frac{-2c_2 \varphi_2 - 2c_2 - 4\varphi_3 + 6\varphi_4}{c_2 + 1} & v_2 &= \frac{\frac{c_2}{2} \varphi_2 + c_2 - 1\varphi_3 - 3\varphi_4}{c_2 + 2} \end{aligned}$$

This scheme has 2 matrix exponentials and 8 distinct φ 's.

As with (2.2.7), Ostermann, Thalhammer & Wright consider the choice of fixing $c_2 = 1$ to help minimise the number of distinct φ -functions evaluations needed, and gives us the published scheme, EGLM₄₂₃ [38, Table 4.2] .

$$\begin{array}{c|cc|cc} 1 & \varphi_1 + \frac{3}{2}\varphi_2 + \varphi_3 & & -2\varphi_2 - 2\varphi_3 & \frac{1}{2}\varphi_2 + \varphi_3 \\ \hline & \varphi_1 + \frac{1}{2}\varphi_2 - 2\varphi_3 - 3\varphi_4 & \frac{1}{3}\varphi_2 + \varphi_3 + \varphi_4 & -\varphi_2 + \varphi_3 + 3\varphi_4 & \frac{1}{6}\varphi_2 - \varphi_4 \end{array} \quad (2.2.9)$$

The number of matrix exponentials is again 1, and the number of distinct φ 's is 4. As before we see a significant reduction in the number of matrix-vector products necessary within the implementation. Specifically here we see that number reduced to just 7. All of these schemes require two function evaluations per time step.

Chapter 3

Multi-value Families of EIs

In this chapter we,

- Introduce two new families of EIs whose construction is based on classical Almost Runge-Kutta Methods (ARKs) introduced by Butcher in [9]. We name these new families,
 - Exponential Almost Runge-Kutta Methods (EARKs)
 - Exponential Almost General Linear Methods (EAGLMs)

We will introduce the classical ARK family and show how the concepts behind its construction can be extended into the EI framework.

- Derive order conditions for these new families and present some example schemes.
- Provide a convergence analysis of EAGLMs within a framework of abstract semilinear parabolic evolution equations.

3.1 Exponential Almost Runge-Kutta Methods

ARKs were introduced by Butcher in 1997 [9]. They are a special case of GLMs and they retain the multi-stage nature of RKs, but allow for the passing of more than one value from step to step. In a sense they are multi-value schemes rather than multi-step schemes. For an ARKs scheme, three values form the inputs and outputs at each step. They are, the approximation to the solution, which is accompanied by approximations to the first and second derivatives. The general form of an ARK is,

$$\begin{pmatrix} Y_1 \\ \vdots \\ Y_S \\ \hline y_{n+1} \\ hy'_{n+1} \\ h^2y''_{n+1} \end{pmatrix} = \left(\begin{array}{c|c} A & W \\ \hline B & Z \end{array} \right) \begin{pmatrix} hf(Y_1) \\ \vdots \\ hf(Y_S) \\ \hline y_n \\ hy'_n \\ h^2y''_n \end{pmatrix} \quad (3.1.1)$$

where the Y_i 's are the stage approximations and y_{n+1} , hy'_{n+1} and $h^2y''_{n+1}$ and the output approximations for the next step. The general form of the tableau is,

$$\begin{array}{c|c} & \begin{matrix} a_{21} \\ \vdots \\ a_{s-1,1} & a_{s-1,3} & \ddots \end{matrix} \\ & \begin{matrix} \vdots & \ddots & \end{matrix} \end{array} \left| \begin{matrix} e & c - Ae & \frac{c^2}{2} - Ac \end{matrix} \right. \\ \hline \begin{array}{c|c} A & U \\ \hline B & V \end{array} = & \begin{array}{cccc|ccc} b_1 & b_2 & \cdots & b_{s-1} & & & \\ b_1 & b_2 & \cdots & b_{s-1} & 0 & 1 & b_0 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 0 & 0 & 0 \\ \beta_1 & \beta_2 & \cdots & \beta_{s-1} & \beta_s & 0 & \beta_0 & 0 \end{array} \quad (3.1.2)$$

In ARKs, the approximation to the second derivative need only be of order h^3 because the method imposes special “annihilation conditions” to ensure this lower order does not adversely affect the solution. A distinct advantage of ARKs is that they have a stage order of 2, as opposed to RKs which have a stage order of at most 1. This higher stage order allows one to interpolate or obtain an error estimate cheaply [40].

Within EIs, EARKs become a special case of EGLMs. The input and output value passed from step-to-step are the function evaluations of the approximate solution, $N(y_{n+1})$, again followed by increasing derivatives, $N_n^{(i)}$. Here, however, we see an important difference with EARKs namely, the values passed are derivatives of the function N rather than the solution y . That is, we pass the values $N^{(i)}(y_{n+1})$ on to the next step.

With traditional ARKs, we arrange that the final internal stage gives us the same value as

that generated to be the first outgoing approximation, y_{n+1} . This means that, in ARKs, the first row of the B matrix is the same as the last row of the A matrix, as can be seen in (3.1.2). Similarly, the first row of the Z matrix is the same as that last row of the W matrix.

EARKs exhibit the same characteristic as ARKs. However, EARKs differ in that the first outgoing approximation passed to the next step is not y_{n+1} , but the function N evaluated at y_{n+1} , usually written N_{n+1} . Still, the first B and Z matrix rows will be identical to the last A and W matrix rows when written in the tableau format. When implementing the scheme, the y_{n+1} stage result used to approximate the N_{n+1} output, is simply reused to avoid a redundant calculation.

A significant difference between ARKs and EARKs is in the second row of the B and Z matrices. In traditional ARKs, the second output value, $y' \equiv f(y)$, and is already available to pass to the next step. As such, the second row of the B and Z matrices contain all zeros except for a 1 in the $(2, s)$ position, where s is the number of stages. In EARKs both the second and third output values will be non-trivial.

The general form of an EARK _{p s r} tableau is,

$$\begin{array}{c|ccc|ccc}
 c_2 & a_{21} & & & & w_{21} & \cdots & w_{2r} \\
 \vdots & \vdots & \ddots & & & \vdots & & \vdots \\
 c_{s-1} & a_{s-1,1} & \cdots & a_{s-1,s-2} & & w_{s-1,1} & \cdots & w_{s-1,r} \\
 1 & b_1 & \cdots & b_{s-1} & 0 & z_1 & \cdots & z_r \\
 \hline
 & b_1 & \cdots & b_{s-1} & 0 & z_1 & \cdots & z_r \\
 & \beta_{11} & \cdots & \beta_{1,s-1} & \beta_{1s} & \delta_{11} & \cdots & \delta_{1r} \\
 & \vdots & & \vdots & \vdots & \vdots & & \vdots \\
 & \beta_{r1} & \cdots & \beta_{r,s-1} & \beta_{rs} & \delta_{r1} & \cdots & \delta_{rr}
 \end{array} \tag{3.1.3}$$

where, using a similar notation to EGLMs, the subscripts denote a p -order s -stage r -value method.

We have concentrated on 3-stage EARKs, where $s = 3$ and $r = 1$ or 2 . The repetition in the final row of the tableaux upper half, and the first row of the lower half means that, s -stage EARKs share more in common with $(s-1)$ -stage EGLMs. The simplest form EARK is a 3-stage, 1-value method. The tableau for such a method would look like the following

$$\begin{array}{c|cc|c}
 c_2 & a_{21} & & w_{21} \\
 1 & b_1 & b_2 & z_1 \\
 \hline
 & b_1 & b_2 & 0 & z_1 \\
 & \beta_{21} & \beta_{22} & \beta_{23} & \delta_{21}
 \end{array} \tag{3.1.4}$$

written out explicitly, such a method takes the following form,

$$\begin{aligned}
K_1 &= N_n \\
Y_2 &= e^{c_2 h L} y_n + h [a_{21} K_1 + w_{21} h N'] \\
K_2 &= N(t_n + c_2 h, Y_2) \\
Y_3 &= e^{h L} y_n + h (b_1 K_1 + b_2 K_2 + z_1 h N') \\
K_3 &= N(t_n + c_3 h, Y_3) \\
y_{n+1} &= Y_3 \\
h N'_{n+1} &= \beta_{21} K_1 + \beta_{22} K_2 + \beta_{23} K_3 + \delta_{21} h N'_n
\end{aligned} \tag{3.1.5}$$

3.1.1 Deriving EARKs

In deriving EARKs the input derivatives of the N function on a given step match those of the exact solution (1.3.9) and as such can be used directly to estimate the next step. Consequently, the w_{2i} entries in the EARKs tableau must be $c_2^{i+1} \varphi_{i+1}$. This will then generate an intermediate step of order $p - 1$, where p is the overall order of the scheme.

Using Taylor expansions of the approximations generated by the internal stages, together with the input values for that step, it is possible to derive the coefficients to produce the necessary output values, N'_{n+1} and N''_{n+1} . Because EARKs do not pass derivatives of the solution y to the next step but rather pass derivatives of the N function, we do not need to involve the φ functions in generating those outputs. Consequently, the second and third rows of the B and Z matrices contain only scalar entries. Here EARKs show their potential to achieve high orders, while avoiding too many matrix-vector products.

3.1.2 3-Stage 4th Order Conditions

As with EGLMs, we will first present the order conditions for EARKs of the form (3.1.3) and then introduce example schemes. Theorem 13 will provide a full proof of these conditions.

First Order (Consistency) The consistency conditions of EARKs are identical to those of ERKs (2.1.9). This is because in the case $N_n = N$ a constant, $N^{(i)} = 0$ for $i > 0$.

Second Order

$$\sum_{i=2}^s b_i c_i + z_1 = \varphi_2 \tag{3.1.6}$$

Third Order

$$\sum_{i=2}^s b_i c_i^2 + 2z_2 = 2\varphi_3 \quad (3.1.7)$$

$$\sum_{i=2}^s b_i \left(\sum_{j=2}^{i-1} c_j a_{ij} + w_{i1} - c_i^2 \varphi_{2,i} \right) = 0 \quad (3.1.8)$$

Fourth Order

$$\sum_{i=2}^s b_i c_i^3 = 6\varphi_4 \quad (3.1.9)$$

$$\sum_{i=2}^s b_i \left(\sum_{j=2}^{i-1} c_j^2 \frac{1}{2} a_{ij} + w_{i2} - c_i^3 \varphi_{3,i} \right) = 0 \quad (3.1.10)$$

$$\sum_{j=2}^{i-1} c_j a_{ij} + w_{i1} - c_i^2 \varphi_{2,i} = 0, i = 2, \dots, s \quad (3.1.11)$$

3.1.3 Outgoing Approximations

To produce the vector of outgoing approximations, $v = (N'_{n+1}, \dots, N_{n+1}^{(r)})$, we construct a matrix $M = (\beta \delta)$, where β is $r \times s$ and δ is $r \times r$, such that,

$$M \begin{pmatrix} N_n \\ N_{n+c_2} \\ \vdots \\ N_{n+c_s} \\ N_{n+1} \\ hN'_n \\ \vdots \\ h^P N_n^{(r)} \end{pmatrix} = \begin{pmatrix} hN'_{n+1} \\ \vdots \\ h^P N_{n+1}^{(r)} \end{pmatrix} \quad (3.1.12)$$

This matrix is constructed by solving a number of linear systems generated from Taylor expansions of the elements of incoming vector of approximations, together with the N_{n+c_i} and N_{n+1} values produced by the internal stages. Specifically, m_j , the j^{th} row of the matrix M , is the solution to the linear system

$$X m_j^T = e_{j+1} \quad (3.1.13)$$

where e_i is the standard basis vector and the $r+s \times r+s$ matrix $X = (Y \ Z)$ is the matrix formed

by the matrices Y and Z arranged side-by-side where,

$$Y = \begin{pmatrix} 1 & c_2 & \cdots & c_i & \cdots & c_{s-1} & 1 \\ -1 & -c_2 & \cdots & -c_i & \cdots & c_{s-1} & 0 \\ \frac{1}{2} & \frac{1}{2}c_2 & \cdots & \frac{1}{2}c_i & \cdots & \frac{1}{2}c_{s-1} & 0 \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ -1^{j-1}\frac{1}{(j-1)!} & & \cdots & -1^{j-1}\frac{1}{(j-1)!}c_i & \cdots & & 0 \\ \vdots & & & \vdots & & & \vdots \end{pmatrix} \quad (3.1.14)$$

$$Z = \begin{pmatrix} 0 & 0 & & 0 \\ 1 & 0 & & \\ -1 & 1 & \ddots & \vdots \\ \frac{1}{2} & -1 & \ddots & \\ \vdots & \frac{1}{2} & \ddots & 0 \\ & \vdots & \ddots & 1 \\ & & \frac{-1^{j-r-1}}{(j-r-1)!} & \vdots \end{pmatrix} \quad (3.1.15)$$

Looking at just the 3-stage, 2-value case, to construct the β_j , δ_j rows for a scheme of the form (3.1.4), one has to solve the linear system

$$\begin{pmatrix} 1 & c & 1 & 0 & 0 \\ -1 & -c & 0 & 1 & 0 \\ \frac{1}{2} & \frac{c}{2} & 0 & -1 & 1 \\ -\frac{1}{6} & -\frac{c}{6} & 0 & \frac{1}{2} & -1 \\ \frac{1}{4!} & \frac{c}{4!} & 0 & -\frac{1}{6} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} \beta_{j1} \\ \beta_{j2} \\ \beta_{j3} \\ \delta_{j1} \\ \delta_{j2} \end{pmatrix} = e_j \quad (3.1.16)$$

3.1.4 Examples

We are now in a position to construct families of EARKs. Looking at a 4th order method, we note that it is not necessary to utilise the N''_n incoming approximation to produce approximations to N'_{n+1} and N''_{n+1} of sufficient order. As such, the δ_{j2} tableau entries can be zero. EARK₄₃₂ c_2 is a 3-stage, 2-value family of 4th order schemes with one free parameter, c_2 .

c_2	$c_2\varphi_{1,2}$		$c_2^2\varphi_{2,2}$	$c_2^3\varphi_{3,2}$		
1	$\varphi_1 - \frac{6}{c_2^3}\varphi_4$	$\frac{6}{c_2^3}\varphi_4$	0	$\varphi_2 - \frac{6}{c_2^2}\varphi_4$	$\varphi_3 - \frac{3}{c_2}\varphi_4$	
	$\varphi_1 - \frac{6}{c_2^3}\varphi_4$	$\frac{6}{c_2^3}\varphi_4$	0	$\varphi_2 - \frac{6}{c_2^2}\varphi_4$	$\varphi_3 - \frac{3}{c_2}\varphi_4$	
	$-\frac{2c_2^2-3c_2}{c_2^3-2c_2+1}$	$-\frac{1}{c_2^3-2c_2^2+c_2}$	$\frac{2c_2+1}{c_2}$	$-\frac{c_2}{c_2-1}$	0	
	$\frac{2c_2^2-6}{c_2^3-2c_2+1}$	$-\frac{4}{c_2^3-2c_2^2+c_2}$	$\frac{2c_2+4}{c_2}$	$-\frac{2c_2+2}{c_2-1}$	0	

(3.1.17)

3.2 Exponential Almost General Linear Methods

As our goal is to have a consistent representation of all our schemes, we use an expanded tableau which can unambiguously contain all method families so far mentioned. In addition, the representation points towards a broader family of methods, which we refer to as EAGLMs. These combine the multi-stage, multi-step nature of EGLMs with the multi-value format of EARKs. To that end, the following tableau is capable of representing all 3-stage schemes presented so far

$$\begin{array}{c|c|c|c} c & A & U & W \\ \hline & B & V & Z \end{array} \quad \begin{array}{c|cccc|cccc|cc} c_2 & a_{21} & & & & e^{c_2 h L} & u_{21} & u_{22} & w_{21} & w_{22} \\ c_3 & a_{31} & a_{32} & & & e^{c_3 h L} & u_{31} & u_{32} & w_{31} & w_{32} \\ 1 & b_1 & b_2 & b_3 & 0 & e^{h L} & v_1 & v_2 & z_1 & z_2 \\ \hline & b_1 & b_2 & b_3 & 0 & e^{h L} & v_1 & v_2 & z_1 & z_2 \\ & \beta_{21} & \beta_{22} & \beta_{23} & \beta_{24} & 0 & \gamma_{21} & \gamma_{22} & \delta_{21} & \delta_{22} \\ & \beta_{31} & \beta_{32} & \beta_{33} & \beta_{34} & 0 & \gamma_{31} & \gamma_{32} & \delta_{31} & \delta_{32} \end{array} \quad (3.2.1)$$

$$K_1 = N_n$$

$$K_2 = N \left(t_n + c_2 h, e^{c_2 h L} y_n + h [a_{21} K_1 + u_{21} N_{t_{n-1}} + u_{22} N_{t_{n-2}} + w_{21} h N' + w_{22} h^2 N''] \right)$$

$$K_3 = N \left(t_n + c_3 h, e^{c_3 h L} y_n + \right.$$

$$\left. h [a_{31} K_1 + a_{32} K_2 + u_{31} N_{t_{n-1}} + u_{32} N_{t_{n-2}} + w_{31} h N' + w_{32} h^2 N''] \right)$$

$$y_{n+1} = e^{h L} y_n + h (b_1 K_1 + b_2 K_2 + b_3 K_3 + v_1 N_{t_{n-1}} + v_2 N_{t_{n-2}} + z_1 h N' + z_2 h^2 N'')$$

(3.2.2)

We can see that the earlier families of methods all become special cases of EAGLMs. For example when representing EGLMs in (3.2.1), the W and Z matrices contain all zeros and likewise the β , γ and δ entries are all zero. For pure EARKs on the other hand, which makes no use of previous steps, the U and V are zero matrices, and the γ entries are zero.

3.2.1 3-Stage 6th Order Conditions

The final extension to order conditions, is to look at the combined family of EAGLMs. We will present a full proof for the derivation of 3-stage, q -step, r -value EAGLM conditions. We will then use that result to prove the conditions presented earlier for EGLMs and EARKs in Section 2.2.1 and Section 3.1.2.

Theorem 3. *EAGLMs of the form*

0											
c_2	a_{21}				$e^{c_2 h L}$	u_{21}	\cdots	u_{2q}	w_{21}	\cdots	w_{2r}
\vdots	\vdots	\ddots			\vdots		\vdots			\vdots	
c_s	a_{s1}	\cdots	$a_{s(s-1)}$		$e^{c_s h L}$	u_{s1}	\cdots	u_{sq}	w_{s1}	\cdots	w_{sr}
	b_1	\cdots	\cdots	b_s	$e^{h L}$	v_1	\cdots	v_q	z_1	\cdots	z_r

$$K_1 = N_n = N(t_n, y_n)$$

$$Y_i = e^{hL} y_n + h \sum_{j=1}^{i-1} a_{ij}(hL) K_j + h \sum_{k=1}^q u_{ik}(hL) N_{n-k} + h \sum_{j=1}^r w_{ij} h^j N^{(j)} \quad (3.2.3)$$

$$K_i = N(Y_i)$$

$$y_{n+1} = e^{hL} y_n + h \left(\sum_{i=1}^s b_i(hL) K_i + \sum_{k=1}^q v_k(hL) N_{n-k} + \sum_{j=1}^r z_j(hL) h^j N^{(j)} \right)$$

will be exact in the case $N_n = N$, a constant, if the following conditions are met

$$\sum_{j=1}^{i-1} a_{ij} + \sum_{j=1}^q u_{ij} = c_i \varphi_{1,i} \quad (3.2.4a)$$

for $i = 1, \dots, s$

$$\sum_i^s b_i + \sum_{i=1}^q v_i = \varphi_1 \quad (3.2.4b)$$

These conditions are referred to as the consistency conditions.

Proof. From the stage approximations with $N(t_n, y_n) = N$, a constant, we obtain:

If

$$y(t_n + c_i h) = Y_i$$

then

$$\begin{aligned} e^{c_i h L} y_n + c_i h \varphi_{1,i} N &= e^{c_i h L} y_n + h \sum_{j=1}^{i-1} a_{ij}(hL) K_j + h \sum_{k=1}^q u_{ij}(hL) N \\ &= e^{c_i h L} y_n + h \left(\sum_{j=1}^{i-1} a_{ij}(hL) + \sum_{k=1}^q u_{ij}(hL) \right) N \\ \Rightarrow \sum_{j=1}^{i-1} a_{ij} + \sum_{j=1}^q u_{ij} &= c_i \varphi_{1,i} \end{aligned}$$

recovering condition (3.2.4a). By a similar approach, the outgoing step approximation, if

$$y(t_n + h) = y_{n+1}$$

then

$$\begin{aligned}
e^{hL}y_n + h\varphi_1N &= e^{hL}y_n + h \sum_{i=1}^s b_i K_i + h \sum_{k=1}^q v_i N \\
&= e^{hL}y_n + h \left(\sum_{i=1}^s b_i + \sum_{k=1}^q v_i \right) N \\
&\Rightarrow \sum_{i=1}^s b_i + \sum_{i=1}^q v_i = \varphi_1
\end{aligned}$$

and we recover condition (3.2.4b). \square

Theorem 4. *EAGLMs of the form (3.2.3), where $s = 2$ or 3 , that is 2 or 3-stage methods, that meet the consistency conditions (3.2.4), can achieve order p if*

$$\sum_{k=2}^{j-1} a_{jk} \frac{c_k^i}{i!} + \sum_{k=1}^q \frac{(-k)^i}{i!} u_{kj} + w_{ki} = c_k^{i+1} \varphi_{i+1,k} \quad (3.2.5a)$$

for $i = 1, \dots, p-3$, $j = 1, \dots, s$,

$$\sum_{j=2}^s b_j \left(\sum_{k=2}^{j-1} a_{jk} \frac{c_k^i}{i!} + \sum_{k=1}^q \frac{(-k)^i}{i!} u_{jk} + w_{ji} - c_j^{i+1} \varphi_{i+1,j} \right) = 0 \quad (3.2.5b)$$

where $i = p-2$, and

$$\sum_{j=2}^s \frac{c_j^i}{i!} b_j + \sum_{j=1}^q \frac{(-j)^i}{i!} v_j + z_i = \varphi_{i+1} \quad (3.2.5c)$$

for $i = 1, \dots, p-1$

Lemma 5. *The 2^{nd} stage approximation to the solution at $y(t_n + c_2h)$, at the intermediate point $t_n + c_2h$*

$$e^{c_2hL}y_n + h \left(a_{21}K_1 + \sum_{k=1}^q u_{2k}N_{n-k} + \sum_{j=1}^r w_{2j}h^j N_n^{(j)} \right)$$

differs from the exact solution, $y(t_n + c_2h)$, by $\sum_{i=1} \mathcal{R}_{2,i}^ h^{i+1}$ with*

$$\mathcal{R}_{j,i}^* = \left(\sum_{k=2}^{j-1} a_{jk} \frac{c_k^i}{i!} + \sum_{k=1}^q \frac{(-k)^i}{i!} u_{jk} + w_{ji} - c_j^{i+1} \varphi_{i+1,j} \right) N_n^{(i)} \quad (3.2.6)$$

where $w_{2i} = 0$ for $i > r$

Proof.

$$y(t_n + c_2h) = e^{c_2hL}y_n + c_2h\varphi_{1,2}N + c_2^2h^2\varphi_{2,2}N' + \dots$$

given condition (3.2.4a) we can determine the format of R_2

$$\begin{aligned}
& e^{c_2 h L} y_n + h \left[a_{21} K_1 + \sum_{k=1}^q u_{2k} N_{n-k} + \sum_{j=1}^r w_{2j} h^j N_n^{(j)} \right] \\
&= y(t_n + c_2 h) - c_2^2 h^2 \varphi_{2,2} N'_n - c_2^3 h^3 \varphi_{3,2} N''_n - \dots \\
&\quad - u_{21} h^2 N'_n + \frac{1}{2!} u_{21} h^3 N''_n - \dots \\
&\quad + \dots \\
&\quad - u_{2q} h^2 N'_n + \frac{q^2}{2!} u_{2q} h^3 N''_n - \dots \\
&\quad + w_{21} h^2 N'_n + \dots + w_{2r} h^{r+1} N_n^{(r)} \\
&= y(t_n + c_2 h) - \sum_{i=1} c_2^{i+1} \varphi_{i+1,2} h^{i+1} N_n^{(i)} \\
&\quad + \sum_{i=1}^q \sum_{j=1} \frac{(-j)^i}{i!} u_{2j} h^{i+1} N_n^{(i)} \\
&\quad + \sum_{i=1} w_{2i} h^{i+1} N_n^{(i)} \\
&= y(t_n + c_2 h) + \sum_{i=1} \left(\sum_{j=1}^q \frac{(-j)^i}{i!} u_{2j} + w_{2i} - c_2^{i+1} \varphi_{i+1,2} \right) h^{i+1} N_n^{(i)}
\end{aligned}$$

Hence

$$\begin{aligned}
e^{c_2 h L} y_n + h \left[a_{21} K_1 + \sum_{k=1}^q u_{2k} N_{n-k} + \sum_{j=1}^r w_{2j} h^j N_n^{(j)} \right] &= y(t_n + c_2 h) + \sum_{i=1} \mathcal{R}_{2,i}^* h^{i+1} \\
&= y(t_n + c_2 h) + R_2
\end{aligned} \tag{3.2.7}$$

□

The 2nd residual R_2 is $O(h^2)$. We also write

$$R_2^2 = \sum_{i=1} \mathcal{R}_{2,i}^{**} h^{i+3}$$

where

$$\mathcal{R}_{k,i}^{**} = \sum_{j=1}^i \mathcal{R}_{k,j}^* \mathcal{R}_{k,i-j+1}^* N^{(j)} N^{(i-j+1)} \tag{3.2.8}$$

Higher powers of R_2 will be $O(h^6)$ or greater and will not be needed.

Lemma 6. When $\mathcal{R}_{2,i}^* = 0$ for $i = 1, \dots, p-1$ the 2nd stage approximation is order p .

Proof. This follows simply from the fact that for $\mathcal{R}_{2,i}^* = 0$ for $i = 1, \dots, p-1$, then

$$y(t_n + c_2 h) + \sum_{i=1}^{p-1} \mathcal{R}_{2,i}^* h^{i+1} = y(t_n + c_2 h) + O(h^{p+1})$$

□

Lemma 7. *The 2nd stage approximation*

$$K_2 = N \left(t_n + c_2 h, e^{c_2 h L} y_n + h \left[a_{21} K_1 + \sum_{k=1}^q u_{2k} N_{n-k} + \sum_{j=1}^r w_{2j} h^j N_n^{(j)} \right] \right)$$

can be expressed as

$$K_2 = \sum_{i=0}^{p-1} \frac{c_2^i}{i!} N_n^{(i)} h^i + \sum_{i=0}^{p-3} \mathcal{K}_{2,i}^* h^{i+2} + \sum_{i=0}^{p-5} \mathcal{K}_{2,i}^{**} h^{i+4} + O(h^p) \quad (3.2.9)$$

where

$$\mathcal{K}_{j,i}^* = \sum_{k=0}^i \frac{c_j^{i-k}}{(i-k)!} \mathcal{R}_{j,k+1}^* N_y^{(i-k)} \quad (3.2.10)$$

$$\mathcal{K}_{j,i}^{**} = \sum_{k=0}^i \frac{c_j^{i-k}}{(i-k)!} \mathcal{R}_{j,k+1}^{**} N_{yy}^{(i-k)} \quad (3.2.11)$$

Proof. From Theorem 5, $K_2 = N(t_n + c_2 h, y(t_n + c_2 h) + R_2)$. Then, using (2.1.6), we have

$$K_2 = \sum_{i=0}^{p-3} \frac{c_2^i}{i!} \left(\sum_{j=1}^2 R_2^j N_y^{(i)} \right) h^i$$

Since R_2^2 is $O(h^6)$ we need only consider $j < 3$, as such

$$K_2 = \sum_{i=0}^{p-1} \frac{c_2^i}{i!} N_n^{(i)} h^i + \sum_{i=0}^{p-3} \frac{c_2^i}{i!} R_2 N_y^{(i)} h^i + \sum_{i=0}^{p-5} \frac{c_2^i}{i!} R_2^2 N_{yy}^{(i)} h^i + O(h^5)$$

focusing on the final $j = 1$ and 2 terms we see

$$\begin{aligned} & \sum_{i=0}^{p-3} \frac{c_2^i}{i!} R_2 N_u^{(i)} h^i + \sum_{i=0}^{p-5} \frac{c_2^i}{i!} R_2^2 N_{uu}^{(i)} h^i \\ &= \sum_{i=0}^{p-3} \frac{c_2^i}{i!} \left(\sum_{j=1}^{p-2-i} \mathcal{R}_{2,j+1}^* h^{j+2} \right) N_u^{(i)} h^i + \sum_{i=0}^{p-5} \frac{c_2^i}{i!} \left(\sum_{j=0}^{p-4-i} \mathcal{R}_{2,j+1}^{**} h^{j+4} \right) N_{uu}^{(i)} h^i \\ &= \sum_{i=0}^{p-3} \left(\sum_{j=0}^i \frac{c_2^{i-j}}{(i-j)!} \mathcal{R}_{2,j+1}^* N_y^{(i-j)} \right) h^{i+2} + \sum_{i=0}^{p-5} \left(\sum_{j=0}^i \frac{c_2^{i-j}}{(i-j)!} \mathcal{R}_{2,j+1}^{**} N_{yy}^{(i-j)} \right) h^{i+4} \\ &= \sum_{i=0}^{p-3} \mathcal{K}_{2,i}^* h^{i+2} + \sum_{i=0}^{p-5} \mathcal{K}_{2,i}^{**} h^{i+4} \end{aligned}$$

□

Lemma 8. *The 3rd stage approximation at the intermediate point $t_n + c_3 h$,*

$$e^{c_3 h L} y_n + h \left(a_{31} K_1 + a_{32} K_2 + \sum_{i=1}^q u_{3i} N_{t_n-i} + \sum_{j=1}^r w_{3j} h^j N^{(j)} \right)$$

differs from the exact solution, $y(t_n + c_3h)$ by

$$R_3 = \sum_{i=1}^{p-2} \mathcal{R}_{3,i}^* h^{i+1} + a_{32} \sum_{i=0}^{p-4} \mathcal{K}_{2,i}^* h^{i+3} + a_{32} \sum_{i=0}^{p-6} \mathcal{K}_{2,i}^{**} h^{i+5} \quad (3.2.12)$$

with $\mathcal{R}_{3,i}^*$, $\mathcal{K}_{j,i}^*$ and $\mathcal{K}_{j,i}^{**}$ as in (3.2.6), (3.2.10) and (3.2.11) respectively.

Proof. The prerequisite condition (3.2.4a) implies that

$$a_{31}K_1 + a_{32}K_2 = c_3h\varphi_{1,3} + \sum_{i=1}^{p-2} \frac{c_2^i}{i!} N_n^{(i)} h^i + \sum_{i=0}^{p-4} \mathcal{K}_{2,i}^* h^{i+2} + \sum_{i=0}^{p-6} \mathcal{K}_{2,i}^{**} h^{i+4}$$

Hence the 3rd stage approximation to $y(t_n + c_3h)$ may be written as follows

$$\begin{aligned} e^{c_3hL}y_n + h \left[a_{31}K_1 + a_{32}K_2 + \sum_{k=1}^q u_{3k}N_{n-k} + \sum_{j=1}^r w_{3j}h^j N_n^{(j)} \right] \\ = y(t_n + c_2h) - \sum_{i=1}^{p-2} \left(\sum_{j=1}^q \frac{(-j)^i}{i!} u_{3j} + w_{3i} - c_3^{i+1} \varphi_{i+1,3} \right) h^{i+1} N_n^{(i)} \\ + \sum_{i=1}^{p-2} \frac{c_2^i}{i!} N_n^{(i)} h^i + \sum_{i=0} \mathcal{K}_{2,i}^* h^{i+2} + \sum_{i=0} \mathcal{K}_{2,i}^{**} h^{i+4} \\ = y(t_n + c_2h) + R_3 \end{aligned}$$

□

When considering R_3^2 we note that

$$R_3 = \sum_{i=1}^2 \mathcal{R}_{3,i}^* h^{i+1} + a_{32} \mathcal{R}_{2,1}^* h^3 N_y + O(h^4)$$

so that

$$R_3^2 = \sum_{i=1}^2 \mathcal{R}_{3,i}^{**} h^{i+3} + a_{32} \mathcal{R}_{2,1}^* \mathcal{R}_{3,1}^* h^5 N_y + O(h^6) \quad (3.2.13)$$

with $\mathcal{R}_{3,i}^{**}$ as in (3.2.8).

Lemma 9. If $\mathcal{R}_{2,i}^* = \mathcal{R}_{3,i}^* = 0$ for $i = 1, \dots, p-1$ then the 3rd stage approximation is order p .

Proof. Given $\mathcal{R}_{2,i}^* = \mathcal{R}_{3,i}^* = 0$ for $i = 1, \dots, p-1$ then

$$\begin{aligned} y(t_n + c_3h) + R_3 &= y(t_n + c_3h) + \sum_{i=1}^{p-1} \mathcal{R}_{3,i}^* h^{i+1} + a_{32} \sum_{i=0}^{p-3} \mathcal{K}_{2,i}^* h^{i+3} + a_{32} \sum_{i=0}^{p-5} \mathcal{K}_{2,i}^{**} h^{i+5} \\ &= y(t_n + c_3h) + O(h^{p+1}) + O(h^{p+2}) + O(h^{2p+3}) \\ &= y(t_n + c_3h) + O(h^{p+1}) \end{aligned}$$

□

Lemma 10. *The 3rd stage outgoing approximation*

$$K_3 = N \left(t_n + c_3 h, e^{c_3 h L} y_n + h \left[a_{31} K_1 + a_{32} K_2 + \sum_{k=1}^q u_{3k} N_{n-k} + \sum_{j=1}^r w_{3j} h^j N_n^{(j)} \right] \right)$$

can be expressed as

$$K_3 = \sum_{i=0}^{p-1} \frac{c_3^i}{i!} N_n^{(i)} h^i + \sum_{i=0}^{p-3} \mathcal{K}_{3,i}^* h^{i+2} + \sum_{i=0}^{p-4} \mathcal{K}_{3,i}^{**} h^{i+4} + \sum_{i=0}^{p-3} \mathcal{L}_{3,i}^* h^{i+3} + a_{32} \left((\mathcal{R}_{2,1}^*)^2 + \mathcal{R}_{2,1}^* \mathcal{R}_{3,1}^* \right) h^5 (N')^2 N_{yy} N_y + O(h^p) \quad (3.2.14)$$

with $\mathcal{K}_{3,i}^*$ and $\mathcal{K}_{3,i}^{**}$ as in (3.2.10) and (3.2.11) respectively, and where

$$\mathcal{L}_{3,i}^* = a_{32} \frac{c_3^i}{i!} \left(\sum_{j=0}^i \left[\sum_{l=0}^{i-j} \frac{c_2^{j-l}}{(j-l)!} \mathcal{R}_{2,j+1}^* N_y^{(i-l)} N_y^{(l)} \right] \right)$$

Proof. Following the format of (3.2.1), we can write

$$K_3 = \sum_{i=0}^{p-1} \frac{c_3^i}{i!} N_n^{(i)} h^i + \sum_{i=0}^{p-3} \frac{c_3^i}{i!} R_3 N_y^{(i)} h^i + \sum_{i=0}^{p-5} \frac{c_3^i}{i!} R_3^2 N_{yy}^{(i)} h^i + O(h^p)$$

Using (3.2.12) we obtain

$$\sum_{i=0}^{p-3} \frac{c_3^i}{i!} R_3 N_y^{(i)} h^i = \sum_{i=0}^{p-3} \frac{c_3^i}{i!} \left(\sum_{j=1}^{p-2-i} \mathcal{R}_{3,i}^* h^{j+1} + a_{32} \sum_{j=0}^{p-4-i} \mathcal{K}_{2,j}^* h^{j+3} + a_{32} \sum_{j=0}^{p-6-i} \mathcal{K}_{2,j}^{**} h^{j+5} \right) N_y^{(i)} h^i$$

The order of the summations of the terms involving $\mathcal{K}_{2,j}^*$ can be rearranged such that

$$\begin{aligned} \sum_{i=0}^{p-3} a_{32} \frac{c_3^i}{i!} \left(\sum_{j=0}^{p-4-i} \mathcal{K}_{2,j}^* h^{j+3} \right) N_y^{(i)} h^i &= \sum_{i=0}^{p-3} a_{32} \frac{c_3^i}{i!} \left(\sum_{j=0}^{p-4-i} \left[\sum_{l=0}^j \frac{c_2^{j-l}}{(j-l)!} \mathcal{R}_{2,l+1}^* N_y^{(j-l)} \right] h^{j+3} \right) N_y^{(i)} h^i \\ &= \sum_{i=0}^{p-3} a_{32} \frac{c_3^i}{i!} \left(\sum_{j=0}^i \left[\sum_{l=0}^j \frac{c_2^{j-l}}{(j-l)!} \mathcal{R}_{2,l+1}^* N_y^{(j-l)} \right] N_y^{(i-j)} \right) h^{i+3} \\ &= \sum_{i=0}^{p-3} a_{32} \frac{c_3^i}{i!} \left(\sum_{j=0}^i \left[\sum_{l=0}^{i-j} \frac{c_2^{i-j-l}}{(i-j-l)!} \mathcal{R}_{2,j+1}^* N_y^{(i-l)} N_y^{(l)} \right] \right) h^{i+3} \end{aligned}$$

For the terms involving $\mathcal{K}_{2,j}^{**}$, only one is below order $O(h^6)$

$$\sum_{i=0}^{p-3} \frac{c_3^i}{i!} a_{32} \left(\sum_{j=0}^{p-6-i} \mathcal{K}_{2,j}^{**} h^{j+5} \right) N_y^{(i)} h^i = a_{32} (\mathcal{R}_{2,1}^*)^2 (N'_n)^2 h^5 N_{yy} N_y + O(h^6)$$

In addition, by (3.2.13)

$$\begin{aligned} \sum_{i=0}^{p-3} \frac{c_3^i}{i!} R_3^2 N_{yy}^{(i)} h^i &= \sum_{i=0}^{p-3} \frac{c_3^i}{i!} \left(\sum_{j=0}^{p-6-i} \mathcal{R}_{3,j}^{**} h^{j+3} + \mathcal{R}_{2,1}^* \mathcal{R}_{3,1}^* h^5 N_y \right) N_y^{(i)} h^i \\ &= \sum_{i=0}^{p-3} \left(\sum_{j=0}^i \frac{c_3^{i-j}}{(i-j)!} \mathcal{R}_{3,j+1}^{**} N_{yy}^{(i-j)} \right) h^{i+4} + a_{32} \mathcal{R}_{2,1}^* \mathcal{R}_{3,1}^* h^{(i+5)} N_y N_{yy} \end{aligned}$$

Collecting $\mathcal{R}_{j,i}^*$'s and $\mathcal{R}_{j,i}^{**}$'s we obtain (3.2.14). □

Lemma 11. *If $\mathcal{R}_{2,i}^* = \mathcal{R}_{3,i}^* = 0$ for $i = 1, \dots, p-3$ then the outgoing approximation,*

$$y_{n+1} = e^{hL} y_n + h \left(b_1 K_1 + b_2 K_2 + b_3 K_3 + \sum_{k=1}^q v_k N_{n-k} + \sum_{j=1}^r z_j h^j N_n^{(j)} \right)$$

differs from the exact solution, $y(t_n + h)$, by

$$\sum_{i=1}^{p-1} \left(\sum_{j=1}^2 \frac{c_j^i}{i!} b_j + \sum_{j=1}^q \frac{(-j)^i}{i!} v_j + z_i - \varphi_{i+1} \right) N_n^{(i)} h^{i+1} + \sum_{j=1}^2 b_j \mathcal{R}_{j,p-1}^* N_y h^p + O(h^{p+1}) \quad (3.2.15)$$

Proof. The conditions $\mathcal{R}_{2,i}^* = \mathcal{R}_{3,i}^* = 0$ for $i = 1, \dots, p-3$ implies

$$\begin{aligned} K_2 &= \sum_{i=0}^{p-1} \frac{c_2^i}{i!} N_n^{(i)} h^i + \sum_{i=0}^{p-3} \mathcal{K}_{2,i}^* h^{i+2} + \sum_{i=0}^{p-5} \mathcal{K}_{2,i}^{**} h^{i+4} + O(h^6) \\ &= \sum_{i=0}^{p-1} \frac{c_2^i}{i!} N_n^{(i)} h^i + \mathcal{K}_{2,p-3}^* h^{p-1} + O(h^p) \\ &= \sum_{i=0}^{p-1} \frac{c_2^i}{i!} N_n^{(i)} h^i + \mathcal{R}_{2,p-2}^* N_y h^{p-1} + O(h^p) \end{aligned}$$

and

$$\begin{aligned} K_3 &= \sum_{i=0}^{p-1} \frac{c_3^i}{i!} N_n^{(i)} h^i + \sum_{i=0}^{p-3} \mathcal{K}_{3,i}^* h^{i+2} + \sum_{i=0}^{p-5} \mathcal{K}_{3,i}^{**} h^{i+4} \\ &\quad + \sum_{i=0}^{p-4} \mathcal{L}_{3,i}^* h^{i+3} + a_{32} \left((\mathcal{R}_{2,1}^*)^2 + \mathcal{R}_{2,1}^* \mathcal{R}_{3,1}^* \right) h^5 (N_n')^2 N_{yy} N_y + O(h^6) \\ &= \sum_{i=0}^{p-1} \frac{c_3^i}{i!} N_n^{(i)} h^i + \mathcal{K}_{3,p-3}^* h^{p-1} + O(h^p) \\ &= \sum_{i=0}^{p-1} \frac{c_3^i}{i!} N_n^{(i)} h^i + \mathcal{R}_{3,p-2}^* N_y h^{p-1} + O(h^p) \end{aligned}$$

also, given the consistency condition (3.2.4b),

$$e^{hL} y_n + h \left(b_1 K_1 + b_2 K_2 + b_3 K_3 + \sum_{k=1}^q v_k N_{n-k} + \sum_{j=1}^r z_j h^j N_n^{(j)} \right)$$

$$\begin{aligned}
&= y(t_n + h) - h^2 \varphi_2 N'_n - h^3 \varphi_3 N''_n - \dots \\
&\quad - v_1 h^2 N'_n + \frac{1}{2!} v_1 h^3 N''_n - \dots \\
&\quad + \dots \\
&\quad - v_q h^2 N'_n + \frac{q^2}{2!} v_q h^3 N''_n - \dots \\
&\quad + z_1 h^2 N'_n + \dots + z_r h^{r+1} N_n^{(r)} \\
&\quad + b_1 \sum_{i=0}^{p-1} \frac{c_2^i}{i!} N_n^{(i)} h^{i+1} + b_1 \mathcal{R}_{2,w-2}^* h^r \\
&\quad + b_2 \sum_{i=0}^{p-1} \frac{c_3^i}{i!} N_n^{(i)} h^{i+1} + b_2 \mathcal{R}_{3,p-2}^* N_y h^p + O(h^{p+1}) \\
&= y(t_n + h) - \sum_{i=1}^{p-1} \varphi_{(i+1)} h^{i+1} N_n^{(i)} \\
&\quad + \sum_{i=1}^{p-1} \sum_{j=1}^q \frac{(-j)^i}{i!} v_j h^{i+1} N_n^{(i)} \\
&\quad + \sum_{i=1}^{p-1} z_i h^{i+1} N_n^{(i)} \\
&\quad + \sum_{i=1}^{p-1} \sum_{j=1}^2 \frac{c_j^i}{i!} b_j h^{i+1} N_n^{(i)} \\
&\quad + \sum_{j=2}^3 \mathcal{R}_{j,p-2}^* N_y h^p + O(h^{p+1}) \\
&= y(t_n + h) + \sum_{i=1}^{p-1} \mathcal{S}_i^* h^{i+1} + \sum_{j=2}^3 \mathcal{R}_{j,p-2}^* N_y h^p + O(h^{p+1})
\end{aligned}$$

with

$$\mathcal{S}_i^* = \left(\sum_{j=1}^2 \frac{c_j^i}{i!} b_j + \sum_{j=1}^q \frac{(-j)^i}{i!} v_j + z_i - \varphi_{i+1} \right) N_n^{(i)} \quad (3.2.16)$$

□

Proof of Theorem 19. Within each $\mathcal{L}_{3,i}^*$ the terms $\frac{c_2^i}{(i)!} \mathcal{R}_{2,i+1}^* N_y^{(i)} N_y h^{i+3}$ occur in isolation, therefore, to achieve order p implies that $\mathcal{R}_{2,i}^* = 0$ for $i = 1, \dots, p-3$. From the $\mathcal{K}_{j,i}^*$ terms, this restriction on $\mathcal{R}_{2,i}^*$ also implies $\mathcal{R}_{3,i}^* = 0$. These two results recover the order condition (3.2.5a).

From this we can apply Lemma 11. To ensure the approximation

$$y(t_n + h) + \sum_{i=1}^{p-1} \left(\sum_{j=1}^2 \frac{c_j^i}{i!} b_j + \sum_{j=1}^q \frac{(-j)^i}{i!} v_j + z_i - \varphi_{i+1} \right) N_n^{(i)} h^{i+1} + \sum_{j=1}^2 b_j \mathcal{R}_{j,p-1}^* N_y h^p$$

is of order p then we require

$$\sum_{j=1}^2 \frac{c_j^i}{i!} b_j + \sum_{j=1}^q \frac{(-j)^i}{i!} v_j + z_i - \varphi_{i+1} = 0$$

for $i = 1, \dots, p-2$, recovering order condition (3.2.5b) and

$$\sum_{j=1}^2 b_j \mathcal{R}_{j,p-1}^* = 0$$

which recovers condition (3.2.5c). \square

Note. When considering 3-stage methods where $c_2 \neq c_3$, the condition (3.2.5b) becomes merged with condition (3.2.5a) for the extended set of values $i = 1, \dots, p-2$. A consequence of this can be seen from Propositions 6 and 9; all such methods will have stage order $p-1$ when the method is of order p .

Theorem 12. EGLMs of the form

0						
c_2	a_{21}				$e^{c_2 h L}$	$u_{21} \quad \cdots \quad u_{2q}$
\vdots	\vdots	\ddots			\vdots	\vdots
c_s	a_{s1}	\cdots	$a_{s(s-1)}$		$e^{c_s h L}$	$u_{s1} \quad \cdots \quad u_{sq}$
	b_1	\cdots	\cdots	b_s	e^{hL}	$v_1 \quad \cdots \quad v_q$

$$K_1 = N_n$$

$$K_i = N \left(e^{hL} y_n + h \sum_{j=1}^{i-1} a_{ij}(hL) K_j + h \sum_{k=1}^q u_{ik}(hL) N_{n-k} \right) \quad (3.2.17)$$

$$y_{n+1} = e^{hL} y_n + h \left(h \sum_{i=1}^s b_i(hL) K_i + \sum_{k=1}^q v_k N_{n-k} \right)$$

will be exact in the case $N_n = N$, a constant, if the consistency conditions (3.2.4) are met.

When N is non-constant, EGLMs of the form (3.2.17) where $s = 2$ or 3 , can achieve order p if they satisfy (3.2.4) and

$$\sum_{k=2}^{j-1} a_{jk} \frac{c_k^i}{i!} + \sum_{k=1}^q \frac{(-k)^i}{i!} u_{kj} = c_j^{i+1} \varphi_{i+1,k} \quad (3.2.18a)$$

for $i = 1, \dots, p-3$, $j = 1, \dots, s$,

$$\sum_{j=2}^s b_j \left(\sum_{k=2}^{j-1} a_{jk} \frac{c_k^i}{i!} + \sum_{k=1}^q \frac{(-k)^i}{i!} u_{jk} - c_j^{i+1} \varphi_{i+1,j} \right) = 0 \quad (3.2.18b)$$

for $i = p-2$, and

$$\sum_{j=2}^s \frac{c_j^i}{i!} b_j + \sum_{j=1}^q \frac{(-j)^i}{i!} v_j = \varphi_{i+1} \quad (3.2.18c)$$

for $i = 1, \dots, p-1$

Proof. This follows naturally from Theorem 3 and 19 where all $w_{ij} = z_{ij} = 0$. □

In the 2-stage, case these conditions become

$$\sum_{k=2}^{j-1} a_{jk} \frac{c_k^i}{i!} + \sum_{k=1}^q \frac{(-k)^i}{i!} u_{kj} = c_k^{i+1} \varphi_{i+1,k}$$

for $i = 1, \dots, p-2$; $j = 1, 2$ and, for $i = 1, \dots, p-1$,

$$\sum_{j=2}^s \frac{c_j^i}{i!} b_j + \sum_{j=1}^q \frac{(-j)^i}{i!} v_j = \varphi_{(i+1)}$$

These match the same conditions as those derived by Ostermann, Thalhaammer & Wright [38, (2.3) & (2.7)].

Theorem 13. *EARKs of the form*

0			
c_2	a_{21}		$e^{c_2 h L}$
\vdots	\vdots	\ddots	\vdots
c_s	a_{s1}	\cdots	$e^{c_s h L}$
	$a_{s(s-1)}$		w_{s1}
	b_1	\cdots	z_1
	b_s		z_r

$$K_1 = N_n$$

$$K_i = N \left(e^{hL} y_n + h \sum_{j=1}^{i-1} a_{ij}(hL) K_j + h \sum_{l=1}^r w_{il} h^l N^{(l)} \right) \quad (3.2.19)$$

$$y_{n+1} = e^{hL} y_n + h \left(h \sum_{i=1}^s b_i(hL) K_i + \sum_{l=1}^r z_l h^l N^{(l)} \right)$$

will be exact in the case $N_n = N$, a constant, if the following conditions are met

$$\sum_{j=1}^{i-1} a_{ij} = c_i \varphi_{1,i} \quad (3.2.20a)$$

for $i = 1, \dots, s$

$$\sum_i b_i = \varphi_1 \quad (3.2.20b)$$

These conditions are the same as the ERKs consistency conditions.

In the case where N_n is non-constant, EARKs of the form (3.2.19) where $s = 2$ or 3 , can achieve order p if they satisfy (3.2.20) and

$$\sum_{k=2}^{j-1} a_{jk} \frac{c_k^i}{i!} + w_{ki} = c_k^{i+1} \varphi_{i+1,k} \quad (3.2.21a)$$

for $i = 1, \dots, p-3, j = 1, \dots, s,$

$$\sum_{j=2}^s b_j \left(\sum_{k=2}^{j-1} a_{jk} \frac{c_k^i}{i!} + w_{ji} - c_j^{i+1} \varphi_{i+1,j} \right) = 0 \quad (3.2.21b)$$

for $i = p-2$, and

$$\sum_{j=2}^s \frac{c_j^i}{i!} b_j + z_i = \varphi_{i+1} \quad (3.2.21c)$$

for $i = 1, \dots, p-1$

Proof. This follows naturally from Theorem 3 and 19 where all $u_{ij} = v_{ij} = 0$. \square

3.2.2 Outgoing Approximations

Like EARKs, the format for deriving the β_j, γ_j and δ_j rows of a method follows from constructing a matrix $M = (\beta \quad \gamma \quad \delta)$ such that

$$M \begin{pmatrix} N_n \\ N_{n+c_2} \\ \vdots \\ N_{n+c_s} \\ N_{n+1} \\ N_{n-1} \\ \vdots \\ N_{n-q} \\ hN'_n \\ \vdots \\ h^r N_n^{(r)} \end{pmatrix} = \begin{pmatrix} hN'_{n+1} \\ \vdots \\ h^r N_{n+1}^{(r)} \end{pmatrix} \quad (3.2.22)$$

As earlier, the m_j rows of the matrix M , are the solutions to the linear system

$$X m_j^T = e_{j+1} \quad (3.2.23)$$

with X being the $(s+q+r) \times (s+q+r)$ matrix

$$X = (Y \quad Z \quad W) \quad (3.2.24)$$

where A and Z are as in (3.1.14) and (3.1.15) respectively and

$$W = \begin{pmatrix} 1 & \cdots & 1 \\ -1 & \cdots & -1 \\ \frac{1}{2} & \cdots & \frac{1}{2} \\ \vdots & & \vdots \\ \frac{-1^{j-1}}{(j-1)!} & \cdots & \frac{-1^{j-1}}{(j-1)!} \\ \vdots & & \vdots \end{pmatrix} \quad (3.2.25)$$

If we look at the EARK scheme (3.1.17) for the case where $c_2 = 1$, one loses the ability to generate approximations of sufficiently high order since the two output values N_{n+c_2} and N_{n+1} are equal. With the two internal stages giving an approximation at the same point in time, there are no longer enough points of information on which to perform the interpolation. The solution is to carry along additional information at each step. The obvious choice for providing this information are previous steps and this naturally leads us to favour these hybrid EAGLMs over pure EARKs.

In practice we shall see that the optimal approach is to restrict both the internal stages and output step calculations to a pure EARKs format. This means, not involving any previous steps in their computation, i.e. $u_{ij} = 0$. This preserves the reduction in φ matrix-vector products exhibited earlier. By using non-zero δ_{ij} values we have the option to include involve the previous steps, in the approximations of N'_{n+1} and N''_{n+1} , where there are no matrix-vector products.

An example of such a scheme is the 3rd order EAGLM₃₃₃₁ c_2 ,

$$\begin{array}{c|cc|cc|c} c_2 & a_{2,1} & & u_{2,1} & u_{2,2} & w_{2,1} & c_2 \\ \hline 1 & b_1 & b_2 & v_{2,1} & v_{2,2} & z_1 & 1 \\ \hline & b_1 & b_2 & 0 & v_{2,1} & v_{2,2} & z_1 \end{array} = \begin{array}{c|cc|cc|c} c_2 & c_2\varphi_{1,2} & & 0 & 0 & c_2^2\varphi_{2,2} \\ \hline 1 & \varphi_1 - \frac{2\varphi_3}{c_2^2} & \frac{2\varphi_3}{c_2^2} & 0 & 0 & \varphi_2 - \frac{2\varphi_3}{c_2} \\ \hline & \varphi_1 - \frac{2\varphi_3}{c_2^2} & \frac{2\varphi_3}{c_2^2} & 0 & 0 & \varphi_2 - \frac{2\varphi_3}{c_2} \end{array} \quad (3.2.26)$$

$$\overline{\beta \mid \gamma \mid \delta} = \left(\begin{array}{ccc|cc} -3 & 0 & \frac{11}{6} & \frac{3}{2} & -\frac{1}{3} \\ -5 & 0 & 2 & 4 & -1 \end{array} \mid 0 \right) \times \begin{pmatrix} N_n \\ N_{t_c} \\ N_{n+1} \\ N_{n-1} \\ N_{n-2} \\ hN'_n \end{pmatrix} = \begin{pmatrix} hN'_{n+1} \\ h^2N''_{n+1} \end{pmatrix}$$

3.2.3 Example

In our research, we have concentrated on 2-stage schemes. Of course, technically, when speaking about EARKs and EAGLMs, we mean 3-stage schemes, because of the repeated B, V, Q

row. For such 2-stage schemes

$$\begin{array}{c|cc|cc|cc} c_2 & a_{21} & & e^{c_2 h L} & u_{21} & u_{22} & w_{21} & w_{22} \\ \hline & b_1 & b_2 & e^{h L} & v_1 & v_2 & z_1 & z_2 \end{array} \quad (3.2.27)$$

$$K_1 = N$$

$$K_2 = N (t_n + c_2 h, e^{c_2 h L} y_n + h [a_{21} K_1 + u_{21} N_{t_{n-1}} + u_{22} N_{t_{n-2}} + w_{21} h N' + w_{22} h^2 N''])$$

we can simplify conditions (3.2.5) to

$$\sum_{k=2}^{j-1} a_{jk} \frac{c_k^i}{i!} + \sum_{k=1}^q \frac{(-k)^i}{i!} u_{kj} + w_{ki} = c_k^{i+1} \varphi_{k,i+1} \quad (3.2.28)$$

for $i = 1, \dots, \beta - 2, j = 1, \dots, s$, and

$$\sum_{j=2}^s \frac{c_j^i}{i!} b_j + \sum_{j=1}^q \frac{(-j)^i}{i!} v_j + z_i = \varphi_{i+1} \quad (3.2.29)$$

for $i = 1, \dots, \beta - 1$.

We will now present some examples of such schemes. We have adopted the convention whereby we call a scheme an EAGLM only if it has non-zero u_{ij} or v_i entries in it's tableau. A scheme where $u_{ij} = v_i = 0$ is called an EARK even if it has non-zero γ_{ij} entries. In other words, an EARK which uses previous steps to generate the N derivatives, is still referred to as an EARK, and not an EAGLM.

3rd Order EARK₃₂₁ c_2

$$\begin{array}{c|cc|cc|c} c_2 & a_{2,1} & & u_{2,1} & u_{2,2} & w_{2,1} \\ \hline 1 & b_1 & b_2 & v_{2,1} & v_{2,2} & z_1 \end{array} = \begin{array}{c|cc|cc|cc} c_2 & c_2 \varphi_{1,2} & & 0 & 0 & c_2^2 \varphi_{2,2} \\ \hline 1 & \varphi_1 - \frac{2\varphi_3}{c_2^2} & \frac{2\varphi_3}{c_2^2} & 0 & 0 & \varphi_2 - \frac{2\varphi_3}{c_2} \\ \hline & \varphi_1 - \frac{2\varphi_3}{c_2^2} & \frac{2\varphi_3}{c_2^2} & 0 & 0 & \varphi_2 - \frac{2\varphi_3}{c_2} \end{array} \quad (3.2.30)$$

4th Order EARK₄₂₂ c_2

$$\begin{array}{c|cc|cc} c_2 & a_{2,1} & & w_{2,1} & w_{2,2} \\ \hline 1 & b_1 & b_2 & z_1 & z_2 \end{array} = \begin{array}{c|cc|cc} c_2 & c_2 \varphi_{1,2} & & c_2^2 \varphi_{2,2} & c_2^3 \varphi_{3,2} \\ \hline 1 & \varphi_1 - \frac{6}{c_2^3} \varphi_4 & \frac{6}{c_2^3} \varphi_4 & \varphi_2 - \frac{6}{c_2^2} \varphi_4 & \varphi_3 - \frac{3}{c_2} \varphi_4 \\ \hline & \varphi_1 - \frac{6}{c_2^3} \varphi_4 & \frac{6}{c_2^3} \varphi_4 & 0 & \varphi_2 - \frac{6}{c_2^2} \varphi_4 & \varphi_3 - \frac{3}{c_2} \varphi_4 \end{array} \quad (3.2.31)$$

4th Order EAGLM

This family of schemes, referred to as EAGLM₄₂₂₁, is 4th order 3-stage 2-step 1-value. It combines the use of one previous timestep, with the N' derivative. We can see clearly the significant

increase in the complexity of the tableau entries over scheme (3.2.30) and (3.2.31).

$$\begin{array}{c|cc|c} c_2 & a_{2,1} & \cdots & c_2 \\ \hline 1 & b_1 & b_2 & \cdots = 1 \\ \hline & b_1 & b_2 & 0 \end{array} \quad \begin{array}{c|c} c_2 \varphi_{1,2} - 2c_2^3 \varphi_{3,2} \\ \frac{c_2(6\varphi_4+2\varphi_3)-6\varphi_4-2c_2^3\varphi_3-2\varphi_3+\varphi_1c_2}{c_2^2} \quad \frac{6\varphi_4+2\varphi_3}{c_2^3+c_2^2} \\ \hline \frac{c_2(6\varphi_4+2\varphi_3)-6\varphi_4-2c_2^3\varphi_3-2\varphi_3+\varphi_1c_2}{c_2^2} \quad \frac{6\varphi_4+2\varphi_3}{c_2^3+c_2^2} \quad 0 \end{array} \quad \begin{array}{c} \cdots \\ \cdots \\ \cdots \end{array}$$

3.3 Convergence Bounds

We saw that Ostermann, Thalhammer & Wright introduced EGLMs in [38]. As part of the theory supporting their new family of methods, they provided a convergence analysis within a framework of abstract semilinear parabolic evolution equations. As part of that analysis they provide a Theorem, [38, (3.4)], which proves a convergence estimate for EGLMs.

Following the format of that Theorem, we will modify the proof and apply it to EAGLMs to show that the convergence bounds derived hold for EAGLMs.

3.3.1 Order Conditions

We use the format of the expansion of the solution of the linear differential equation from [38, Lemma 1.1] in our own convergence analysis.

Lemma 14. [38, Lemma 1.1] *The exact solution of the initial value problem*

$$y'(t) = Ly(t) + f(t), \quad t \geq t_n, \quad (3.3.1)$$

with $y(t_n)$ given, has the following representation

$$y(t_n + \tau) = e^{\tau L} y(t_n) + \sum_{l=0}^{m-1} \tau^{l+1} \varphi_{l+1}(\tau L) N^{(l)}(t_n) + R_n(m, \tau) \quad (3.3.2)$$

$$R_n(m, \tau) = \int_0^\tau e^{(\tau-\sigma)L} \int_0^\sigma \frac{(\sigma-\xi)^{m-1}}{(m-1)!} N^{(m)}(t_n + \xi) d\xi d\sigma \quad (3.3.3)$$

Proof. The proof of this follows from substituting the Taylor expansion of N

$$\begin{aligned} N(t_n + \sigma) &= \sum_{l=0}^{m-1} \frac{\sigma^l}{l!} N^{(l)}(t_n) + S_n(m, \sigma) \\ S_n(m, \sigma) &= \int_0^\sigma \frac{(\sigma - \xi)^{m-1}}{(m-1)!} N^{(m)}(t_n + \xi) d\xi \end{aligned} \quad (3.3.4)$$

into the variation of constants formula

$$y(t_n + \tau) = e^{\tau L} y(t_n) + \int_0^\tau e^{(\tau - \sigma)L} f(t_n + \sigma) d\sigma \quad (3.3.5)$$

□

Within the framework of EAGLMs, the internal stages and numerical approximation are defined by

$$\begin{aligned} Y_{ni} &= e^{c_i h L} y_n + h \sum_{j=1}^{i-1} a_{ij}(hL) N_{nj} + h \sum_{k=1}^q u_{ik}(hL) N_{n-k} + h \sum_{m=1}^r w_{im}(hL) h^m N_n^{(m)} \\ N_{ni} &= N(Y_{ni}) \end{aligned} \quad (3.3.6)$$

$$y_{n+1} = e^{hL} y_n + h \sum_{i=1}^s b_i(hL) N_{ni} + h \sum_{k=1}^q v_k(hL) N_{n-k} + h \sum_{m=1}^r z_m(hL) h^m N_n^{(m)}$$

with $q + r = \theta - 1$. $N_n^{(i)}$ is defined by a linear combination of the available values at a given step, that is

$$\begin{aligned} h^i N_n^{(i)} &= \sum_j \alpha_{ij} \mathcal{N}_{nj}, \quad \mathcal{N}_{nj} \in \{N_{n-k}, N_{(n-1)i}\}, \\ k &= 0, \dots, q, \quad i = 1, \dots, s, \quad m = 1, \dots, r \end{aligned} \quad (3.3.7)$$

and we introduce the notation

$$\Delta N_n^{(i)} = \sum_j (\alpha_{ij} \mathcal{N}_{nj}(\hat{y}_n) - \alpha_{ij} \mathcal{N}_{nj}(y_n)) \quad (3.3.8)$$

We also define

$$\begin{aligned} \mathcal{S}_{ni}(m) &= \sum_j \alpha_{ij} S_n(m, h\omega), \quad \omega \in \{-k, -1 + c_i\} \\ k &= 0, \dots, q, \quad i = 1, \dots, s \end{aligned} \quad (3.3.9)$$

We assume the starting values, y_1, \dots, y_q and $N'_S, \dots, N_S^{(r)}$ have been computed beforehand. Defining the exact solution values,

$$\hat{y}_n = t(t_n), \quad \hat{Y}_{ni} = y(t_n + c_i h), \quad (3.3.10)$$

the defects of the internal stages take the form

$$D_{ni} = \hat{Y}_{ni} - e^{c_i h L} \hat{y}_n - h \sum_{j=1}^s a_{ij} N_{n+c_i} - h \sum_{k=1}^q u_{ik} N_{n-k} - h \sum_{m=1}^r w_{im} h^m N_n^{(m)} \quad (3.3.11)$$

and those of the numerical solution are

$$d_{n+1} = \hat{y}_{n+1} - e^{hL} \hat{y}_n - h \sum_{i=1}^s b_i N_{n+c_i} - h \sum_{k=1}^q v_k N_{n-k} - h \sum_{m=1}^r z_m h^m N_n^{(m)} \quad (3.3.12)$$

Making use of the expansion of the exact solution (1.3.9) we get the following expansion for the defects of the internal stage,

$$D_{ni} = \sum_{l=1}^q h^l \Theta_{li}(hL) f^{(l-1)}(t_n) + R_{ni}^{(\theta)}, \quad (3.3.13)$$

$$\Theta_{li}(hL) = c_i^l \varphi_l(c_i hL) - \sum_{j=1}^{i-1} \frac{c_j^{l-1}}{(l-1)!} a_{ij}(hL) - \sum_{k=1}^q \frac{(-k)^{l-1}}{(l-1)!} u_{ik}(hL) - w_{li} \quad (3.3.14)$$

and for the defects of the numerical solution we get the expansion

$$d_{n+1} = \sum_{l=1}^r h^l \Psi_l(hL) f^{(l-1)}(t_n) + \varrho_{n+1}^{(\theta)}, \quad (3.3.15)$$

$$\Psi_l(hL) = \varphi_l(hL) - \sum_{j=1}^s \frac{c_j^{l-1}}{(l-1)!} b_j(hL) - \sum_{k=1}^q \frac{(-k)^{l-1}}{(l-1)!} v_k(hL) - z_l \quad (3.3.16)$$

The numerical solution will be of *stage order* θ and *quadrature order* p if $D_{ni} = O(h^{\theta+1})$ for $1 \leq i \leq s$ and $d_{n+1} = O(h^{p+1})$ [38]. Therefore we will require $\Theta_{li}(hL) = 0$ for $1 \leq i \leq s$ and $1 \leq l \leq \theta$, and $\Psi_l(hL) = 0$ for $1 \leq l \leq p$. This implies

$$\begin{aligned} c_i^l \varphi_l(c_i hL) &= \sum_{j=1}^{i-1} \frac{c_j^{l-1}}{(l-1)!} a_{ij}(hL) \\ &\quad + \sum_{k=1}^q \frac{(-k)^{l-1}}{(l-1)!} u_{ik}(hL) + w_{li}, \quad 1 \leq i \leq s \quad 1 \leq l \leq q \end{aligned} \quad (3.3.17)$$

$$\varphi_l(hL) = \sum_{j=1}^s \frac{c_j^{l-1}}{(l-1)!} b_j(hL) + \sum_{k=1}^q \frac{(-k)^{l-1}}{(l-1)!} v_k(hL) + z_l, \quad 1 \leq l \leq p. \quad (3.3.18)$$

This “recovers” the EAGLM order conditions (3.2.4) and (3.2.5), as proved in Theorem 3. We will proceed under the assumption that the scheme satisfies these order conditions. This implies that the remainders, which combine the residuals (3.3.3) and (3.3.4), are defined by

$$\begin{aligned} R_{ni}^{(\theta)} &= R_n(\theta, c_i h) - h \sum_{j=1}^{i-1} a_{ij}(hL) S_n(\theta, c_j h) \\ &\quad - h \sum_{k=1}^q u_{ik}(hL) S_n(\theta, -kh) - h \sum_{m=1}^r w_{im}(hL) h^m S_{nm}(\theta) \\ \varrho_{n+1}^{(p)} &= R_n(\theta, c_i h) - h \sum_{i=1}^S b_i(hL) S_n(p, c_i h) \\ &\quad - h \sum_{k=1}^q v_k(hL) S_n(p, -kh) - h \sum_{m=1}^r z_m(hL) h^m S_{nm}(p) \end{aligned} \quad (3.3.19)$$

The errors are defined by

$$e_n = \hat{y}_n - y_n, \quad E_{ni} = \hat{Y}_{ni} - Y_{ni}, \quad 1 \leq i \leq s \quad (3.3.20)$$

so that the stage errors take the form

$$E_{ni} = e^{c_i h L} e_n + h \sum_{j=1}^S a_{ij} \Delta N_{ni} + h \sum_{k=1}^q u_{ik} \Delta N_{n-k} + h \sum_{m=1}^r w_{im} h^m \Delta N_n^{(m)} + D_{ni} \quad (3.3.21)$$

and those of the numerical solution are

$$e_{n+1} = e^{hL} \hat{y}_n + h \sum_{i=1}^S b_i \Delta N_{ni} + h \sum_{k=1}^q v_k \Delta N_{n-k} + h \sum_{m=1}^r z_m h^m \Delta N_n^{(m)} + d_{n+1} \quad (3.3.22)$$

which gives us the recurrence for e_n

$$e_n = e^{(t_n - t_{\theta-1})L} e_{\theta-1} + \sum_{l=q}^n e^{(t_n - t_l)L} d_l + h \sum_{l=q}^{n-1} e^{(t_n - t_{l+1})L} \times \left(\sum_{j=1}^S b_j \Delta N_{li} + \sum_{k=1}^q v_k \Delta N_{l-k} + \sum_{m=1}^r z_m h^m \Delta N_l^{(m)} \right), \quad n \geq \theta - 1 \quad (3.3.23)$$

Within the analytical framework of abstract semilinear parabolic evolution equations, we will avail of the following key bounds, presented by Ostermann, Thalhammer & Wright [38, (3.5) and (3.6)]. Appendix A.1 summarises many of the definitions which we use within this analysis. See [23] for a thorough treatment of the theory of sectorial operators and analytic semigroups.

The requirement (A.1.2), together with Hypothesis 17, renders (1.3.2) a semilinear parabolic problem [38]. We will make use of the following two bounds from [38, (3.5) & (3.6)].

$$\|t^{v-\mu} e^{tL}\|_{X_v \leftarrow X_\mu} \leq C \quad (3.3.24)$$

$$\|t^{v-\mu} \varphi_l(tL)\|_{X_v \leftarrow X_\mu} \leq C \quad (3.3.25)$$

where $0 \leq t \leq T$ and $0 \leq \mu \leq v \leq 1$, with a constant $C > 0$. The bound (3.3.25) is valid of all $l \geq 1$. We also extend assumption [38, (3.11)],

$$\|a_{ij}(hL)\|_{X_v \leftarrow X_\mu} + \|b_i(hL)\|_{X_v \leftarrow X_\mu} + \|u_{ij}(hL)\|_{X_v \leftarrow X_\mu} + \|v_i(hL)\|_{X_v \leftarrow X_\mu} \leq C h^{-v+\mu}, \quad h > 0, \quad 0 \leq \mu \leq v \leq 1 \quad (3.3.26)$$

to include the w_{ij} and z_{ij} method coefficients,

$$\|a_{ij}(hL)\|_{X_v \leftarrow X_\mu} + \|b_i(hL)\|_{X_v \leftarrow X_\mu} + \|u_{ij}(hL)\|_{X_v \leftarrow X_\mu} + \|v_i(hL)\|_{X_v \leftarrow X_\mu} + \|w_{ij}(hL)\|_{X_v \leftarrow X_\mu} + \|z_i(hL)\|_{X_v \leftarrow X_\mu} \leq C h^{-v+\mu}, \quad h > 0, \quad 0 \leq \mu \leq v \leq 1 \quad (3.3.27)$$

which we can verify by applying (3.3.25).

In addition we extend the assumption [38, (3.1) Remark 1], that the function N is locally Lipschitz-continuous

$$\|N(v) - N(w)\|_X \leq C_1(\zeta) \|v - w\|_{X_\alpha}, \quad \|v\|_{X_\alpha} + \|w\|_{X_\alpha} \leq \zeta \quad (3.3.28)$$

and require that the derivatives $N^{(i)}$ are also locally Lipschitz-continuous

$$\left\| N^{(i)}(v) - N^{(i)}(w) \right\|_X \leq C_2(\zeta) \|v - w\|_{X_\alpha}, \quad \|v\|_{X_\alpha} + \|w\|_{X_\alpha} \leq \zeta \quad (3.3.29)$$

3.3.2 Proof of Convergence Bound

We are now in a position to present a modified formulation of Ostermann, Thalhammer & Wright's proof for [38, Theorem 3.1], applying the results to EAGLMs.

Theorem 15. *Under the requirements of Hypothesis 17, assume that EAGLMs of the form (3.3.6), applied to the initial value problem (1.3.2), satisfy (3.3.28) and fulfill the order conditions as defined in Theorem 3 and 19. Suppose that $N^{(\theta)}(t) \in X_\beta$ for some $0 \leq \beta \leq \alpha$ and $N^{(p)}(t) \in X$. Then for stepsizes $h > 0$ the estimate*

$$\begin{aligned} \|y(t_n) - y_n\|_{X_\alpha} &\leq C \sum_{l=0}^q \|y(t_l) - y_l\|_{X_\alpha} + Ch^{\theta+1-\alpha+\beta} \sup_{0 \leq t \leq t_n} \|N^{(\theta)}(t)\|_{X_\beta} \\ &\quad + Ch^p \sup_{0 \leq t \leq t_n} \|N^{(p)}(t)\|_X, \quad t_q \leq t_n \leq T \end{aligned} \quad (3.3.30)$$

holds with a constant $C > 0$ independent of n and h .

In addition we require that the approximation of $h^i N_n^{(i)}$, through a linear combination of N_{n-k} , $k \geq 0$ (3.3.7), is of order $O(h^{p+i})$.

Proof. Starting from the EAGLMs form of e_n (3.3.31), measured in the norm of X_α

$$\begin{aligned} \|e_n\|_{X_\alpha} &\leq \left\| e^{(t_n - t_{\theta-1})L} \right\|_{X_\alpha \leftarrow X_\alpha} \|e_{\theta-1}\|_{X_\alpha} + \left\| \sum_{l=q}^n e^{(t_n - t_l)L} d_l \right\| \\ &\quad + h \sum_{l=\theta-1}^{n-1} \sum_{i=1}^s \left\| e^{(t_n - t_{l+1})L} b_i(hL) \right\|_{X_\alpha \leftarrow X} \|\Delta N_{li}\|_X \\ &\quad + h \sum_{l=\theta-1}^{n-1} \sum_{k=1}^q \left\| e^{(t_n - t_{l+1})L} v_k(hL) \right\|_{X_\alpha \leftarrow X} \|\Delta N_{l-k}\|_X \\ &\quad + h \sum_{l=\theta-1}^{n-1} \sum_{j=1}^r \left\| e^{(t_n - t_{l+1})L} z_j(hL) \right\|_{X_\alpha \leftarrow X} h^j \|\Delta N_l^{(j)}\|_X \end{aligned} \quad (3.3.31)$$

we look at the terms involving $\Delta N_l^{(m)}$, these are bounded

$$\sum_{m=1} \left\| e^{(t_n - t_{l+1})L} z_m(hL) \right\|_{X_\alpha \leftarrow X} h^m \|\Delta N_l^{(m)}\|_X \leq Ch(t_n - t_l)^{-\alpha} \sum_{m=1} h^m \|\mathcal{E}_{lm}\|_{X_\alpha} \quad (3.3.32)$$

$$\mathcal{E}_{nm} \in \{e_{n-k}, E_{(n-1)i}\}, \quad k = 0, \dots, q, \quad i = 1, \dots, s.$$

In combination with the similar bounds on the ΔN_{li} and ΔN_{l-k} terms as presented in [38] we have the result

$$\begin{aligned} \|e_n\|_{X_\alpha} &\leq C \|e_{\theta-1}\|_{X_\alpha} + \left\| \sum_{l=q}^n e^{(t_n-t_l)L} d_l \right\|_{X_\alpha} \\ &\quad + Ch \sum_{l=\theta-1}^{n-1} (t_n - t_l)^{-\alpha} \left(\sum_{i=1}^s \|E_{li}\|_{X_\alpha} + \sum_{k=1}^q \|e_{l-k}\|_{X_\alpha} + \sum_{m=1} h^m \|\mathcal{E}_{lm}\|_{X_\alpha} \right). \end{aligned} \quad (3.3.33)$$

Here the constant $C > 0$ depends on T , but is independent of h . This is the EAGLMs equivalent of [38, (3.12)].

Likewise, looking at the error of the internal stages (3.3.21)

$$\begin{aligned} \|E_{li}\|_{X_\alpha} &\leq \|e^{c_i h L}\|_{X_\alpha \leftarrow X_\alpha} \|e_l\|_{X_\alpha} + h \sum_j^{i-1} \|a_{ij}\|_{X_\alpha \leftarrow X} \|\Delta N_{lj}\|_X \\ &\quad + h \sum_k^q \|u_{ik}\|_{X_\alpha \leftarrow X} \|\Delta N_{l-k}\|_X + h \sum_m^r \|w_{im}\|_{X_\alpha \leftarrow X} h^m \|\Delta N_l^{(m)}\|_X + \|D_{li}\|_{X_\alpha} \end{aligned} \quad (3.3.34)$$

and focusing on the additional terms introduced by the EAGLM formulation, those being the terms involving $\Delta N_l^{(m)}$, we have the bound

$$\sum_{m=1} \|w_{im}(hL)\|_{X_\alpha \leftarrow X} h^m \|\Delta N_l^{(m)}\|_X \leq Ch^{1-\alpha} \sum_{m=1} h^m \|\mathcal{E}_{lm}\|_{X_\alpha} \quad (3.3.35)$$

which leads us to the estimate

$$\begin{aligned} \|E_{li}\|_{X_\alpha} &\leq C \|e_l\|_{X_\alpha} + Ch^{1-\alpha} \sum_{j=1}^{i-1} \|E_{lj}\|_{X_\alpha} + Ch^{1-\alpha} \sum_{k=1}^q \|e_{l-k}\|_{X_\alpha} \\ &\quad + Ch^{1-\alpha} \sum_{m=1} h^m \|\mathcal{E}_{lm}\|_{X_\alpha} + \|D_{li}\|_{X_\alpha} \end{aligned} \quad (3.3.36)$$

From this we follow [38] and insert this relation into (3.3.33), to recover the bound [38, (3.13)] for EAGLMs,

$$\begin{aligned} \|e_n\|_{X_\alpha} &\leq C \|e_{\theta-1}\|_{X_\alpha} + Ch \sum_{l=0}^{n-1} (t_n - t_l)^{-\alpha} \|e_l\|_{X_\alpha} \\ &\quad + Ch \sum_{l=\theta-1}^{n-1} \sum_{i=1}^s (t_n - t_l)^{-\alpha} \|D_{li}\|_{X_\alpha} + \left\| \sum_{l=q}^n e^{(t_n-t_l)L} d_l \right\|_{X_\alpha} \end{aligned} \quad (3.3.37)$$

Next we must also estimate the terms involving the defects, (3.3.11) and (3.3.12). The assumption that the order conditions are satisfied implies $D_{li} = R_{li}^{(q)}$ and $d_l = \varrho_l^{(p)}$, giving the formulation for the remainders as in (3.3.19). From the condition imposed on $h^i N_n^{(i)}$, the following bounds on $S_n(m, \sigma)$ (3.3.4) and $\mathcal{S}_{ni}(m)$ (3.3.9),

$$\|S_l(\theta, c_j h)\|_{X_\beta} + \|S_l(\theta, -kh)\|_{X_\beta} + \|\mathcal{S}_{nl}(\theta)\|_{X_\beta} \leq Ch^\theta \|f^{(\theta)}\|_{X_{\beta, \infty}} \quad (3.3.38)$$

imply that

$$\begin{aligned}
\left\| R_{li}^{(\theta)} \right\|_{X_\alpha} &\leq \|R_l(\theta, c_i h)\|_{X_\alpha} + h \sum_{j=1}^{i-1} \|a_{ij}\|_{X_\alpha \leftarrow X_\beta} \|S_l(\theta, c_j h)\|_{X_\beta} \\
&\quad + h \sum_{k=1}^q \|u_{ik}\|_{X_\alpha \leftarrow X_\beta} \|S_l(\theta, -kh)\|_{X_\beta} + h \sum_{m=1}^r \|w_{im}\|_{X_\alpha \leftarrow X_\beta} \|\mathcal{S}_{nl}(\theta)\|_{X_\beta} \\
&\leq Ch^{\theta+1-\alpha+\beta} \left\| f^{(\theta)} \right\|_{X_{\beta, \infty}}
\end{aligned} \tag{3.3.39}$$

and

$$\begin{aligned}
\left\| \varrho_l^{(p)} \right\|_X &\leq \|R_{l-1}(p, h)\|_X + h \sum_{i=1}^{l-1} \|b_i\|_{X \leftarrow X} \|S_{l-1}(p, h)\|_X \\
&\quad + h \sum_{k=1}^q \|v_k\|_{X \leftarrow X} \|S_l(p, -kh)\|_X + h \sum_{m=1}^r \|z_m\|_{X \leftarrow X} \|\mathcal{S}_{nl}(p)\|_X \\
&\leq Ch^{p+1} \left\| f^{(p)} \right\|_{X, \infty}
\end{aligned} \tag{3.3.40}$$

With these two bounds established we recover the bound [38, (3.14)] which now holds for EAGLMs.

$$\sum_{l=q}^{n-1} \left\| e^{(t_n - t_l)L} \right\|_{X_\alpha \leftarrow X} \left\| \varrho_l^{(p)} \right\|_X + \left\| \varrho_n^{(p)} \right\|_{X_\alpha} \leq Ch^{p+1} \sum_{l=q}^{n-1} (t_n - t_l)^{-\alpha} \left\| f^{(p)} \right\|_{X, \infty} \tag{3.3.41}$$

Having shown that EAGLMs satisfy all the same bounds as EGLMs the results of [38, Theorem 3.1] hold for EAGLMs. \square

Ostermann, Thalhammer & Wright establish in [38], that the convergence of an EGLM is $\min(p, \theta + 1)$, where p is the quadrature order and θ is the stage order. Here, $q = \theta - 1$ steps are necessary to achieve a stage order of θ . We can see from Theorem 15, that the convergence of an EAGLM is also $\min(p, \theta + 1)$, where now, the requirement on the number of steps and derivative values is $q + r = \theta - 1$.

3.3.3 Equivalence between EGLMs and EAGLMs

We will look at how there is an equivalence between EAGLMs and EGLMs. We will show this by looking at EGLM₃₂₂ c_2 (2.2.6) and EARK₃₂₁ c_2 (3.2.30).

Writing EGLM₃₂₂ c_2 (2.2.6) out explicitly in the form of (2.2.3) we get,

$$\begin{aligned}
Y_2 &= e^{c_2 h L} y_n + h \left[(c_2 \varphi_{21} + c_2^2 \varphi_{22}) N_n - c_2^2 \varphi_{22} N_{n-1} \right] \\
K_2 &= N(t_n + c_2 h, Y_2)
\end{aligned} \tag{3.3.42}$$

$$\begin{aligned}
y_{n+1} &= e^{hL} y_n + h \left[\left(\varphi_1 + \frac{c_2 - 1}{c_2} \varphi_2 + \frac{-2}{c_2} \varphi_3 \right) N_n + \left(\frac{1}{c_2^2 + c_2} \varphi_2 + \frac{2}{c_2^2 + c_2} \varphi_3 \right) K_2 \right. \\
&\quad \left. + \left(\frac{-c}{c_2 + 1} \varphi_2 - \frac{2}{c_2 + 1} \varphi_3 \right) N_{n-1} \right]
\end{aligned} \tag{3.3.43}$$

We can see that N_n and the previous step N_{n-1} are both multiplied by a linear combination of φ -functions. If we now look at EARK_{321c2} (3.2.30), written explicitly in the form of (3.2.3) we get

$$Y_2 = e^{c_2 h L} y_n + h [c_2 \varphi_{21} N_n - c_2^2 \varphi_{22} N'_n] \quad (3.3.44)$$

$$K_2 = N(t_n + c_2 h, Y_2)$$

$$y_{n+1} = e^{hL} y_n + h \left[\left(\varphi_1 - \frac{2}{c_2^2} \varphi_3 \right) N_n + \frac{2}{c_2^2} \varphi_3 K_2 + \left(\varphi_2 - \frac{2}{c_2} \varphi_3 \right) N'_n \right] \quad (3.3.45)$$

If we approximate N'_n by $\frac{3}{2}N_n - 2N_{n-1} + \frac{1}{2}N_{n-2}$, we can see that (3.3.44) and (3.3.45) become

$$\begin{aligned} Y_2 &= e^{c_2 h L} y_n + h [c_2 \varphi_{21} N_n - c_2^2 \varphi_{22} (\frac{3}{2}N_n - 2N_{n-1} + \frac{1}{2}N_{n-2})] \\ &= e^{c_2 h L} y_n + h [(c_2 \varphi_{21} - \frac{3}{2}c_2^2 \varphi_{22}) N_n + 2\varphi_{22} c_2^2 N_{n-1} - \frac{1}{2}\varphi_{22} c_2^2 N_{n-2}] \end{aligned} \quad (3.3.46)$$

$$K_2 = N(t_n + c_2 h, Y_2)$$

$$\begin{aligned} y_{n+1} &= e^{hL} y_n + h \left[\left(\varphi_1 + \frac{3}{2}\varphi_2 - \frac{3c_2+2}{c_2^2} \varphi_3 \right) N_n + \frac{2}{c_2^2} \varphi_3 K_2 + \right. \\ &\quad \left. \left(\frac{4}{c_2} \varphi_3 - 2\varphi_2 \right) N_{n-1} + \left(\frac{1}{2}\varphi_2 - \frac{1}{c_2} \varphi_3 \right) N_{n-2} \right] \end{aligned} \quad (3.3.47)$$

which is a 3rd Order EGLM, albeit one which uses two previous steps within the method rather than just one as (2.2.6) does.

For any EAGLM where the $N_n^{(i)}$'s are approximated by a linear combination of the current N_n , and previous steps N_{n-j} , we can rewrite the scheme as an EGLM. This means, that for all EAGLMs satisfying that condition on the derivatives, the results of [38, Theorem 3.1] automatically hold.

3.3.4 Summary

We have introduced in this chapter two new families of EIs. The first family, EARKs, pass derivatives from one step to the next and are considered to be multi-value schemes. This property, together with their multi-stage nature, makes them a special case of EGLMs, though with the critical distinction of being 1-step methods.

We also introduced in this chapter a broader framework of EAGLMs, this framework allowed us to create a unified representation of the earlier families. We showed that once order conditions for this family were derived, the order conditions for the other families could be recovered as special cases of the EAGLM conditions. A convergence analysis of EAGLMs showed that they also satisfy the convergence bounds which Ostermann, Thalhammer & Wright proved for EGLMs.

Through numerical experiments in Chapter 5, we will show that EARK schemes demonstrate excellent accuracy performance together with a very efficient implementation. When we look to further optimizing the implementation of schemes we will see that approximating the outgoing derivative approximations to sufficient order can limit the optimisation opportunities. The switch to the EAGLM family provides the additional flexibility to allow one to construct schemes with the superior accuracy of EARKs, combined with implementations that outperform EGLMs in computational cost.

Chapter 4

Stability

In this chapter we study the stability regions of the families of EIs introduced in the last chapter.

We follow the analysis method of Cox & Matthews [13], that is, we linearise the autonomous ODE,

$$u' = Lu + N(u)$$

about a fixed point u_0 to obtain,

$$u' = Lu + \lambda u \tag{4.0.1}$$

where $\lambda = N'(u_0)$.

We then apply a specific EI scheme to (4.0.1) and plot the boundary curves of the stability regions. Comparing these boundary curves allows us to gauge the relative stability of different schemes and identify desirable stability properties.

4.1 Stability Analysis

In numerical analysis stability is a measure of the extent to which a numerical scheme gives useful approximations. Stability analysis helps identify the range of step-sizes for which a scheme provides numerically stable results. We will compare and analyse the stability properties of some of the methods we have seen so far. In general, the techniques used in this section can only give an indication of the relative stability of different schemes.

The standard method of analysing stability is to apply the scheme to the test problem

$$u' = \lambda u, \quad \text{where } u = u_r + iu_i \in \mathbb{C} \quad (4.1.1)$$

A discretisation method is applied to (4.1.1) to obtain a homogeneous linear difference equation,

$$\sum_{k=0}^M (\alpha_k u_{n-k} - \lambda \Delta t \beta_k u_{n-k}) = 0 \quad (4.1.2)$$

which is solved explicitly. Then we consider a region of the complex plane where the solution is bounded.

The approach for analysing the stability of schemes which treat the linear and non-linear parts of the equation separately is developed in [5, 13]. We compute a number of boundary curves of the stability regions of a more general test problem

$$u' = -Lu + \lambda u \quad (4.1.3)$$

with each curve corresponding to different values of L . After we discretise (4.1.3) we can fix the parameter $y = hL$ and plot the boundary of the respective stability region.

4.2 Comparisons with mixed Explicit-Implicit Schemes

Beylkin, Keiser & Vozovoi [5] studied the stability behaviour of ELP schemes both explicit and implicit with the stability properties of known implicit-explicit scheme, namely AMAB schemes. Figure 4.1 reproduces the stability plots for the two mixed explicit-implicit schemes presented in that paper for fixed $y = -2$. Citing [19] they note that a method is stiffly stable if it is consistent in a neighbourhood about the origin and absolutely stable away from the origin in the left imaginary plane.

They concluded that both explicit and implicit ELP schemes have excellent stability properties. In particular the stability regions of explicit ELP schemes are reminiscent of classical

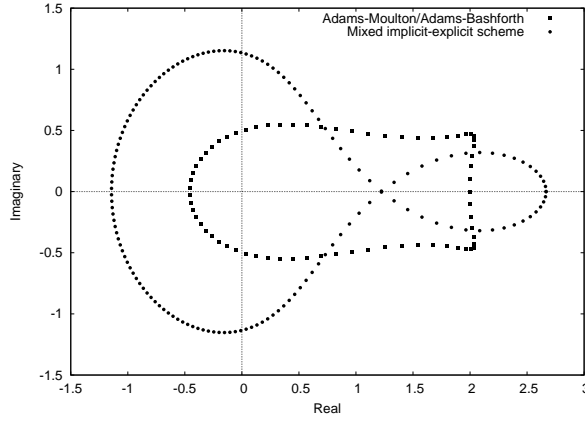


Figure 4.1: Mixed explicit-implicit schemes stability boundaries for $y = -2$

implicit schemes. They also note that growth of the stability region as L becomes larger is a very important property for a scheme to be useful, and is a property which the ELP schemes exhibit.

4.3 Stability of ERKs

Cox & Matthews [13] study the stability of several 2nd-order schemes and in particular, some linearly implicit schemes such as an AMAB, and the standard Integrating Factor methods; Integrating Factor/Adams-Bashford Method (IFAB) and Integrating Factor/Classical Runge-Kutta Method (IFRK). These are compared with a number of ETDs and ERKs. To perform this analysis for such composite schemes, we use the same approach as in [5]. We linearise the non-linear, autonomous ODE test problem (4.1.3) about a fixed point u_0 , such that $Lu_0 + N(u_0) = 0$ leading to

$$u' = Lu + \lambda u \quad (4.3.1)$$

where u is now the perturbation to u_0 and $\lambda = N'(u_0)$. The fixed point u_0 is stable if

$$\text{Re}(L + \lambda) < 0 \quad (4.3.2)$$

As noted earlier, this technique can only provide us with a relative comparison of the stabilities of various schemes.

In general, both L and λ are complex so the resulting stability region is four-dimensional. In order to plot two dimensional stability regions, we will look at two cases. First, by assuming λ is complex and that L is fixed, negative and real, we can plot the resulting stability regions in the complex plane. Second we look at the case where L is not fixed and both L and λ are real.

The analysis is performed by applying a scheme to the test problem (4.1.3). For ERK₂ (2.1.19) with $c_2 = 1$, that results in the following expression,

$$u_{n+1} = u e^{hL} + h \left(\frac{\lambda \left(\frac{e^{hL}-1}{hL} - 1 \right) \left(u e^{hL} + \frac{\lambda u (e^{hL}-1)}{L} \right)}{hL} \right) + h \left(\lambda u \left(\frac{e^{hL}-1}{hL} - \frac{e^{hL}-1}{hL} \right) \right) \quad (4.3.3)$$

then by setting $r = u_{n+1}/u_n$, $x = h\lambda$ and $y = hL$ we obtain

$$r = \frac{(xy + x^2) e^{2y} + (y^3 + (-x^2 - 2x) y - 2x^2) e^y + (x^2 + x) y + x^2}{y^3} \quad (4.3.4)$$

In the first case we will fix $y < 0 \in \mathbb{R}$. We wish to plot the boundary of the stability region which occurs when $r = 1$. To plot this in the complex plane we set $r = e^{i\theta}$ and solve for x on the interval $\theta \in [0, 2\pi)$.

We can see in Figure 4.2(a) the stability region boundaries for ERK₂ (2.1.19) when $y = -1, -2, -5$ respectively. We can identify from the boundary curves that the stability regions of (2.1.19) are broader than those of the mixed explicit-implicit schemes from Figure 4.1.

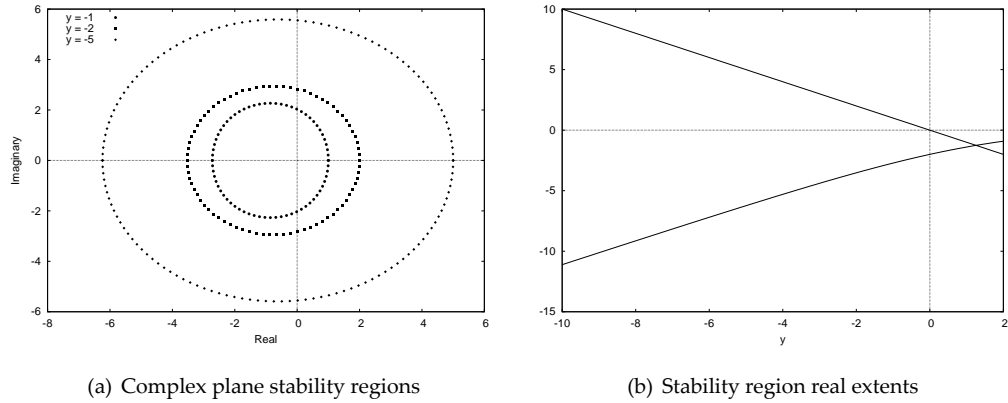


Figure 4.2: ERK₂ (2.1.19) stability graphs

For the second case, we fix $r = 1$, and plot the growth of the real extents of the stability regions against a varying y . For (2.1.19), the solutions to (4.3.4) are

$$x = -\frac{y^2}{e^y - y - 1} \quad x = -y \quad (4.3.5)$$

and Figure 4.2(b). shows a graph of those solutions. We can see as y grows in magnitude the stability regions real extents also grow approximately linearly.

Figure 4.3(a) reproduces the Cox and Matthew plots for ETD2 (1.3.14). The region of stability is significantly smaller than that of ERK₂ (2.1.19) and Figure 4.3(b) highlights the slower growth of the real extents of that stability region as y grows.

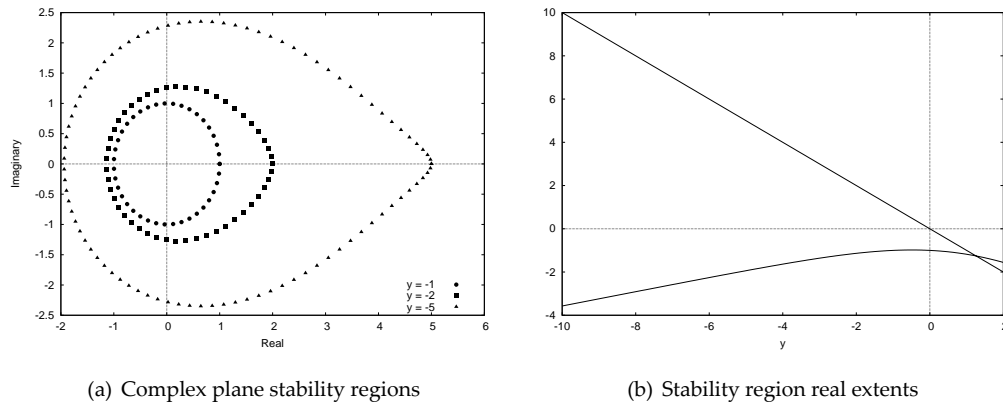


Figure 4.3: ETD2 (1.3.14) stability boundaries

Finally in [27], Krogstad investigated the stability regions of a number of fourth order schemes, notably his own ERK_4 Krogstad (5.3.5), and some multi-step generalization of IF methods, also developed by him. He came to the conclusion that (5.3.5) had the largest stability region.

He also suggests that, since the generalized IF methods touch the imaginary axis only at a point, they will not perform well for equations where the eigenvalues of the linearisation lie on the imaginary axis.

4.4 Unconditional Stability of ERKs

In [32], Maset & Zennaro studied the stability of ERK methods and, in particular, they looked at the necessary requirement for such methods to have unconditional stability. Before looking at their results, we need to introduce some notation. We will use $\mu(L)$ to represent the logarithmic norm of L , and γ for Lipschitz condition of the function $N(t, (t))$ with respect to the second parameter

$$\|N(t, y_1) - N(t, y_2)\| \leq \gamma \|y_1 - y_2\|. \quad (4.4.1)$$

They consider the linear system (1.3.2) at two different starting values, u_0 and v_0 such that

$$u'(t) = Lu(t) + N(t, u(t)), \quad t \geq 0, \quad u(t) = u_0, \quad (4.4.2)$$

and

$$v'(t) = Lv(t) + N(t, v(t)), \quad t \geq 0, \quad v(t) = v_0, \quad (4.4.3)$$

To guarantee contractivity for the system, we require that the condition

$$\mu(L) + \gamma \leq 0 \quad (4.4.4)$$

must hold. This implies

$$\|\delta(t)\| \leq \|\delta_0\|, \quad t \geq 0 \quad (4.4.5)$$

where

$$\delta(t) = u(t) - v(t)$$

$$\delta_0 = u_0 - v_0$$

for all u_0 and v_0 . Also, if we replace the \leq in (4.4.4) with $<$ then the system is asymptotically stable, that is

$$\|\delta(t)\| \rightarrow 0 \quad t \rightarrow -\infty \quad (4.4.6)$$

It is worth noting also, that this result, when applied to (4.1.3), reproduces the requirement (4.3.2) identified earlier.

Applying ERKs with stepsize h to the linear system at different starting values, (4.4.2) and (4.4.3), the differences

$$\delta_{n+1} = u_{n+1} - v_{n+1}$$

$$\Delta_{n+1}^i = U_i - V_i, \quad i = 1, \dots, s$$

where

$$u_{n+1} = e^{hL}u_n + h \sum_{i=1}^s b_i(hL) N(U_i)$$

$$U_i = e^{c_i hL}u_n + h \sum_{j=1}^{i-1} a_{ij}(hL) N(U_j)$$

and v_{n+1} and V_i are defined similarly. Then δ_{n+1} and Δ_{n+1}^i satisfy the following,

$$\delta_{n+1} = e^{hL}\delta_n + h \sum_{i=1}^s b_i(hL) [N(U_i) - N(V_i)] \quad (4.4.7)$$

$$\Delta_{n+1}^i = e^{c_i hL}\delta_n + h \sum_{j=1}^{i-1} a_{ij}(hL) [N(U_j) - N(V_j)] \quad (4.4.8)$$

Let \mathcal{M} be a class of matrices closed under positive scalar multiplication. The stability properties are then studied on the class $C(\mathcal{M})$ with $L \in \mathcal{M}$. We introduce, for $\alpha \leq 0$, the $(1 \times v)$ -vector $\bar{b}(\alpha)$ with

$$\bar{b}_i(\alpha) = \sup_{M \in \mathcal{M}, \mu(M) \leq \alpha} \|b_i(M)\|, \quad i = 1, \dots, v \quad (4.4.9)$$

and the $(s \times s)$ -matrix $\bar{A}(\alpha)$ whose elements are

$$\bar{a}_{ij}(\alpha) = \sup_{M \in \mathcal{M}, \mu(M) \leq \alpha} \|a_{ij}(M)\|, \quad i, j = 1, \dots, s \quad (4.4.10)$$

Maset & Zennaro then give a bound for $\|\delta_{n+1}\|$ written in terms of $\|\delta_n\|$ which holds when $L \in \mathcal{M}$. If

$$\|\delta_{n+1}\| \leq e^{\mu(hL)} \|\delta_n\| + h\gamma \sum_{i=1}^s \|b_i(hL)\| \|\Delta_{n+1}^i\| \quad (4.4.11)$$

$$\|\Delta_{n+1}^i\| \leq e^{c_i \mu(hL)} \|\delta_n\| + h\gamma \sum_{j=1}^{i-1} \|a_{ij}(hL)\| \|\Delta_{n+1}^j\| \quad (4.4.12)$$

then

$$\|\delta_{n+1}\| \leq e^{h\mu(hL)} \|\delta_n\| + h\gamma \bar{b}(h\mu(L)) \bar{\Delta}_{n+1}, \quad (4.4.13)$$

$$(I - h\gamma \bar{A}(h\mu(L))) \bar{\Delta}_{n+1} \leq e^{ch\mu(hL)} I_s \|\delta_n\| \quad (4.4.14)$$

Noting here that the matrix $\bar{A}(h\mu(L))$ is strictly lower triangular, they obtain the bound

$$\|\delta_{n+1}\| \leq \bar{S}(h\mu(L), h\gamma) \|\delta_n\| \quad (4.4.15)$$

where

$$\bar{S}(\alpha, \beta) := e^\alpha + \sum_{k=0}^{s-1} \beta^{k+1} \bar{b}(\alpha) \bar{A}(\alpha)^k e^{c\alpha} 1_v, \quad \alpha \in \mathbb{R} \text{ and } \beta \geq 0 \quad (4.4.16)$$

and prove the following

Lemma 16. *If an explicit Exponential Runge-Kutta Method (ERK) satisfies*

$$\bar{S}(-\beta, \beta) \leq 1, \quad \beta \geq 1 \quad (4.4.17)$$

then it is unconditionally contractive and asymptotically stable on the class $\mathcal{C}(\mathcal{M})$ with respect to the norm $\|\cdot\|$

The parameters α and β to function \bar{S} can be related back to stability plot analysis as they correspond to the variables $y = hL$ and $x = h\lambda$ respectively.

Since \bar{S} is an increasing function in it's first parameter, an unconditionally stable scheme remains that way so long as $\alpha + \beta \leq 0$. Therefore when looking at stability plots for such schemes we would expect $r = \frac{u_{n+1}}{u_n} \leq 1$ for all $x \leq -y$.

Example ERK

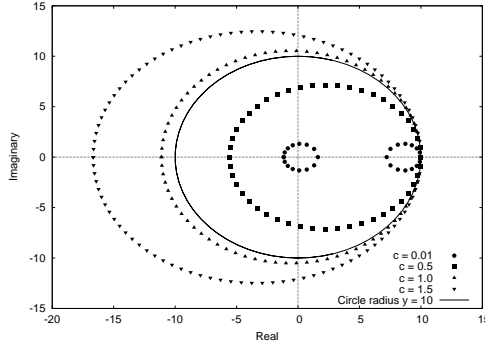
Maset and Zennaro also extend these results for the general case to the two ERKs, both of which we saw earlier (Section 2.1.3). The first, ERK₂ (2.1.19),

0		I
c_α	$c_\alpha \varphi_{1,2}$	$e^{c_2 hL}$
	$\varphi_1 - \frac{1}{c_\alpha} \varphi_2$	$\frac{1}{c_\alpha} \varphi_2$

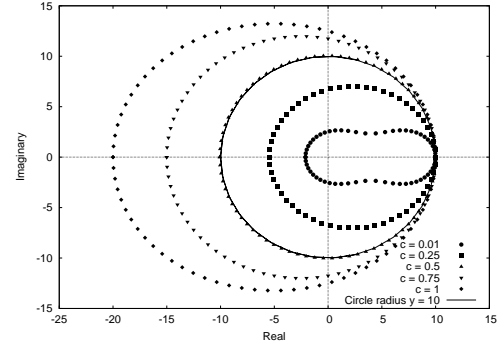
and ERK₂ (2.1.21),

0	I
c_β	$c_\beta \varphi_{1,2}$
$(1 - \frac{1}{2c_\beta})\varphi_1$	$\frac{1}{2c_\beta}\varphi_1$

and they demonstrate that they are unconditionally stable when $c_\alpha \geq 1$ and $c_\beta \geq \frac{1}{2}$.



(a) ERK₂ (2.1.19) Stability Boundaries



(b) ERK₂ (2.1.21) Stability Boundaries

This result reinforces the stability region plots, see Figures 4.4(a) and 4.4(b), which show that the stability regions of the respective scheme contain the circular region $x \leq -y$ only when the conditions on c_α and c_β are met.

4.5 Stability of EGLMs

In Section 2.2 that Ostermann, Thalhammer & Wright [38], introduced the family of EGLMs, which combine exponential explicit Runge-Kutta methods with exponential Adams-Bashforth methods.

They provide a convergence analysis for this class of schemes and use those results to construct some example methods. By applying our analysis through boundary plots to one such method, EGLM₃₂₂ c_2 (2.2.6), we can see how the many desirable properties mentioned earlier are present. Figures 4.4(c) and 4.4(d) show those plot for (2.2.6).

We can see from Figure 4.4 that the stability region of EGLM₃₂₂ c_2 (2.2.6) extends back into the negative complex plane. This is a property observed in the ERK example, but not in ETD2 plot.

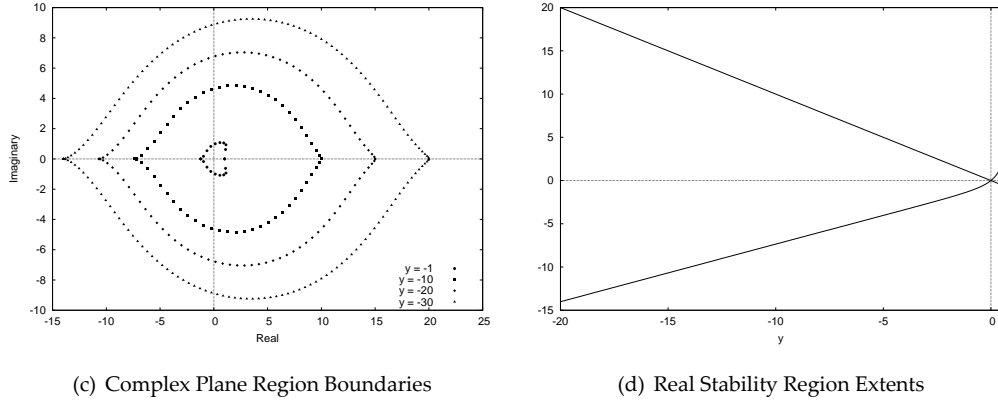


Figure 4.4: $\text{EGLM}_{322}c_2$ (2.2.6) Stability Plots

4.6 Stability of EARKs

Finally, we can apply this analysis to the new EARK schemes. In performing the analysis we use previous steps to approximate the derivatives of N needed by the methods. Figure 4.5 plots both the stability regions for $\text{EARK}_{321}c_2$ (3.2.30). Note that the growth of the stability regions is again linear with L . In this particular case the value of N' was approximated by $-2N_{n-2} + \frac{1}{2}N_{n-1} + \frac{2}{1}N_n = N' + O(h^3)$.

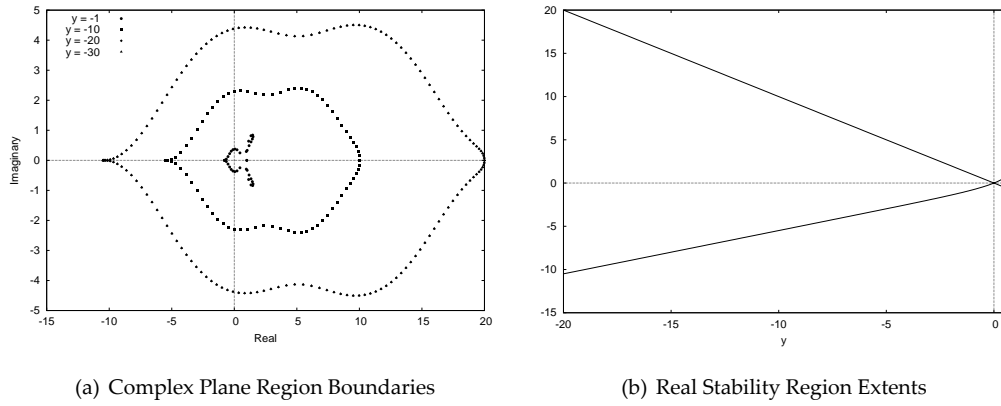


Figure 4.5: $\text{EARK}_{321}c_2$ (3.2.30) Stability Plots

Comparisons with earlier methods

In getting a better handle on the comparative stabilities of the schemes seen so far, we produce some plots with the stability regions of different schemes plotted side-by-side for fixed $hL = y$. Figure 4.6 shows the two primary 3rd order schemes; $\text{EGLM}_{322}c_2$ (2.2.6) and $\text{EARK}_{321}c_2$ (3.2.30) for $y = 20$.

It is evident here that the EARK scheme has a smaller stability region than the EGLM. However given that EARK schemes have a lower computational cost per step over EGLMs, we can scale the EARK region by the savings in φ -vector products. This is similar to a technique used by Butcher in [8] where he scaled the stability regions relative to the number of internal stages of the classical schemes being compared. Figure 4.7 shows that new stability region and we can see now that the real axis stability extents of the EARK scheme exceed those of the EGLM scheme.

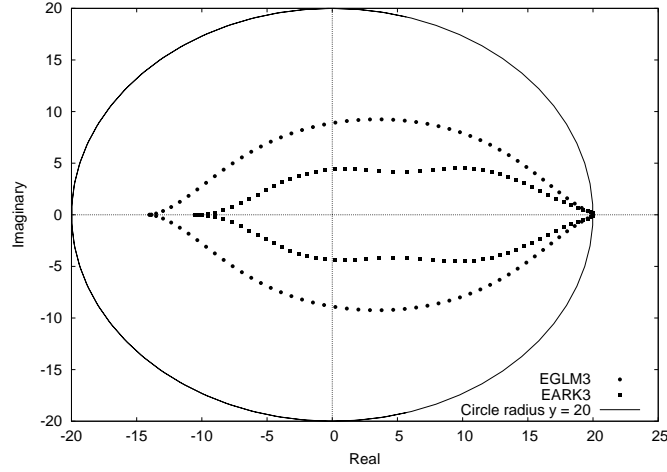


Figure 4.6: $\text{EGLM}_{322}c_2$ (2.2.6) and $\text{EARK}_{321}c_2$ (3.2.30) side-by-side stability regions

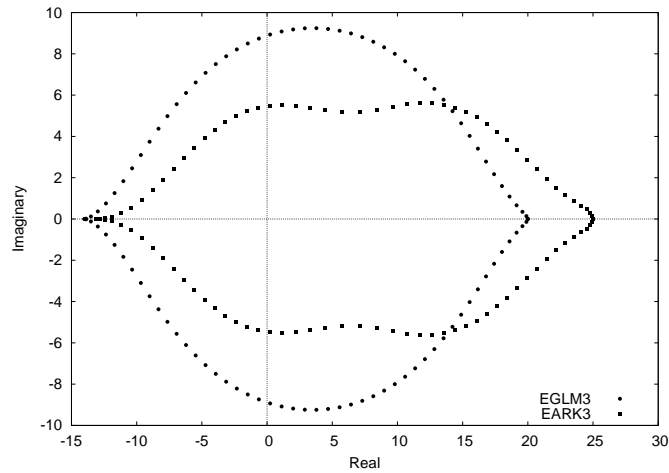


Figure 4.7: $\text{EGLM}_{322}c_2$ (2.2.6) with scaled $\text{EARK}_{321}c_2$ (3.2.30) stability region

4.7 Summary

We compared the stability region of ERK_2 (2.1.19) with that of ETD_2 (1.3.14) and showed that the region for ETD_2 was significantly smaller, in particular with regards to its extension into

negative complex plane. This result supported what we expected from literature; that ERKs have superior stability properties over ETDs.

By performing the same stability analysis for $\text{EGLM}_{322}c_2$ (2.2.6) and $\text{EARK}_{321}c_2$ (3.2.30), we observed that both families of methods also exhibited stability regions extending into the negative complex plane. This indicated that both possessed similar stability properties to those of ERKs. In addition, our study showed that the superior performance of EARKs offset their slightly weaker stability properties compared with EGLMs.

Chapter 5

Numerical Experiments

In this chapter we study the computational costs involved in implementing EIs. To do this we

- Study the CPU and memory costs associated with the φ -functions. These functions represent a bottleneck for all EIs and we look at the different approaches which have been developed to compute the matrix exponential and the related φ -functions.
- Analyse the relative per-step accuracy, and the per-step cost, of the schemes introduced earlier. We run numerical tests, plotting accuracy against stepsize, to visualise the convergence orders of a wide selection of schemes. We also analyse the computational costs involved when assuming an optimal implementation.
- Review the families of schemes from a variable stepsize perspective and we discuss any changes needed to ensure that the schemes can cope in a variable stepsize environment.
- Look at local truncation errors and the different approaches to estimating it. We use one such approach, known as embedding, to construct an EGLM and EARK scheme which produce local truncation errors estimates. We use this estimate to guide an adaptive step-size controller.
- Implement a complete adaptive integrator and perform a comprehensive benchmark against a suite of test problems. We see that an EARK-based scheme is the best performing EI, and on large problems it outperforms Matlab's ODE15s.

All numerical experiments were performed in Matlab 2010b 64bit running on Windows 7 x64. The CPU was an Intel Core 2 Quad Q9450 clocked at 2.66GHz and the system had 8GB of RAM.

5.1 Test Problems

We have selected a number of test problems to help determine the relative rankings of the EIs seen so far, and demonstrate the superior performance of EAGLMs.

5.1.1 The Kuramoto-Sivashinsky Equation

The Kuramoto-Sivashinsky Equation is a 1D PDE of the form

$$y_t = -y_{xx} - y_{xxxx} - yy_x \quad (5.1.1)$$

with periodic boundary conditions and with the initial condition,

$$y(0, x) = \cos\left(\frac{1}{16}\right) \left(1 + \sin\left(\frac{1}{16}\right)\right) \quad (5.1.2)$$

with $x \in [0, 32\pi]$. The PDE exhibits complex dynamical behaviour and which arises in a great number of applications [1]. It is an example of deterministic chaos [26].

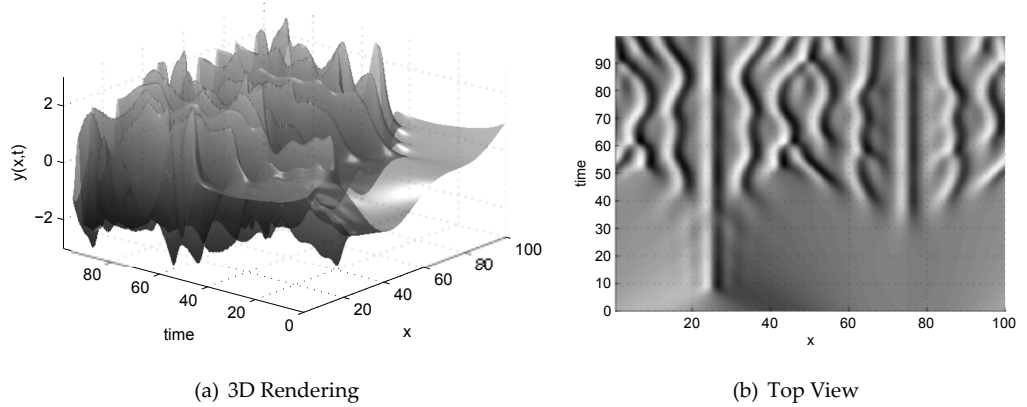


Figure 5.1: The 1D Kuramoto-Sivashinsky Equation Solution (5.1.1)

The second-derivative term acts as an energy source and destabilises the system, while the fourth-derivative term has a stabilising effect. The non-linear term acts to transfer energy for low to high wavenumbers [26].

We avail of the EXPINT package's implementation [4, Section 4.2.3]. The code uses a Fourier spectral discretisation, which results in a diagonal L matrix. This linear part of the problem is very stiff due to the fourth derivative [13].

5.1.2 Brusselator System

One of the key test problems we use for the purpose of studying the variations in local truncation error, is the Brusselator System [21]. It models diffusion in a chemical reaction

$$\begin{aligned} u'_i &= 1 + u_i^2 v_i - 4u_i + \alpha (N+1)^2 (u_{i-1} - 2u_i + u_{i+1}) \\ v'_i &= 3u_i + u_i^2 v_i + \alpha (N+1)^2 (v_{i-1} - 2v_i + v_{i+1}) \end{aligned} \quad (5.1.3)$$

and is solved on the time interval $[0, 10]$ with $\alpha = 1/50$ and initial conditions,

$$\left. \begin{aligned} u_i(0) &= 1 + \sin(2\pi x_i) \\ v_i(0) &= 3 \end{aligned} \right\} \quad \text{with } x_i = \frac{i}{(N+1)}, \quad \text{for } i = 1, \dots, N \quad (5.1.4)$$

There are $2N$ equations in the system, but the Jacobian is banded with a constant width 5 if the equations are ordered as $u_1, v_1, u_2, v_2, \dots$. As N increases, the problem becomes increasingly stiff.

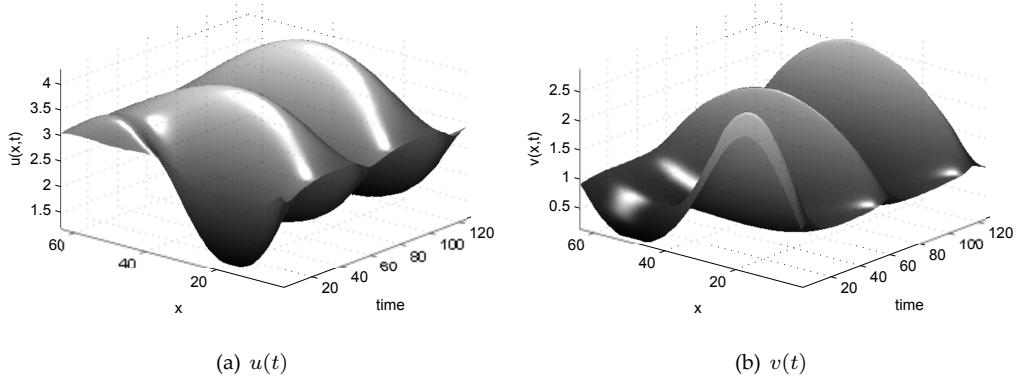


Figure 5.2: The Brusselator System Solution (??)

Figure 5.2 shows a rendering of the solutions of u and v . Our implementation is based on Matlabs BRUSSODE.

5.1.3 The Allen-Cahn Equation

The Allen-Cahn equation is a 1D parabolic PDE

$$y_t = \lambda y_{xx} + y - y^3, \quad (5.1.5)$$

with initial condition,

$$y(0, x) = 0.53x + 0.47 \sin(-1.5\pi x)$$

and boundary conditions,

$$y(t, -1) = -1, \quad y(t, 1) = 1$$

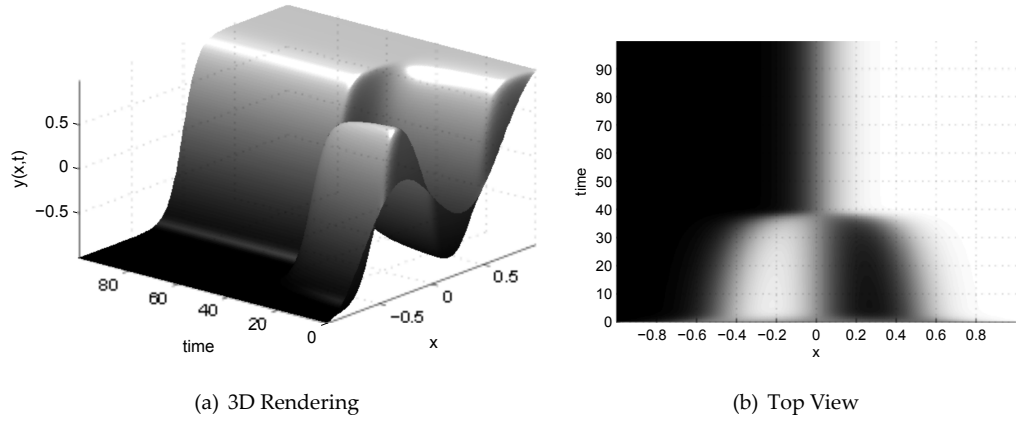


Figure 5.3: The Allen-Cahn Equation (5.1.5)

Looking at the illustration of the problem in Figure 5.3 we can see that it exhibits interesting behaviour about $t = 37$. Here the solution switches abruptly, from being metastable, to a lower energy state [12].

We make use of the implementation supplied by the EXPINT Matlab package [4, Section 4.2.6]. The boundary conditions are handled by defining, $y = w + x$, and working with w , giving us

$$\begin{aligned} w_t &= \lambda w_{xx} + (w + x) - (w + x)^3 \\ w_0 &= 0.47(-x_i + \sin(-1.5\pi x_i)) \end{aligned} \quad (5.1.6)$$

which has homogeneous boundary conditions, with $x \in [-1, 1]$. The use of a Chebyshev differentiation matrix for discretising the linear part, λw_{xx} , means that the L matrix is dense.

5.1.4 Hochbruck & Ostermann Parabolic PDE

The main test problem we will use in fixed stepsize experiments is the Hochbruck & Ostermann Parabolic PDE [25, Problem 6.1]

$$y_t = y_{xx} + \frac{1}{1 + y^2} + \Phi, \quad (5.1.7)$$

with $x \in [0, 1]$, and initial condition,

$$y(0, x) = x(1 - x), \quad (5.1.8)$$

subject to homogeneous Dirichlet boundary conditions. The function Φ is chosen such that the exact solution is $y = x(1 - x)e^t$. The problem is discretised in space with $n = 200$, $\Delta x = \frac{1}{n}$,

$$\begin{aligned} y'_i &= Ay + \frac{1}{1 + y_i^2} + g(t) \\ y_i(0) &= \frac{x_i}{n} \left(1 - \frac{x_i}{n}\right) \end{aligned} \quad (5.1.9)$$

where $1 \leq i \leq n-1$, and we integrate from $t = 0$ to $t = 1$. We use the implementation provided by the EXPINT Matlab package [4].

5.1.5 The Reaction Diffusion Advection (RDA) 2D Equation

The 2D RDA is another stiff problem. The L matrix is a pent-diagonal, finite differences, matrix. The equation describes reaction-diffusion-advection

$$u_t = \varepsilon (u_{xx} + u_{yy}) - \alpha (u_x + u_y) + \rho u \left(u - \frac{1}{2}\right) (1 - u) \quad (5.1.10)$$

on the unit square $\Omega = [0, 1]^2$, subject to homogeneous Neumann boundary conditions with the initial condition

$$u(t = 0, x, y) = 0.3 + 256 (x(1-x)y(1-y))^2. \quad (5.1.11)$$

This problem was presented by Caliarì & Ostermann [10]. We will run our experiments for similar combinations of parameters as used by Caliarì & Ostermann.

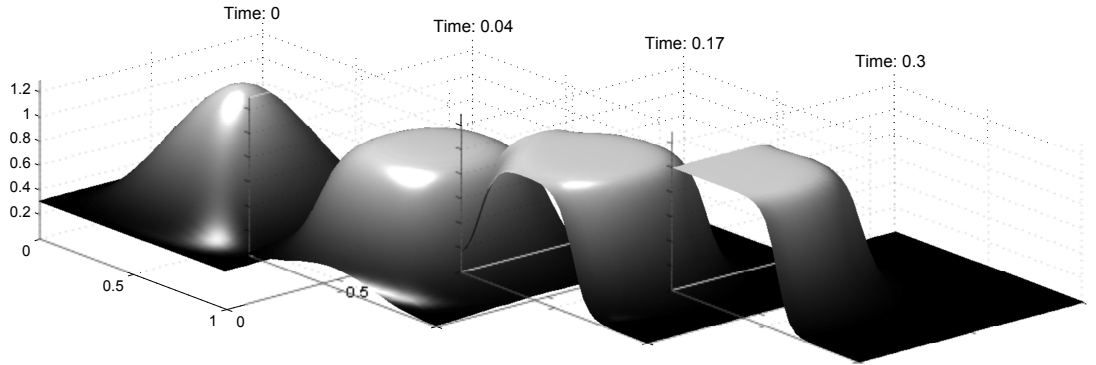


Figure 5.4: Rendering of 2D RDA Equation with $\varepsilon = 0.05$, $\alpha = -1$, $\rho = 100$

Being a 2D problem, the L matrix can be very large, even for moderately coarse mesh discretisation, and depending on the integrator, can have excessive memory requirements.

5.1.6 The Gray-Scott Equation

The Gray-Scott equation represents a process known as cubic autocatalysis, and is given by the equations

$$\begin{cases} \frac{k_f}{\rightarrow} A + 2B \xrightarrow{k_1} 3B, & r = k_1 a b^2 \\ B \xrightarrow{k_2} C, & r = k_2 b \end{cases} \quad (5.1.12)$$

where k_f, k_1 and k_2 are positive rate constants [22]. The model, in non-dimensional form, leads to the following system of reaction-diffusion equations.

$$u_t = D_u u_{xx} - k_1 u v^2 + k_f (1 - u), \quad (5.1.13)$$

$$v_t = D_v v_{xx} + k_1 u v^2 - k_2 v \quad (5.1.14)$$

for the concentrations $u(t, x)$ and $v(t, x)$ of U and V respectively. Here $k_f = \alpha$, $k_1 = 1$ and $k_2 = \alpha + \beta$. D_u and D_v are the diffusion coefficients of the chemicals U and V , and are usually chosen so that the ratio $\frac{D_u}{D_v} = 2$ [4].

The problem is solved on $[0, L]^d$, where d is the space dimension and $L = 2.5$, subject to homogeneous Neumann boundary conditions. For the 1D problem, the initial conditions, with $f = -150$, are,

$$u(0, x) = 1 - \frac{1}{2} e^{f \left(x - \frac{L}{2}\right)^2}$$

$$v(0, x) = \frac{1}{2} e^{f \left(x - \frac{L}{2}\right)^2}$$

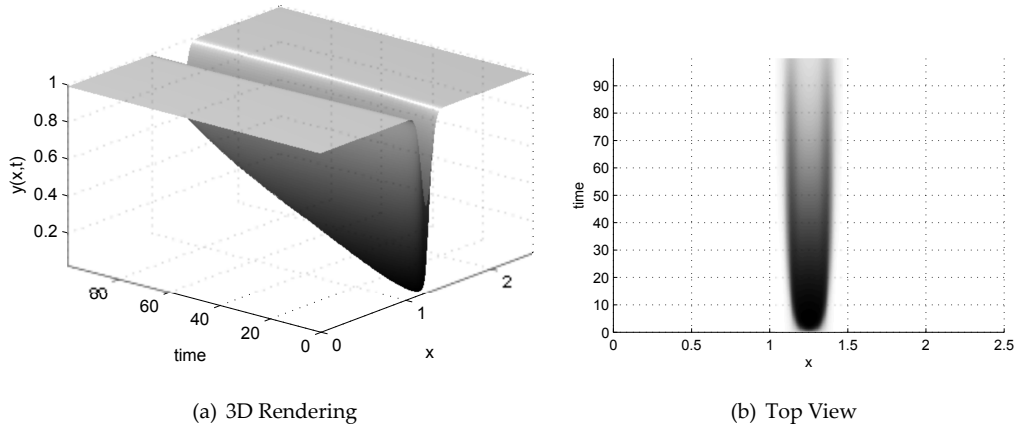


Figure 5.5: The 1D Gray-Scott Equation (5.1.13) for $u(t, x)$

The initial conditions of the 2D problem are

$$u(0, x, y) = 1 - e^{f \left(x - \frac{L}{2}\right)^2 + \left(y - \frac{L}{2}\right)^2}$$

$$v(0, x, y) = e^{f \left(x - \frac{L}{2}\right)^2 + 2 \left(y - \frac{L}{2}\right)^2}$$

and those of the 3D problem are

$$v(0, x, y, z) = e^{f(x-L)^2 + 10(y-L)^2 + 10(z-L)^2}$$

$$u(0, x, y, z) = 1 - v_0.$$

5.2 Computing φ -functions

Problems arise in implementing Exponential Integrators when some of the eigenvalues of the matrix, A , are close to zero. The first of two of the complications that occur are rounding errors introduced into the calculation due to cancellation among the scheme coefficients. Secondly it becomes impractical to use the explicit form of the φ -functions as the matrix A is, or is close to being, singular.

In addition, calculating the exponential of a matrix is not an easy task, computationally, in its own right [36]. If the matrix exponential is computed explicitly then the resultant matrix will typically not retain any of the sparse properties of the original A , [36] making this approach unsuitable for very large matrices due to excessive storage requirements.

To tackle these difficulties a number of approaches have been developed that work with the application of the matrix exponential on a vector, $e^A \times v$, without generating e^A explicitly. This technique lends itself nicely to the large sparse matrices we expect to get from spatial discretisations of PDE's.

5.2.1 Padé approximations

Simply computing the φ -functions directly is not practical when the norm, $\|A\|$, is small, as the results will be compromised due to cancellation errors. Alternatively, one could use the Taylor expansion

$$\varphi_1(A) = 1 + \frac{A}{2} + \frac{A^2}{3!} + \frac{A^3}{4!} + \dots \quad (5.2.1)$$

and this will work well for small $\|A\|$. However when the eigenvalues of A are large, this approach proves too inaccurate. Cox & Matthews proposed using a cut-off in terms of the eigenvalues, to switch between the two methods. They noted however that this is not always practical, in particular both methods suffer from inaccuracies around the switch-over point [13].

A well known approach when concerned with just computing the matrix exponential combines Padé approximations with repeated scaling and squaring [35]. Here the (p, q) Padé approximation to e^A is given by

$$R_{pq}(A) = [D_{pq}(A)]^{-1} N_{pq}(A),$$

where

$$N_{pq}(A) = \sum_{i=0}^p \frac{(p+q-j)!p!}{(p+q)!j!(p-j)!} L^i,$$

$$D_{pq}(A) = \sum_{j=0}^q \frac{(p+q-j)!p!}{(p+q)!j!(p-j)!} (-A)^j$$

This Padé approximation alone can only be used if $\|A\|$ is sufficiently small, if this is not the case the technique becomes expensive and loses accuracy [31]. These difficulties can be controlled however by exploiting a fundamental property of the exponential function

$$e^A = \left(e^{\frac{A}{m}}\right)^m$$

The idea is to chose m to be a power of 2 for which $e^{\frac{A}{m}}$ can be efficiently computed. The matrix $(e^{A/m})^m$ can then be calculated by repeated squaring. If m is chosen to be the smallest power of 2 such that $\frac{\|A\|}{m} \leq 1$ then $e^{\frac{A}{m}}$ can be computed with Padé approximations.

Expanding on this method Hochbruck, Lubich & Selhofer [24] applied this technique to the 1st φ -function by exploiting the property that

$$\varphi_1(2x) = \frac{(e^x + 1) \varphi_1(x)}{2}$$

For the higher φ -functions, φ_k when $k > 1$, reversing the scaling becomes more difficult but is still possible. The implementation within the EXPINT package makes use of the relations

$$\varphi_{2k}(2A) = \frac{1}{2^{2k}} \left[\varphi_k(A) \varphi_k(L) + \sum_{j=k+1}^{2l} \frac{2}{(2k-j)!} \varphi_k(A) \right], \quad (5.2.2)$$

$$\varphi_{2k+1}(2A) = \frac{1}{2^{2k+1}} \left[\varphi_k(A) \varphi_{k+1}(A) + \sum_{j=k+2}^{2k+1} \frac{2}{(2k+1-j)!} \varphi_k(A) + \frac{1}{j!} \varphi_{k+1}(A) \right] \quad (5.2.3)$$

to undo the scaling [4]. Unfortunately, as can be seen in Table 5.1, by using this approach the work needed to compute φ_k increases as k increases.

Fn.	3-point 1D FD Matrix, 512×512	5-point 2D FD Matrix, 1024×1024
φ_1	56.2	15.7
φ_2	84.3	25.3
φ_3	112.7	35
φ_4	142.1	44.9

Table 5.1: Padé timings in seconds using PHIPADE from the EXPINT package, $h = 1 \times 10^{-2}$

5.2.2 Krylov Subspace Methods

Krylov subspace methods for approximating the action of a matrix exponential on a vector have been around for a long time and are seen as a very promising approach. In the late eighties and early nineties a number of authors pioneered this technique, notably Friesner, Tuckerman, Dornblaser & Russo [17], and Gallopoulos & Saad [18].

EXPOKIT Krylov Implementation

The popular software package, EXPOKIT [44], from Sidje provides codes in Fortran and Matlab source for performing this calculation, for both the operations of the matrix exponential and that of the first φ -function, upon a vector using the Krylov subspace approach.

The idea behind Krylov methods is to approximate the vector $\varphi_k(A)v$, which resides in the large space \mathbb{R}^n , within the smaller space \mathbb{R}^m . This Krylov space,

$$K_m = \text{span} \{v, Av, A^2v, A^3v, \dots, A^{m-1}v\} \quad (5.2.4)$$

is the smaller subspace we wish to work with, but unfortunately the vectors A^jv form a bad basis as they all point in nearly the same direction of the dominant eigenvector.

The solution is to use the Gram-Schmidt procedure to generate an orthonormal basis of the Krylov space

$$K_m = \text{span} \{v_1, v_2, \dots, v_m\} \quad (5.2.5)$$

Now taking V_m to be the $n \times m$ matrix whose columns are v_1, v_2, \dots, v_m then the $m \times m$ matrix $H_m = V_m^T A V_m$ is the projection of the action of A on the Krylov subspace. Note that the matrix H_m is Hessenberg. Also if the matrix A is symmetric then H_m is both symmetric and Hessenberg, meaning that it is tridiagonal.

The next step is to approximate $\varphi_k(A)v$ by $\varphi_k(V_m H_m V_m^T)v$. Since we know that $V_m^T V_m = I_m$ and $V_m V_m^T v = v$ we have

$$\varphi_k(V_m H_m V_m^T)v = V_m \varphi_k(H_m) V_m^T v \quad (5.2.6)$$

Additionally we note that $V_m^T v = \beta e_1$, where $\beta = \|v\|$ and e_1 is the first standard basis vector, which leads us finally to the approximation

$$\begin{aligned} \varphi_k(A)v &\approx \varphi_k(V_m H_m V_m^T)v \\ &\approx \beta V_m \varphi_k(H_m) e_1 \end{aligned} \quad (5.2.7)$$

For example a 1024×1024 matrix from a standard 3-point finite-differences discretisation,

$$A = 1025^2 \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix}$$

For A the EXPOKIT Krylov code constructs the following $m \times m$ projection matrix of the action of A onto the Krylov subspace.

$$H_m = \begin{pmatrix} -0.001 & 0.0004 & & & & & 1 & 0 & 0 \\ 0.0004 & -1.23 & 11.33 & & & & 0 & 0 & 0 \\ & 11.33 & -2\alpha & \alpha & & & & & \\ & & \alpha & -2\alpha & \alpha & & & & \\ & & & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\ & & & & \alpha & -2\alpha & \alpha & & \\ & & & & & \alpha & -2\alpha & 0 & 0 & 0 \\ & & & & & & 0 & 0 & 1 & 0 \\ & & & & & & & 0 & 0 & 1 \\ & & & & & & & & 0 & 0 \end{pmatrix}, \quad \alpha = \frac{1025^2}{10000}$$

where m , the user specified subspace dimension, is in this case 45.

PHIPM Krylov Implementation

The EXPOKIT implementation is limited by its use of a fixed Krylov subspace dimension and by the fact that it can only be used to compute φ_1 . This means that the user is required to be able to estimate the optimal subspace dimension prior to using the methods, in addition the lack of support for higher φ -function restricts its use to the 1st order ETD scheme.

To tackle the first limitation Hochbruck, Lubich & Selhofer [24] proposed an approach to dynamically adapt the size of the subspace. In a recent paper, Niesen & Wright developed a Krylov method along with accompanying Matlab code for calculating e^A and the higher order φ -functions [37]. This implementation, which we refer to as PHIPM, addresses a number of the shortcomings of the EXPOKIT implementation.

Their solver combines time-stepping ideas [44] with the approach proposed in [24] for adapting the subspace dimension. Crucially the PHIPM code works on a full linear combina-

tion of φ -functions upon input vectors v_0, \dots, v_p ,

$$e^A v_0 + \varphi_1(A) v_1 + \varphi_2(A) v_2 + \dots + \varphi_p(A) v_p \quad (5.2.8)$$

during a single call to the code. This allows it to compute the result of an entire stage of an exponential scheme very efficiently. Table 5.2 presents some computational timings for two common problem matrices. The final row, "All", refers to the calculation of a full linear combination up to φ_4 ,

$$e^{hL} v_0 + h\varphi_1(hL) v_1 + h^2\varphi_2(hL) v_2 + h^3\varphi_3(hL) v_3 + h^4\varphi_4(hL) v_4 \quad (5.2.9)$$

(a) 3-point 1D FD Matrix, 512×512			(b) 5-point 2D FD Matrix, 1024×1024		
Fn.	Error	Timing	Fn.	Error	Timing
$h\varphi_1$	2.4026×10^{-15}	0.108	$h\varphi_1$	1.9255×10^{-16}	0.004
$h^2\varphi_2$	4.248×10^{-17}	0.094	$h^2\varphi_2$	5.4287×10^{-16}	0.007
$h^3\varphi_3$	9.7932×10^{-17}	0.103	$h^3\varphi_3$	1.1915×10^{-15}	0.003
$h^4\varphi_4$	1.0567×10^{-16}	0.071	$h^4\varphi_4$	1.9057×10^{-15}	0.006
All	1.0021×10^{-12}	0.118	All	1.6653×10^{-13}	0.004

Table 5.2: PHIPM at tolerance 1×10^{-13} with $h = 1 \times 10^{-2}$

5.2.3 Real Leja Points Method

Krylov subspace methods are a class of polynomial methods. An alternative class of polynomial methods is based on direct interpolation or approximation of the corresponding scalar analytic function on the spectrum of the relevant matrix.

To compute the matrix exponential through the use of polynomial methods, denote the characteristic polynomial of A by $c(z)$ where

$$c(z) = \det(zI - A) = z^n - \sum_{k=0}^{n-1} c_k z^k$$

From the Cayley-Hamilton theorem, which states that every square matrix is annihilated by its characteristic polynomial, we can say that $c(A) = 0$ and hence that

$$A^n = c_0 I + c_1 A + \dots + c_{n-1} A^{n-1}$$

It follows that any power of A can be expressed in terms of I, A, \dots, A^{n-1}

$$A^k = \sum_{j=0}^{n-1} \beta_{kj} A^j$$

This implies that e^{tA} is a polynomial in A with analytic coefficients in t

$$\begin{aligned} e^{tL} &= \sum_{k=0}^{\infty} \frac{t^k A^k}{k!} = \sum_{k=0}^{\infty} \frac{t^k}{k!} \left[\sum_{j=0}^{n-1} \beta_{kj} A^j \right] \\ &= \sum_{j=0}^{n-1} \left[\sum_{k=0}^{\infty} \beta_{kj} \frac{t^k}{k!} \right] A^j \end{aligned}$$

We are then required to generate the coefficients β_{kj} . Alternatively, if we can generate another set of matrices $\{A_0, \dots, A_{n-1}\}$ which span the same subspace as I, A, \dots, A^{n-1} then the analytic functions β_j exist such that

$$e^{tA} = \sum_{j=0}^{n-1} \beta_j(t) A_j \quad (5.2.10)$$

which may be easier to generate.

Bergamaschi & Vianello compared Chebyshev series expansions against the then existing Krylov based methods for the matrix exponential of large, sparse, symmetric matrices. They noted that the Chebyshev series approach represented a viable alternative to Krylov methods highlighting in particular the inherent simplicity of Chebyshev, which opens the door to efficient implementations and optimising storage requirements [3].

Later Bergamaschi, Vianello & Caliarì looked at applying polynomial methods to the φ -functions. They proposed and analysed the Real Leja Points Method (ReLPM) approach which uses pseudo-spectral estimates via families of confocal ellipses.

The ReLPM code estimates the spectral focal interval by Arnoldi approximation or Gershgorin's theorem, either of which will provide a polynomial approximation with similar behaviour [11]. The polynomial is then interpolated via Newton interpolation, not at uniform points but rather on real Leja sequences of the corresponding focal intervals. This guarantees maximal, and therefore, superlinear convergence [2].

The three authors together with Martínez, then compared the ReLPM method against the Krylov implementation from EXPKIT and showed superior performance in all tests [2].

Caliari & Ostermann employed the ReLPM method in Rosenbrock-type integrators up to order 4. They presented numerical tests in both Matlab and FORTRAN demonstrating excellent performances [10]. The Matlab implementation tested is the one we used in our own numerical experiments. It is suitable for computing $\varphi_k(L)v$ products for $k = 1, 2, 3, 4$.

(a) 3-point 1D FD Matrix, 512×512			(b) 5-point 2D FD Matrix, 1024×1024		
Fn.	Error	Timing	Fn.	Error	Timing
$h\varphi_1$	1.3807×10^{-13}	0.138	$h\varphi_1$	3.4694×10^{-17}	0.011
$h^2\varphi_2$	6.0957×10^{-14}	0.146	$h^2\varphi_2$	1.0156×10^{-16}	0.012
$h^3\varphi_3$	1.7204×10^{-15}	0.154	$h^3\varphi_3$	7.0255×10^{-17}	0.014
$h^4\varphi_4$	4.3039×10^{-15}	0.161	$h^4\varphi_4$	1.6887×10^{-15}	0.015
All	1.9003×10^{-12}	0.609	All	4.5519×10^{-14}	0.05

Table 5.3: ReLPM at tolerance 1×10^{-13} with $h = 1 \times 10^{-2}$

5.2.4 Contour Integration

Polynomial based methods are not the only techniques being developed. One alternative approach is concerned with using ideas from complex analysis. Kassam & Trefethen looked at evaluating

$$\varphi_k(z) = \frac{1}{2\pi i} \int_{\Gamma} \frac{\varphi_k(s)}{s-z} ds \quad (5.2.11)$$

over a contour Γ in the complex plane [26].

The contours they worked with were circles of radius 1 centered on z far away from the origin. In the matrix case the contour must enclose the eigenvalues of z . The technique can be generalised to the non-diagonal matrices

$$\varphi_k(z) = \frac{1}{2\pi i} \int_{\Gamma} \varphi_k(z) (sI - A)^{-1} ds \quad (5.2.12)$$

However the amount of computational work becomes greatly increased as a number of matrix inverses must be calculated.

Their approach can also be extended to the more desirable action of the φ -functions upon a vector by evaluating

$$\varphi_k(z) b = \frac{1}{2\pi i} \int_{\Gamma} \varphi_k(z) (sI - A)^{-1} b ds \quad (5.2.13)$$

The matrix inverses are still needed for non-diagonal matrices.

Kassam & Trefethen concentrated on a constant time stepping exponential integrator where the φ -functions were explicitly evaluated. As such, the computationally expensive work only needed to be done once prior to the main time-stepping loop. Crucially, this approach does not extend to the variable-step case, where the matrix L can change at each time step.

The circular contours used by Kassam & Trefethen are not the only types which have been looked at. For non-diagonal matrices, alternative contours which enclose the spectrum of the

matrix are necessary. Here, it is important to note that, in the case of the matrix exponential function, the integral

$$f(t) = \frac{1}{2\pi i} \int_{\Gamma} e^{zt} F(z) dz \quad (5.2.14)$$

where

$$F(z) = (zI - A)^{-1} f_0$$

is also that of the inverse Laplace transform

It is known as the Bromwich integral where Γ , the contour of integration is initially the Bromwich line $\text{Re}(z) = \sigma$. The parameter σ should be large enough such that all the eigenvalues of A lie in the half-plane $\text{Re}(z) \leq \sigma$ [46].

As it is, (5.2.14) is not suitable for numerical integration. One issue here is that the exponential factor is highly oscillatory on the Bromwich line. Secondly $F(z)$ typically decays slowly.

Originally, Talbot suggested solving these issues by deforming the Bromwich line into a contour and presented an approach based on a cotangent contour which can be expressed as

$$\gamma : z(\theta) = \sigma + \mu(\theta \cot \theta + vi\theta), \quad -1\pi \leq \theta \leq \pi \quad (5.2.15)$$

Talbot also devised a numerical method for the inversion using trapezoidal and midpoint rules [45].

Weideman [46] optimised Talbot's method by finding near to optimal parameters to define a closely related contour

$$\Gamma : z(\theta) = \sigma + \mu \left(1 + \frac{2\theta^2}{\theta^2 - \pi^2} + vi\theta \right), \quad -1\pi \leq \theta \leq \pi \quad (5.2.16)$$

This contour is mentioned in Talbot's original paper [45] and is easier to analyse, though it is noted that Talbot's contour offers superior accuracy.

Later, Trefethen & Weideman looked at two alternative contours; a parabolic and a hyperbola, and they determined the optimal parameters to define these contours [47]. The convergence achieved with these near optimal parameters for the cotangent, parabolic and hyperbola contours are $O(2.85^{-N})$, $O(3.20^{-N})$ and $O(3.89^{-N})$ respectively for the trapezoid rule.

Rational Approximations

In [30] Ya Yan Lu proposed the use of uniform rational Chebyshev approximations. For the matrix exponential case, the method computes the largest eigenvalue of T , λ_1 , and then approximates e^T by a Chebyshev rational approximation of $e^{T-\lambda_1 I}$, seeing as

$$e^T = e^{\lambda_1} e^{T-\lambda_1 I}$$

and then calculating e^A by

$$e^A = Qe^T Q^T \quad (5.2.17)$$

Lu extends this approach in [31] to computing the φ functions, though still only for symmetric matrices.

Schmelzer, Trefethen & Weideman investigated the use of this technique to optimise the more general case of arbitrarily shaped contours. They claim that, when using rational approximations, a convergence rate approximately twice as fast, $O(9.28903^{-N})$, can be achieved. Some numerical results to support that claim can be seen in Table 5.4, which compares the presented Matlab code for contour integration [41, Figure 3.2] against code for the method of best rational approximations [41, Figure 4.4].

Matrix Dimension	Contour Integration	Rational Approximations
2401×2401	0.48	0.23
9801×9801	2.35	1.04
39601×39601	12.79	5.66

Table 5.4: Computational timings for $e^L v$ in seconds

However, there is a caveat; the rational approximations, though faster in a number of situations, are noted to be sensitive to small changes in the problem type or parameter [41].

A weakness identified by Lu is that, for best rational approximations the coefficients are hard to compute. However Schmelzer, Trefethen & Weideman utilise the Carathéodory-Fejér method to produce approximate functions to the true best rational approximations. The approximate functions are accurate enough that they can be considered exact in practise. They present, in that paper, a Matlab function for computing e^A .

Schmelzer & Trefethen adapted this Matlab code to compute the φ_k functions making the approach applicable to Exponential Integrators [42]. As it is common within the Exponential Integrators to require the matrix vector product for several φ -functions they suggest the reuse of common poles for $\varphi_1, \dots, \varphi_k$. Using that approach the number of LU decompositions can be reduced by a factor of 2.

They make use of a recurrence relation

$$r_{(l+k)}(z) = \sum_{j=1}^n \frac{c_j z_j^{-1}}{z - z_j}, \quad k \in \mathbb{Z} \quad (5.2.18)$$

for rational approximation using common poles. Though not optimal, the common poles are proved to be sufficiently so on the negative real axis. In numerical experiments, they note an actual typical increase of efficiency by a factor of 2 to 3.5 [42].

Nonetheless, it must be emphasised that in variable step integration, for non-diagonal matrices, a number of matrix inverses must be computed at each time-step. In our numerical experiments one can see that this is a crucial weakness of the approach and for a number of problems it is simply inappropriate to use.

(a) 3-point 1D FD Matrix, 512×512			(b) 5-point 2D FD Matrix, 1024×1024		
Fn.	Error	Timing	Fn.	Error	Timing
$h\varphi_1$	9.4039×10^{-15}	0.003	$h\varphi_1$	1.1345×10^{-15}	0.101
$h^2\varphi_2$	6.1826×10^{-15}	0.001	$h^2\varphi_2$	9.5589×10^{-15}	0.045
$h^3\varphi_3$	7.7083×10^{-16}	0.001	$h^3\varphi_3$	1.1189×10^{-15}	0.033
$h^4\varphi_4$	1.9599×10^{-16}	0.001	$h^4\varphi_4$	2.8185×10^{-16}	0.022
All	1.6560×10^{-12}	0.007	All	9.9032×10^{-14}	0.201

Table 5.5: Rational Approximations with $h = 1 \times 10^{-2}$

5.2.5 Conclusions

Having put the different approaches to φ -functions computations through a number of numerical tests we can begin to draw some concrete conclusions on their relative performances. The crucial observation which comes from Table 5.1, is that attempting to compute the matrix exponential or φ -functions explicitly is not at all practical and this supports the existing literature on the subject. In addition to the excessive CPU time costs there is the issue of memory usage and indeed the 1024×1024 pent-diagonal matrix from the 2D RDA problem was at the limit of what our desktop PC could work with.

Clearly the way forward is to follow the recommended approach of working with the operation of the φ -functions upon a vector. Tables 5.2, 5.3 and 5.5 look at three competing proposals for this approach. The PHIPM code proved to be the most robust, showing very consistent performance. It also introduced the novel approach of working on a full linear combination of φ -functions upon an input vector. This could possibly be applied to the two other techniques which would put their experiments for the final, linear combination, entries of Tables 5.3 and 5.5 on a more level playing field. Such a modification however is beyond the scope of our investigation.

The conclusion which we can draw about the ReLPM package is that it shows superior performance over that of EXPKIT and of the Rational Approximations approach. However it does not compete as well against PHIPM. The Rational Approximations tests show very good, indeed, excellent performances. They are, however a bit deceptive as the tabulated timings are

restricted to the moderately-sized matrices which we were limited to in testing against Padé results. This hides the significant slowdown experienced as the matrix sizes grow.

For the final conclusions we will present comparison plots for four different L matrices, each representative of a particular type of PDE discretisation.

- Figure 5.6 tests a real tri-diagonal matrix. This matrix is the result of a finite difference discretisation of the 1D Hochbruck & Ostermann Parabolic PDE (5.1.9).
- Figure 5.7 tests a real pent-diagonal matrix. This matrix is the result of a finite difference discretisation of the 2D RDA Equation (5.1.10).
- Figure 5.8 tests a complex diagonal matrix. This matrix is the result of a Fourier spectral discretisation of the Kuramoto-Sivashinsky Equation (5.1.1).
- Figure 5.9 tests a dense real matrix. This matrix is a Chebyshev differentiation matrix from the Allen-Cahn equation (5.1.6).

From these plots, we can clearly identify the relative rankings between the approaches, which demonstrate PHIPM to be the most efficient followed closely by the ReLPM implementation.

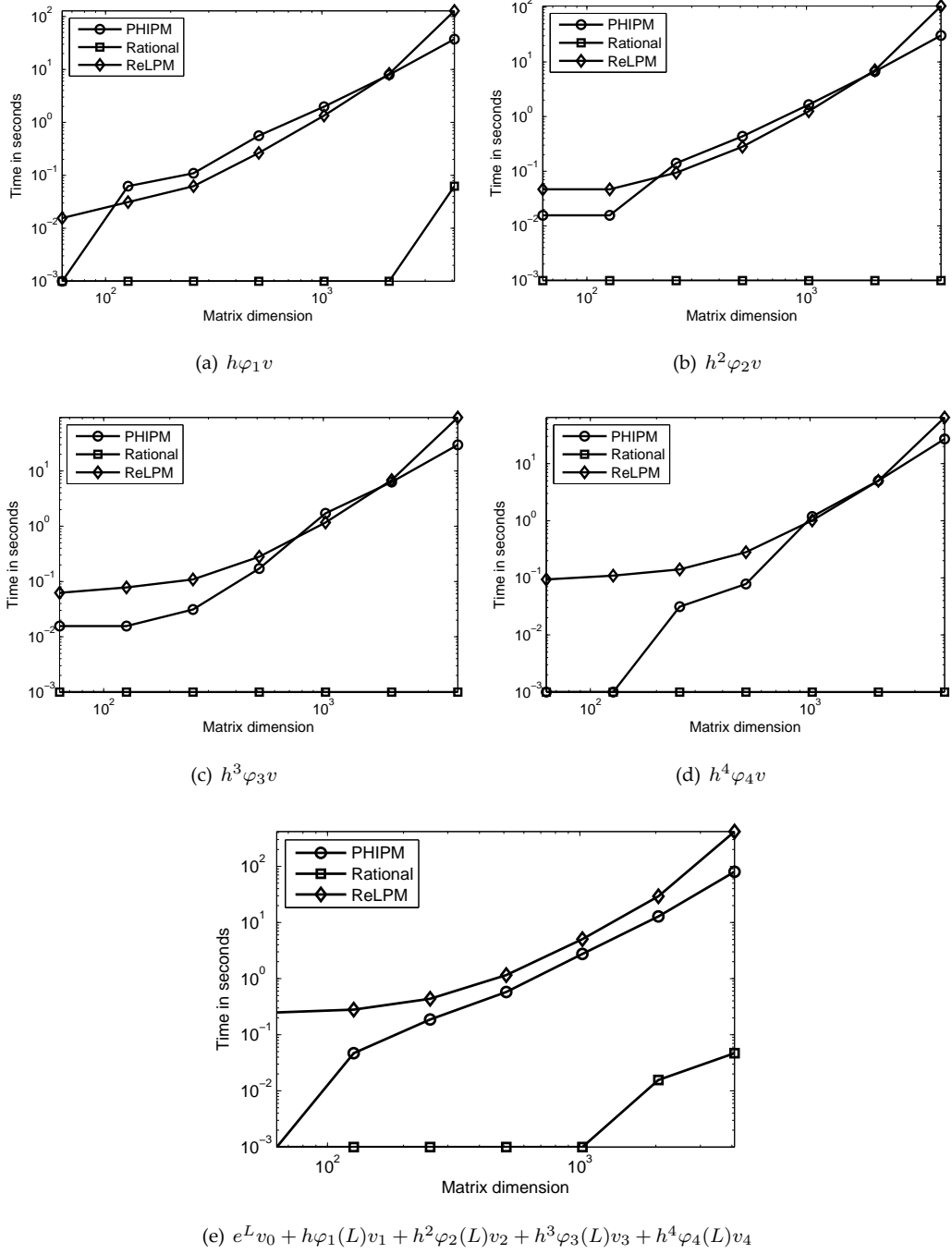
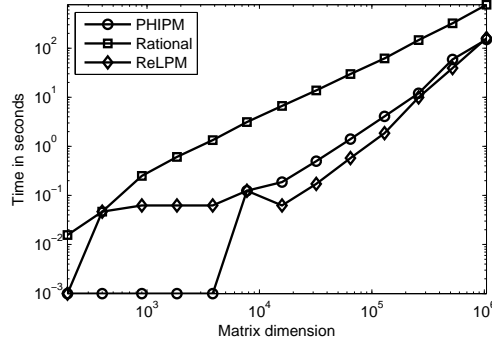
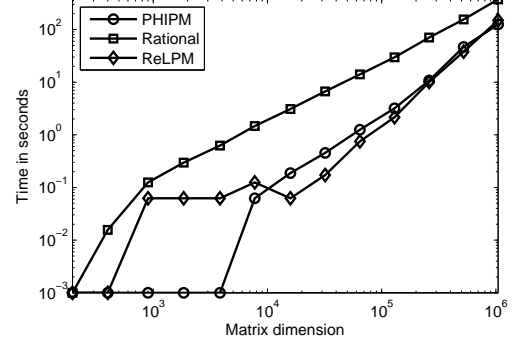


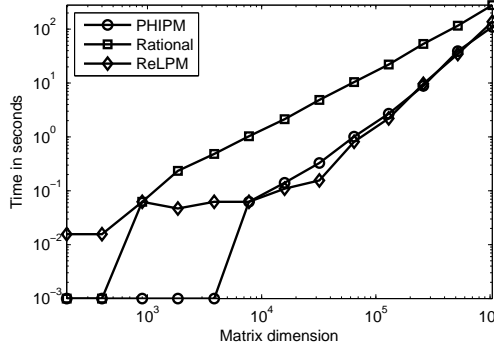
Figure 5.6: φ Timings for the Hochbruck & Ostermann Parabolic PDE, $h = 0.001$



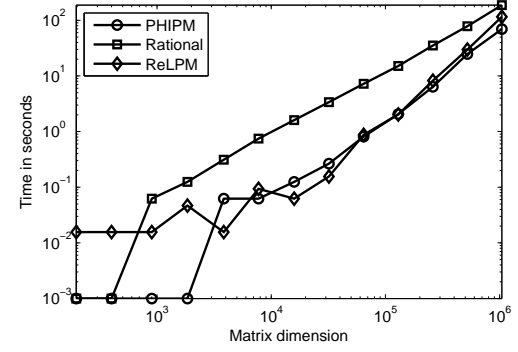
(a) $h\varphi_1 v$



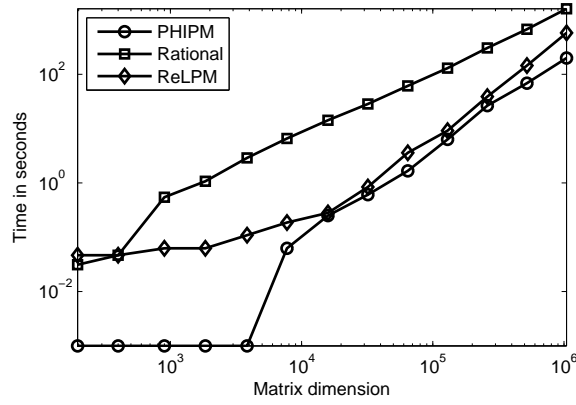
(b) $h^2\varphi_2 v$



(c) $h^3\varphi_3 v$



(d) $h^4\varphi_4 v$



(e) $e^L v_0 + h\varphi_1(L)v_1 + h^2\varphi_2(L)v_2 + h^3\varphi_3(L)v_3 + h^4\varphi_4(L)v_4$

Figure 5.7: φ Timings for the 2D RDA Equation, $h = 0.001$

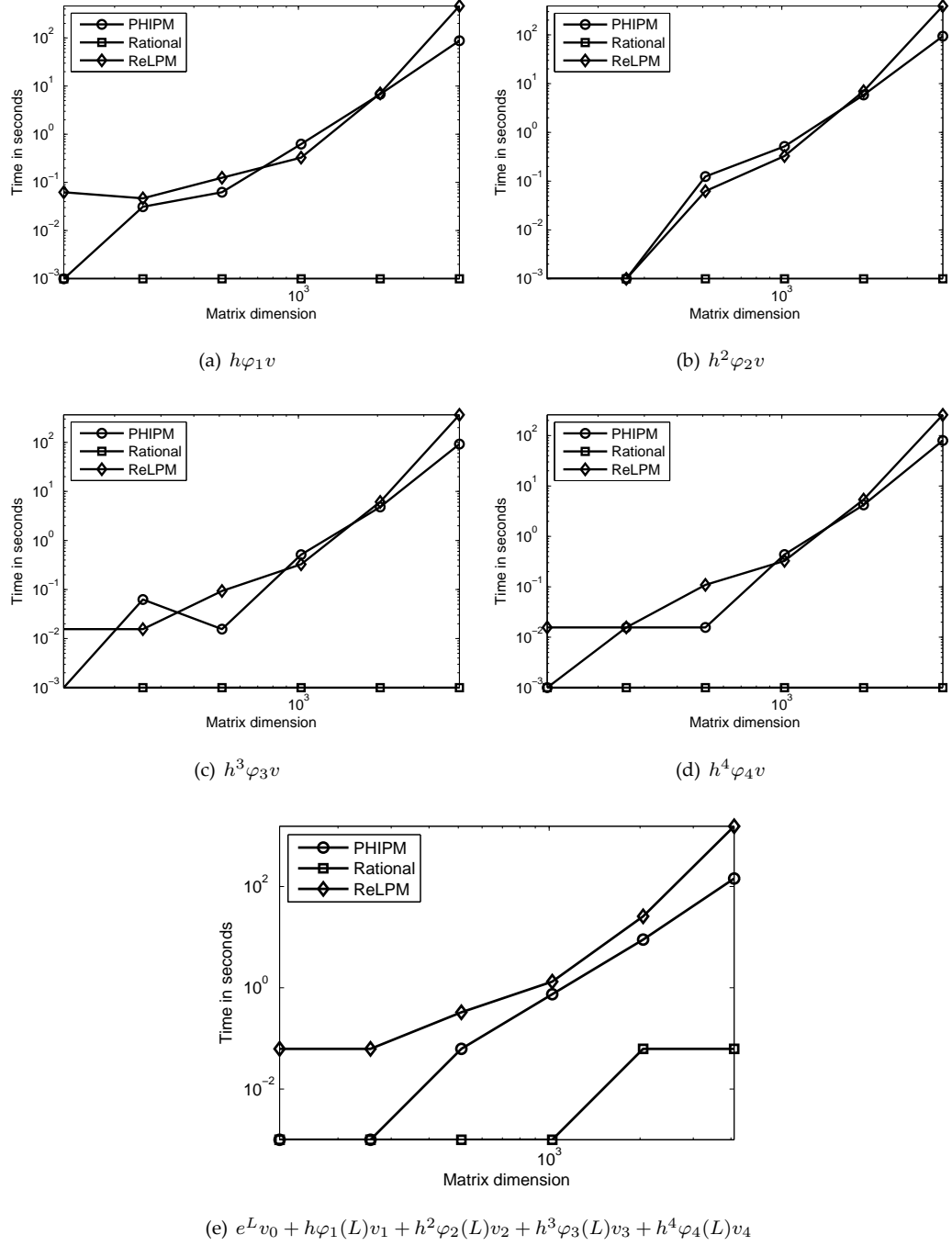
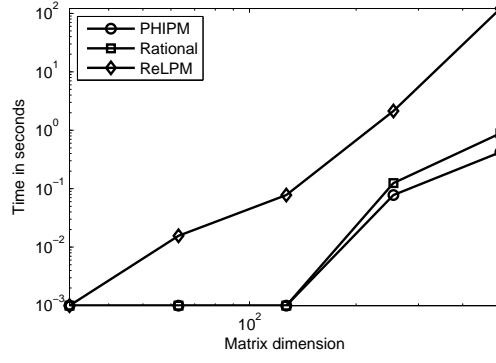
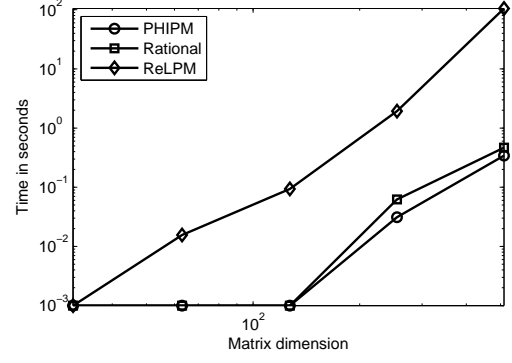


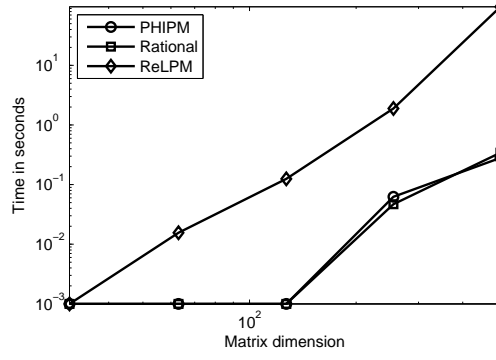
Figure 5.8: φ Timings for the Kuramoto-Sivashinsky Equation, $h = 0.001$



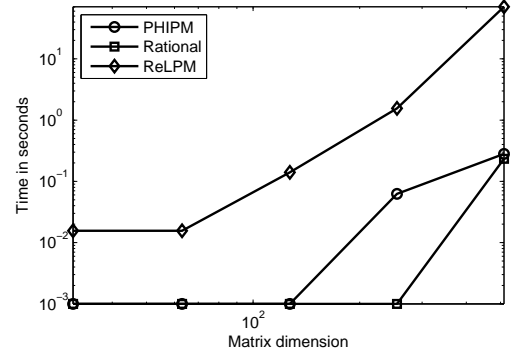
(a) $h\varphi_1 v$



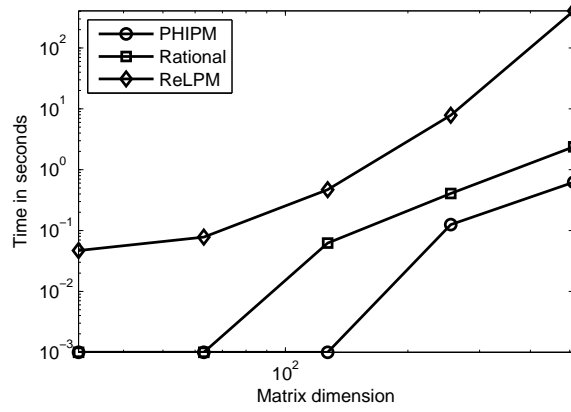
(b) $h^2\varphi_2 v$



(c) $h^3\varphi_3 v$



(d) $h^4\varphi_4 v$



(e) $e^L v_0 + h\varphi_1(L)v_1 + h^2\varphi_2(L)v_2 + h^3\varphi_3(L)v_3 + h^4\varphi_4(L)v_4$

Figure 5.9: φ Timings for the Allen-Cahn Problem, $h = 0.001$

5.3 Convergence Order

It is important to verify the analytically derived order conditions for the various EI families. To do this, we run a number of fixed stepsize experiments against the 1D Hochbruck & Ostermann Parabolic PDE (5.1.9), a test problem indicative of the type of real world problem that we are interested in. By comparing schemes over the same number of time steps and stepsizes, we can get a very reliable, relative, measure of their accuracy and convergence properties. In all of these experiments, we will include the ETD Euler (1.3.13) scheme which will serve to represent a baseline, 2nd order solution.

5.3.1 Two-Stage ERKs

We return to the two families of 2-stage ERKs introduced in Section 2.1.3, which we saw were also presented in [25]. These families are:

- ERK₂ (2.1.19)

0		I
c_2	$c_2\varphi_{1,2}$	e^{c_2hL}
	$\varphi_1 - \frac{1}{c_2}\varphi_2 \quad \frac{1}{c_2}\varphi_2$	

represents a one-parameter family of 2nd order ERKs. Schemes constructed from this family require the computation of two φ -functions for $c_2 = 1$, and three when $c_2 \neq 1$.

- ERK₂ (2.1.21)

0		I
c_2	$c_2\varphi_{1,2}$	e^{c_2hL}
	$(1 - \frac{1}{2c_2})\varphi_1 \quad \frac{1}{2c_2}\varphi_1$	

is less computationally expensive as it does not make use of φ_2 . However, the schemes derived for this family do not achieve full 2nd order.

Both of these methods were presented earlier by Strehmel & Weiner who showed them to be B-consistent of order 1. B-consistency, together with B-stability, is necessary for B-convergence, which is a convergence property based on the one-side Lipschitz condition

$$\langle f(t, y_1) - f(t, y_2), y_1 - y_2 \rangle \leq \gamma \|y_1 - y_2\|^2. \quad (5.3.1)$$

Frank, Schneid & Ueberhuber introduced B-convergence, which permits the derivation of uniform global error bounds independent of the stiffness of the considered problem [16]. See Appendix A.2 for the definitions of these properties.

5.3.2 Three-Stage ERKs

Presented with the 2-stage ERKs in [25], are two families of 3-stage ERKs:

- ERK₃ [25, Scheme 5.8]

0			I	(5.3.2)
c_2	$c_2\varphi_{1,2}$		e^{c_2hL}	
$\frac{2}{3}$	$\frac{2}{3}\varphi_{1,3} - \frac{4}{9c_2}\varphi_{2,3}$	$\frac{4}{9c_2}\varphi_{2,3}$	$e^{\frac{2}{3}hL}$	
	$\varphi_1 - \frac{3}{2}\varphi_2$	0	$\frac{3}{2}\varphi_2$	

- ERK₃ [25, Scheme 5.9]

0			I	(5.3.3)
c_2	$c_2\varphi_{1,2}$		e^{c_2hL}	
c_3	$c_3\varphi_{1,3} - \alpha$	α	e^{c_3hL}	
	$\varphi_1 - \left(\frac{\gamma}{\beta} - \frac{1}{\beta}\right)\varphi_2$	$\frac{\gamma}{\beta}\varphi_2$	$\frac{1}{\beta}\varphi_2$	

$$\gamma = (3c_3^2 - 2c_3)/(2c_2 - 3c_2^2)$$

$$\alpha = \gamma c_2 \varphi_{2,2} + \frac{c_3^2}{c_2} \varphi_{2,3}$$

$$\beta = \gamma c_2 + c_3$$

Both of these families require five distinct φ -functions. Though (5.3.2) and (5.3.3) are families of 3-stage schemes, we saw in Section 2.1 that they only achieve weak 3rd order (Definition 4).

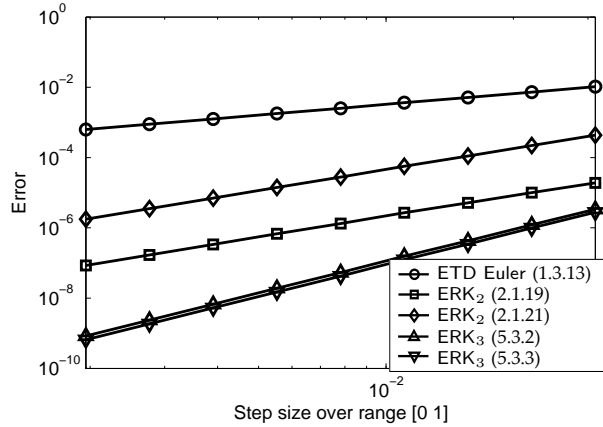


Figure 5.10: Fixed step integration of Problem 5.1.9

Figure 5.10 shows the fixed step-size results, plotting step-size against L_∞ -norm of error. It compares the two 2-stage (2.1.19, 2.1.21), and the two 3-stage (5.3.2, 5.3.3) schemes. The plot highlights clearly that ERK₂ (2.1.21) does not achieve the same degree of accuracy as (2.1.19).

We can also determine from the slopes of the plots that schemes ERK₃ (5.3.2) and (5.3.3) outperform the 2nd order schemes in terms of convergence order and accuracy. This highlights that weak 3rd order convergence still provides a performance advantage.

Computational Cost

What is not clear from this type of plot however is the cost per step differences between the various schemes. We could plot the schemes against CPU time rather than stepsize, to get a relative measure of the trade-off between accuracy and computational cost. In doing so, however, it is crucial to ensure that we are using the most optimal implementation for each scheme. Any repetitions within the scheme tableau can represent opportunities for optimisation, which a naive implementation will miss. Added to this is the complication that, for any given scheme, the most efficient procedure for computing a single step depends on the specific method used to evaluate the φ -functions.

The solution adopted here is to rate the CPU cost of each scheme in terms of a count of the number of the most expensive operations that need to be performed at each time step. We saw in Section 5.2 that the φ -functions themselves represent that expensive computational operation. To get a measure of the cost of each method we analysed them under each of the main categories for φ -function evaluation.

- **Explicit Evaluation.** This is the approach used by the EXPINT package, which itself relies on Padé approximations. When the φ -functions are computed explicitly, they can be reused for a number of matrix-vector products at negligible cost. As such, a count of the number of distinct matrix exponentials and φ -functions gives us a very good relative indicator of a schemes complexity.

We will take into account a key optimisation applicable to the majority of schemes. We utilise the identity

$$e^{c_i h L} \times y_n + h c_i \varphi_{1,i}(hL) \times v = y_n + h c_i \varphi_{1,i}(hL) \times (L y_n + v) \quad (5.3.4)$$

to save us from having to perform the matrix-exponential by a vector operation, ($\expm \times \text{vector}$) and this gives a significant performance boost.

- **$\varphi \times$ vector Operations.** This approach was taken by the EXP0KIT Krylov code as well as the ReLPM and Contour Integration techniques. Here there is a possible need to compute a $\varphi \times$ vector product with the same φ -function at two different stages within a step. Therefore, the number of distinct φ -functions represents only a lower bound on the estimated number of operations necessary to compute one step. In practice, each scheme must be

considered separately to determine the true, optimal, number of operations needed. A pseudo code example illustrates this procedure,

- 1: $K_2 \leftarrow N(t_n + c_2 h, e^{c_2 h L} \times y_n + h c_2 \varphi_{1,2} \times N_n)$
- 2: $K_3 \leftarrow N(t_n + \frac{2}{3} h, e^{\frac{2}{3} h L} \times y_n + h [\frac{2}{3} \varphi_{1,3} \times N_n + \frac{4}{9 c_2} \varphi_{1,3} \times (K_2 - N_n)])$
- 3: $y_{n+1} \leftarrow N(t_n + h, e^{h L} \times y_n + h [\frac{2}{3} \varphi_1 \times N_n + \varphi_2 \times (K_3 - \frac{3}{2} N_n)])$

This algorithm performs a single step for ERK₃ (5.3.2). There are a total of seven expensive matrix-vector operations. By incorporating the (5.3.4) optimisation, we can reduce this to five operations in total,

- 1: $K_2 \leftarrow N(t_n + c_2 h, y_n + h c_2 \varphi_{1,2} \times (Ly_n + N_n))$
- 2: $K_3 \leftarrow N(t_n + \frac{2}{3} h, y_n + \frac{2}{3} h \varphi_{1,3} \times (Ly_n + N_n) + h \frac{4}{9 c_2} \varphi_{1,3} \times (K_2 - N_n))$
- 3: $y_{n+1} \leftarrow N(t_n + h, y_n + h \varphi_1 \times (Ly_n + N_n) + h \varphi_2 \times (\frac{3}{2} K_3 - N_n))$

- **$\varphi \times$ vector Linear Combinations** Finally, this is the technique introduced by the PHIPM code, which computes a full linear combination of φ computations in one call to the code. This means each stage can be completed in a single operation. Using ERK₃ (5.3.2) again as an example, we present a pseudo code algorithm,

- 1: $M_1 \leftarrow \begin{pmatrix} y_n & N_n \end{pmatrix}$
- 2: $K_2 \leftarrow N(t_n + c_2 h, \text{phipm}(M_1))$
- 3: $M_2 \leftarrow \begin{pmatrix} y_n & N_n & -\frac{1}{9 c_2} N_2 + \frac{1}{9 c_2} K_2 \end{pmatrix}$
- 4: $K_3 \leftarrow N(t_n + c_2 h, \text{phipm}(M_2))$
- 5: $M_3 \leftarrow \begin{pmatrix} y_n & N_n & -\frac{3}{2} N_2 + \frac{3}{2} K_2 \end{pmatrix}$
- 6: $y_{n+1} \leftarrow \text{phipm}(M_3)$

The operations to construct the M_i matrices are computationally negligible in comparison to the `phipm` operation. Therefore, under this performance measure, we simply count the number of stages.

Table 5.6 summarises the computational cost analysis for the ETD Euler (1.3.13) and the four ERK schemes (2.1.19, 2.1.21, 5.3.2, 5.3.3), under each of the three categories of φ -function evaluation. ETD Euler is 1-stage and requires only one φ -function. It's computational cost therefore, is 1 under each of the categories and it serves as an excellent baseline. The cost should be interpreted as a relative measure of the CPU time needed to take a time step. For example, under the explicit evaluation of φ -functions, we would expect ERK₃ (5.3.2) to take 2.5 (ratio of distinct φ 's is 5:2) times longer to complete the same number of time steps as ERK₂ (2.1.19).

We can test this cost analysis experimentally by performing CPU timings. Table 5.7 shows the CPU timings, in seconds, for the 2-stage and 3-stage ERKs for a fixed stepsize experiment. Table 5.7 also shows the same experiment with the measurements normalised against the ETD Euler timings. Figure 5.11 plots, for those same schemes, an accuracy against CPU timing com-

Scheme		Explicit Evaluation	$\varphi \times \text{vector}$	Linear Combination
ETD Euler	(1.3.13)	1	1	1
ERK ₂	(2.1.19)	3	3	2
ERK ₂	(2.1.21)	2	2	2
ERK ₃	(5.3.2)	5	5	3
ERK ₃	(5.3.3)	6	6	4

Table 5.6: Relative performance measure for the 2 and 3-Stage ERKs

(a) CPU Timings (secs)					(b) Normalised CPU Timings		
Scheme		phipade	ReLPM	phipm	phipade	ReLPM	phipm
ETD Euler	(1.3.13)	254	6.3	4.7	1	1	1
ERK ₂	(2.1.19)	864	22.8	7.4	3.4	3.6	1.56
ERK ₂	(2.1.21)	482	11.9	8.5	1.9	1.88	1.77
ERK ₃	(5.3.2)	1405	39.2	10.4	5.5	6.18	2.2
ERK ₃	(5.3.3)	1635	49	10	6.4	7.72	2.1

Table 5.7: Timings for 256 Timesteps, with ETD Euler and the 2 & 3-Stage ERKs

parison. The implementation used to compute the φ -functions for the plot is ReLPM from Section 5.2.3.

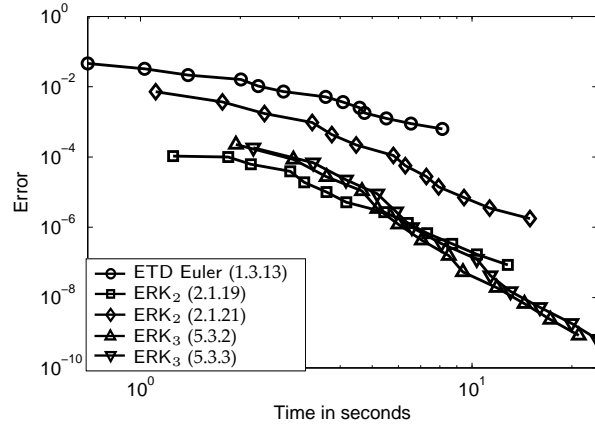


Figure 5.11: CPU Timings against Global Error for Problem 5.1.9

5.3.3 Four-Stage ERKs

Our interest in ERK methods has mostly been with the 4-stage, weakly 4th order schemes. Three such schemes, well established in the literature, are

- That of Krogstad for whom the stability of the scheme took high priority [27],

$$\begin{array}{c|cccc}
 0 & & & & \\
 \frac{1}{2} & \frac{1}{2}\varphi_{1,2} & & & \\
 \frac{1}{2} & \frac{1}{2}\varphi_{1,3} - \varphi_{2,3} & \varphi_{2,3} & & \\
 1 & \varphi_{1,4} - 2\varphi_{2,4} & 0 & 2\varphi_{2,4} & \\
 \hline
 & \varphi_1 - 3\varphi_2 + 4\varphi_3 & 2\varphi_2 - 4\varphi_3 & 2\varphi_2 - 4\varphi_3 & -\varphi_2 + 4\varphi_3
 \end{array} \tag{5.3.5}$$

- The scheme of Cox-Matthews which takes the 3rd order EI extension of the standard 4th order RK, and tweaks it to make a 4th order ERK [13],

$$\begin{array}{c|cccc}
 0 & & & & \\
 \frac{1}{2} & \frac{1}{2}\varphi_{1,2} & & & \\
 \frac{1}{2} & 0 & \frac{1}{2}\varphi_{1,3} & & \\
 1 & \frac{1}{2}\varphi_{1,3} (\varphi_{0,3} - 1) & 0 & \varphi_{1,3} & \\
 \hline
 & \varphi_1 - 3\varphi_2 + 4\varphi_3 & 2\varphi_2 - 4\varphi_3 & 2\varphi_2 - 4\varphi_3 & 4\varphi_3 - \varphi_2
 \end{array} \tag{5.3.6}$$

- The scheme of Strehmel and Weiner who proved that the method is B-consistent of order two [4],

$$\begin{array}{c|cccc}
 0 & & & & \\
 \frac{1}{2} & \frac{1}{2}\varphi_{1,2} & & & \\
 \frac{1}{2} & \frac{1}{2}\varphi_{1,3} - \frac{1}{2}\varphi_{2,3} & \frac{1}{2}\varphi_{2,3} & & \\
 1 & \varphi_{1,4} - 2\varphi_{2,4} & -2\varphi_{2,4} & 4\varphi_{2,4} & \\
 \hline
 & \varphi_1 - 3\varphi_2 + 4\varphi_3 & 0 & 4\varphi_2 - 8\varphi_3 & -\varphi_2 + 4\varphi_3
 \end{array} \tag{5.3.7}$$

Each of these schemes is weakly 4th order.

Also benchmarked is our own scheme, ERK₄c₂c₃ (2.1.45), which is parametrised with two free variables, c₂ and c₃,

$$\begin{array}{c|cccc}
 0 & & & & \\
 c_2 & a_{21} & & & \\
 c_3 & a_{31} & a_{32} & & \\
 c_2 & a_{41} & a_{42} & 0 & \\
 \hline
 & b_1 & b_2 & b_3 & b_4
 \end{array}$$

$$a_{21} = c_2\varphi_{1,2}$$

$$a_{31} = c_3\varphi_{1,3} - \frac{c_3^2}{c_2}\varphi_{2,3}$$

$$a_{41} = c_2\varphi_{1,4} - 2c_2\varphi_{2,4}$$

$$b_1 = \varphi_1 - \frac{c_3 + c_2}{c_2 c_3} \varphi_2 + \frac{2}{c_2 c_3} \varphi_3$$

$$b_3 = -\frac{c_2}{c_3^2 - c_2 c_3} \varphi_2 + \frac{2}{c_3^2 - c_2 c_3} \varphi_3$$

$$a_{32} = \frac{c_3^2}{c_2} \varphi_{2,3}$$

$$a_{42} = 2c_2\varphi_{2,4}$$

$$b_2 = \frac{c_3}{2c_2 c_3 - 2c_2^2} \varphi_2 - \frac{1}{c_2 c_3 - c_2^2} \varphi_3$$

$$b_4 = \frac{c_3}{2c_2 c_3 - 2c_2^2} \varphi_2 - \frac{1}{c_2 c_3 - c_2^2} \varphi_3$$

The scheme is strongly 3rd order. We test the scheme with two choices of parameters

$$c_2 = 1, \quad c_3 = \frac{1}{2} \quad (5.3.8a)$$

$$c_2 = \frac{1}{2}, \quad c_3 = 1 \quad (5.3.8b)$$

As we remarked in Section 2.1, for the choice of parameters (5.3.8b), the scheme achieves weak 4th order.

Numerical Experiment

Again we perform a numerical experiment with the Hochbruck-Ostermann equation (5.1.9)

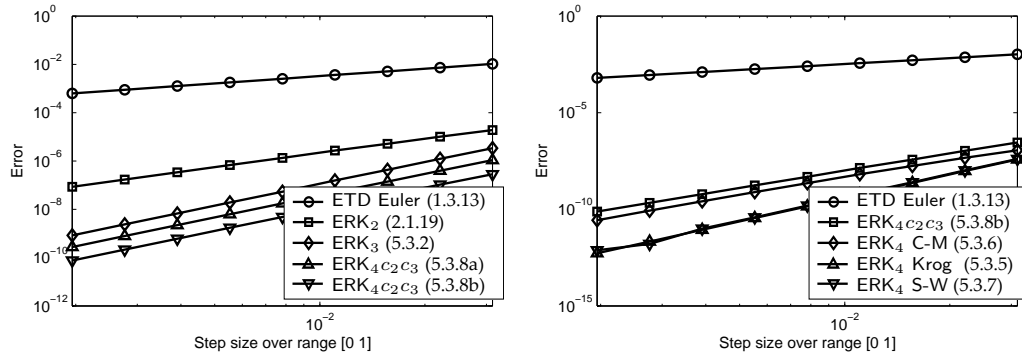


Figure 5.12: Fixed stepsize ERKs for Problem 5.1.9

Figure 5.12 shows the fixed step-size results for the 4-stage schemes, along with some of the earlier schemes for comparison. Scheme ERK₄C₂C₃ was included twice, with two different sets of parameters, to highlight the improvement that a high weak-order scheme can achieve over an exclusively lower strong-order scheme. It is clear from this plot that the schemes ERK₄ Krogstad and ERK₄ Strehmel-Weiner are the superior performers in terms of convergence order for this test problem.

Once again we will look at a break down of the operations cost of each scheme.

Of particular note here is that the Cox-Mathews scheme (5.3.6) would require one additional “linear combination” than we would expect. This is due to the a_{41} entry in the tableau, written as either $\frac{1}{2}\varphi_{1,3}(\varphi_{0,3} - 1)$ or $\frac{1}{4}(\varphi_{1,3})^2$, which cannot be formulated as part of the linear combination structure of PHIPM for stage four.

Scheme	Explicit Evaluation	$\varphi \times$ vector	Linear Comb.
ERK ₄ c_2c_3 , $c_2 = 0.5, c_3 = 1$ (5.3.8a)	8	7	4
ERK ₄ c_2c_3 , $c_2 = 0.5, c_3 = 1$ (5.3.8b)	5	6	4
ERK ₄ Cox-Matthews (5.3.6)	5	6	5
ERK ₄ Krogstad (5.3.5)	6	6	4
ERK ₄ Strehmel-Weiner (5.3.7)	6	6	4

Table 5.8: Relative performance measure for the 4-Stage ERKs

5.3.4 EGLMs

We will run some of the EGLMs seen in Section 2.2 through the same numerical experiment as before. The schemes which we will look at are the strongly 3rd order EGLM₃₂₂ c_2 (2.2.6),

$$\begin{array}{c|cc|cc} c_2 & a_{2,1} & & e^{c_2 hL} & u_{2,1} \\ \hline & b_1 & b_2 & e^{hL} & v_1 \end{array}$$

$$a_{2,1} = c_2\varphi_{1,2} + c_2^2\varphi_{2,2}$$

$$b_1 = \varphi_1 + \frac{c_2 - 1}{c_2}\varphi_2 + \frac{-2}{c_2}\varphi_3$$

$$u_{2,1} = -c_2^2\varphi_{2,2}$$

$$b_2 = \frac{1}{c_2^2 + c_2}\varphi_2 + \frac{2}{c_2^2 + c_2}\varphi_3$$

$$v_1 = \frac{-c_2}{c_2 + 1}\varphi_2 - \frac{2}{c_2 + 1}\varphi_3$$

and the strongly 4th order EGLM₄₂₃ c_2 (2.2.8),

$$\begin{array}{c|cc|ccc} c_2 & a_{21} & & e^{c_2 hL} & u_{21} & u_{22} \\ \hline & b_1 & b_2 & e^{hL} & v_1 & v_2 \end{array}$$

$$a_{2,1} = c_2\varphi_{1,2} + \frac{3c_2^2}{2}\varphi_{2,2} + c_2^3\varphi_{3,2}$$

$$u_{2,1} = -2c_2^2\varphi_{2,2} - 2c_2^3\varphi_{3,2}$$

$$b_1 = \varphi_1 + \frac{\frac{3c_2-2}{2}\varphi_2 + c_2 - 3\varphi_3 - 3\varphi_4}{c_2}$$

$$v_1 = \frac{-2c_2\varphi_2 - 2c_2 - 4\varphi_3 + 6\varphi_4}{c_2 + 1}$$

$$u_{2,2} = \frac{c_2^2}{2}\varphi_{2,2} + c_2^3\varphi_{3,2}$$

$$b_2 = \frac{2\varphi_2 + 6\varphi_3 + 6\varphi_4}{c_2^3 + 3c_2^2 + 2c_2}$$

$$v_2 = \frac{\frac{c_2}{2}\varphi_2 + c_2 - 1\varphi_3 - 3\varphi_4}{c_2 + 2}$$

For both schemes we will also look at the case $c_2 = 1$, as this provides the greatest opportunity for constructing an optimal implementation. Note that, when $c_2 = 1$, these schemes have appeared in the literature [38] as EGLM₃₂₂ and EGLM₄₂₃ respectively.

Numerical Experiment

As part of this experiment, we will also include the best performing scheme seen so far, in this case ERK₄ Krogstad along with the baseline ETD Euler (1.3.13), to help illustrate the relative

rankings of these EGLMs. For these experiments we set $c_2 = 0.5$ for schemes (2.2.6) and (2.2.8).

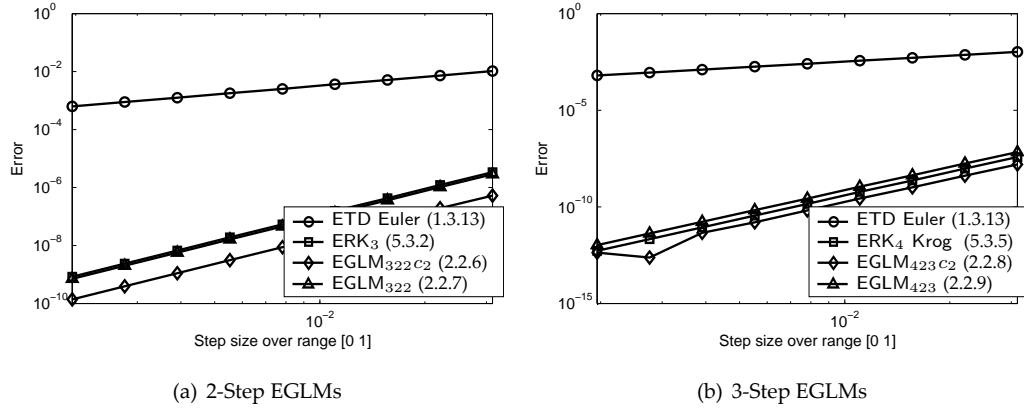


Figure 5.13: Fixed stepsize EGLMs for Problem 5.1.9

We see from Figure 5.13(b) that the $c_2 = 1$ case for both EGLMs, (2.2.7) and (2.2.9), causes a slight accuracy reduction, but not an order reduction. Table 5.9 summarises the computational cost of each method and it shows clearly that the accuracy drop is compensated for in terms of reduced computational cost in both the 'Explicit Evaluation' and ' $\varphi \times$ vector' categories.

Scheme	Explicit Evaluation	$\varphi \times$ vector	Linear Comb.
EGLM ₃₂₂ c_2 (2.2.6)	5	5	2
EGLM ₃₂₂ (2.2.7)	3	4	2
EGLM ₄₂₃ c_2 (2.2.8)	7	7	2
EGLM ₄₂₃ (2.2.9)	4	6	2

Table 5.9: Relative performance measure for the 4-Stage EGLMs

5.3.5 EARKs and EAGLMs

Finally we study the performance characteristics of our new EARK schemes. Specifically, we look at a 3rd order and a 4th order scheme, both 2-stage. These schemes, introduced in Section 3.2, are referred to as EARK₃₂₁ c_2 and EARK₄₂₂ c_2 respectively and we will include the particular case of parameter choice where $c_2 = 1$, as this allows for the most efficient implementation.

$$\begin{array}{c|cc|c|c}
 1 & \varphi_1 & & & \varphi_2 \\
 1 & \varphi_1 - 2\varphi_3 & 2\varphi_3 & 0 & \varphi_2 - 2\varphi_3 \\
 1 & \varphi_1 - 2\varphi_3 & 2\varphi_3 & 0 & \varphi_2 - 2\varphi_3
 \end{array} \tag{5.3.9}$$

$$\begin{array}{c|cc|cc}
1 & \varphi_1 & & \varphi_2 & \varphi_3 \\
1 & \varphi_1 - 6\varphi_4 & 6\varphi_4 & \varphi_2 - 6\varphi_4 & \varphi_3 - 3\varphi_4 \\
\hline
& \varphi_1 - 6\varphi_4 & 6\varphi_4 & 0 & \varphi_2 - 6\varphi_4 & \varphi_3 - 3\varphi_4
\end{array} \tag{5.3.10}$$

Note that we do not need to consider the different methods of computing the outgoing derivatives. The reason for this is that the computational cost of producing the derivatives is negligible compared with computing the φ -functions. Therefore, in performance terms, an EAGLM scheme with zero u_{ij} and v_{ij} entries has identical CPU performance to that of an EARK scheme, regardless of the format of the M matrix.

Numerical Experiments

We follow the same procedure as before with EAGLMs, plotting them against the best performing 3rd order and 4th order EGLMs.

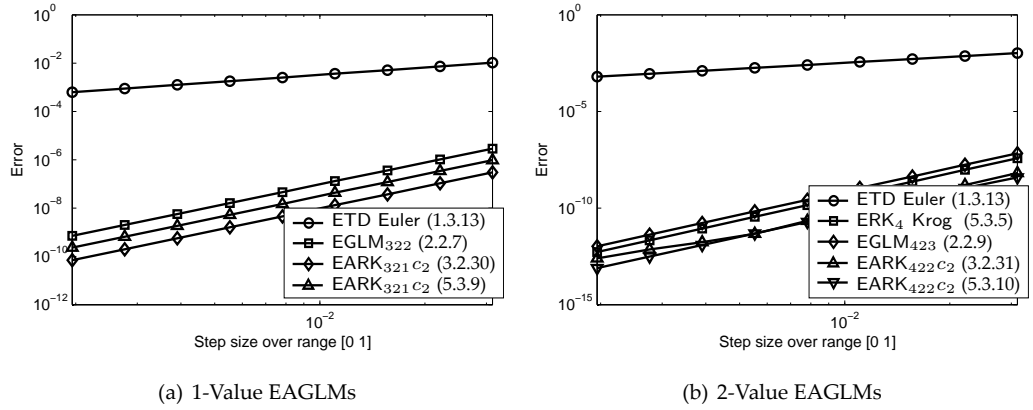


Figure 5.14: Fixed stepsize EAGLMs for Problem 5.1.9

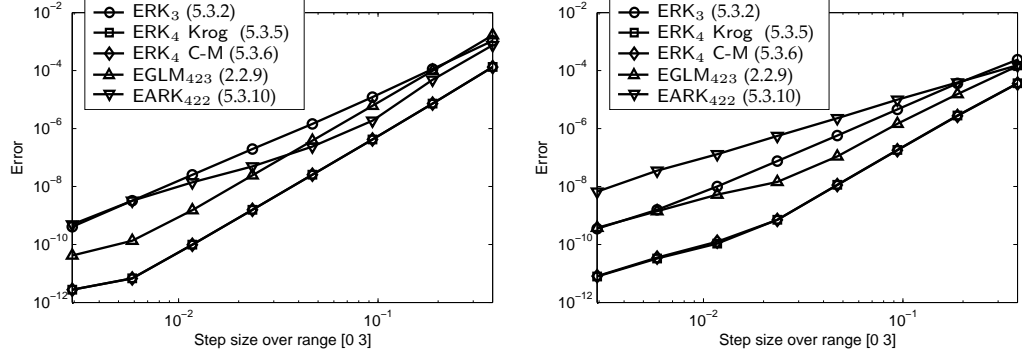
An interesting result of this experiment is that $\text{EARK}_{422}c_2$ does not suffer at all from accuracy degradation for the case $c_2 = 1$, as was observed in the EGLMs and $\text{EARK}_{321}c_2$. Table 5.10 summarises the computational cost of each method

Scheme	Explicit Evaluation	$\varphi \times \text{vector}$	Linear Comb.
$\text{EARK}_{321}c_2$ (3.2.30)	5	5	2
EARK_{321} (5.3.9)	3	3	2
$\text{EARK}_{422}c_2$ (3.2.31)	7	7	2
EARK_{422} (5.3.10)	4	4	2

Table 5.10: Relative performance measure for the 4-Stage EAGLMs

The Allen-Cahn Equation Experiments

We run two fixed stepsize experiments, using the same parameter configurations as Kassam & Trefethen [26, (3.4)] and Krogstad [27, Section 5.2]. Those parameters are $N = 80$, $\varepsilon = 0.001$ and $N = 50$, $\varepsilon = 0.01$ respectively. In both cases we integrate for $t \in [0, 3]$.



(a) Comparison with [26, Fig. 3, pg. 1224]

(b) Comparison with [27, Fig. 8, pg. 85]

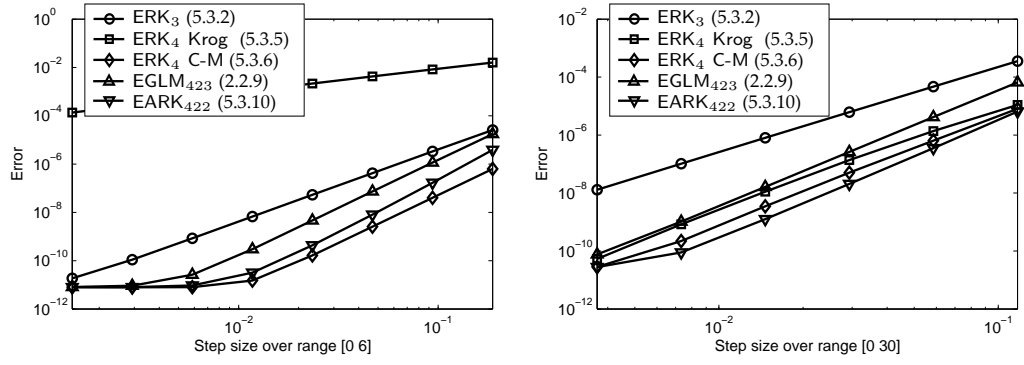
Figure 5.15: Allen-Cahn Equation, fixed stepsize, $N = 50$

The results of the fixed stepsize experiments match up with the corresponding experiments in [26, 27]. From Figure 5.15(a) we can confirm that our implementation of ERK₄ Cox-Matthews is achieving the correct performance, while Figure 5.15(b) confirms the same for our ERK₄ Krogstad implementation. It is clear from these experiments that these 4th order ERKs perform better than our adaptive schemes, EGLM₄₂₃ and EARK₄₂₂.

The Kuramoto-Sivashinsky Equation Experiments

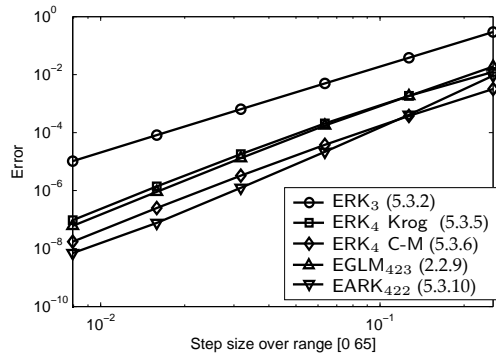
To perform a comparison with results published in [26, 27, 13] we run three fixed stepsize experiments, the first with $N = 32$, $t \in [0, 3]$ and then two experiments with $N = 128$ for times, $t \in [0, 30]$ and $t \in [0, 65]$.

As with the Allen-Cahn comparisons, the results of our fixed stepsize experiments match up, for the most part, with those published already. An exception are the results from Figure 5.16(b) which seem to show our ERK implementations achieving superior accuracy over the results in [26, Fig. 3, pg. 1224].



(a) Comparison with [13, Fig. 7, pg. 447]

(b) Comparison with [26, Fig. 3, pg. 1224]



(c) Comparison with [27, Fig. 4, pg. 85]

Figure 5.16: Kuramoto-Sivashinsky Equation, fixed stepsize

5.3.6 Summary

The conclusion that can be drawn from this analysis is that the EARK and EAGLM schemes offer the best accuracy performance in their respective order categories.

In terms of computational efficiency, EARKs and EAGLMs are as efficient as the EGLMs under the “Explicit Evaluation” and “ $\varphi \times$ vector Linear Combinations” methods of computing the φ -functions.

Under the “ $\varphi \times$ vector” approach, the EARKs are the most efficient schemes in the 3rd and 4th order categories by a significant margin. This margin would increase for higher order methods, a 2-stage p order EGLM with $c_2 = 1$, will require $2p - 2$ “ $\varphi \times$ vector” operations, while a similar EARKs would require only p operations.

5.4 Variable Step Size

5.4.1 Basic Requirements

If exponential integrators are to be benchmarked competitively against traditional solvers, they will need to implement adaptive stepsize strategies as well as generate efficient and reliable error estimates. We shall see that EARKs and EAGLMs offer both of these features.

A fundamental requirement for a scheme is its ability to handle changes in step size without losing stability or accuracy. Schemes must possess this property if they are to work within an adaptive step-size environment. The 1-step nature of some schemes, such as those constructed from ERKs, makes them inherently compatible with a variable stepsize environment. Such schemes can be used without any changes to their implementation.

Multi-step Schemes

Multi-step schemes, such as EGLMs in the format presented so far, are not practical for use in a variable stepsize algorithm. As they stand, they would require a number of restarting steps to be calculated after every change in stepsize. The issue is, that during the derivation of the order conditions in Section 2.2, and that when constructing the schemes, an assumption was made that the previous timesteps occurred with uniform spacing.

The solution to this is to not treat the previous values as being at integral steps lengths during the derivation of the order conditions. This means that when Taylor expanding the previous values we view them in the more general form N_{t_n-j}

$$\begin{aligned} N_{t_n-j} &\equiv N(t_n - jh, y(t_n - jh)) \\ &= N - jhN' + \frac{1}{2}j^2h^2N'' - \frac{1}{6}j^3h^3N''' + \dots \end{aligned} \tag{5.4.1}$$

where j is any positive real number. Thus it is possible to construct an EGLM scheme which is fully capable of preserving both its stability and accuracy.

Sample 4th Order EGLM Here is an example 4th Order scheme with previous steps $N_{t_{n-p}}$ and $N_{t_{n-q}}$

0					
c	a ₂₁		e^{c_2hL}	u ₂₁	u ₂₂
	b ₁	b ₂	e^{hL}	v ₁	v ₂

$$\begin{aligned}
a_{2,1} &= c\varphi_{2,1} + \frac{c^2(p+q)}{pq}\varphi_{2,2} + \frac{2c^3}{pq}\varphi_{2,3} \\
u_{2,1} &= \frac{c^2p}{p^2-pq}\varphi_{2,2} + \frac{2c^3}{p^2-pq}\varphi_{2,3} \\
b_1 &= \varphi_1 + \frac{(c(p+q)-pq)\varphi_2 + 2(c-pq)\varphi_3 - 6\varphi_4}{cpq} \\
v_1 &= \frac{-2cp\varphi_2 + 2(q-c)\varphi_3 + 6\varphi_4}{(c+q)pq - p^3 - cp^2} \\
u_{2,2} &= \frac{c^2p}{q^2-pq}\varphi_{2,2} + \frac{2c^3}{p^2-pq}\varphi_{2,3} \\
b_2 &= \frac{pq\varphi_2 + 2(p+q)\varphi_3 + 6\varphi_4}{c^3 + c^2p + c(p+c)q} \\
v_2 &= \frac{cp\varphi_2 + 2(c-p)\varphi_3 - 6\varphi_4}{q^3(c-p)q^2 - cpq}
\end{aligned}$$

Multi-value Schemes

Multi-value EARKs, by their 1-step nature, do not require the same treatment as EGLMs to make them compatible with a variable stepsize environment. Indeed, both the order conditions and the majority of the scheme tableau remain unchanged. However, care does need to be taken regarding the incoming and outgoing approximations. The lower, $M = (\beta \delta)$, section of an EARK tableau is a matrix which is constructed on the assumption that the approximations are scaled by the current step size, h . For example, taking scheme EARK_{432c_2} (3.1.17), and fixing $c_2 = \frac{1}{2}$, we have the M matrix

$$M = \left(\begin{array}{ccc|cc} -4 & -8 & 4 & 1 & 0 \\ 22 & -32 & 10 & 6 & 0 \end{array} \right) \quad (5.4.2)$$

We have seen already in (3.1.12) how this matrix operates on the vector of input values and on outgoing approximations of the current step, to produce the input derivative approximations for the next step. In that earlier example, we were assuming a constant stepsize h . With the switch to variable stepsizes, one must ensure that the incoming approximations are scaled only by the current h_n and not by any previous h_{n-j} .

This implementation detail is best taken care of by writing the M matrix as a function of h_n . Each element of the M matrix becomes multiplied by a h^α where α obeys the following simple pattern

$$\left(\begin{array}{ccc|ccc} \beta_{11} & \cdots & \beta_{ij} & \delta_{11} & \cdots & \delta_{i1} \\ \vdots & \ddots & & \vdots & \ddots & \\ \beta_{1j} & & \beta_{ij} & \delta_{1j} & & \delta_{ij} \end{array} \right) : \left(\begin{array}{ccc|ccc} h^{-1} & \cdots & h^{-1} & h^0 & \cdots & h^{i-1} \\ \vdots & \ddots & & \vdots & \ddots & \\ h^{-j} & & h^{-j} & h^{1-j} & & h^{i-j} \end{array} \right) \quad (5.4.3)$$

Returning to our example (5.4.2) we now express M as the following function

$$M(h) = \left(\begin{array}{ccc|cc} \frac{-4}{h} & \frac{-8}{h} & \frac{4}{h} & 1 & 0h \\ \frac{22}{h^2} & \frac{-32}{h^2} & \frac{10}{h^2} & \frac{6}{h} & 0 \end{array} \right) \quad (5.4.4)$$

where we generate the output derivatives through the operation

$$M(h_n) \begin{pmatrix} N_n \\ N_{n+c_2} \\ N_{n+1} \\ N'_n \\ N''_n \end{pmatrix} = \begin{pmatrix} N'_{n+1} \\ N''_{n+1} \end{pmatrix} \quad (5.4.5)$$

Here we see that the vectors of incoming and outgoing approximations have no dependence on any stepsize h .

We noted earlier, a property of the M matrices is that the lower rows do not involve any φ -functions. When we add this distinction to the complexity introduced in adapting the matrices to handle variable h , we are naturally led to exclusively viewing M as a separate entity, presented alongside the more traditional tableau. This can be done without sacrificing the possibility of reintegrating the two to recover the tableau representation given earlier (3.1.3). This parallels neatly with the implementation details of the schemes where the two elements are treated very differently due to the computationally beneficial lack of φ 's in the M matrix.

Variable Stepsize Compatibility of EAGLMs

The hybrid EAGLMs, which we are most interested in, must follow the EGLMs approach of using an expansion of the previous values in deriving the conditions for producing the internal stage and initial output approximations.

In determining the coefficients for the M matrix, we must combine the techniques developed for EARKs with the more general form of the N_{n-j} 's. Together, this allows us to construct approximations of sufficiently high orders to N'_{n+1} and N''_{n+1} as before. The format which this follows is almost identical to that seen in example (5.4.3). The lower M matrix of an EAGLM tableau has an additional section of γ_{ij} values, as seen in (3.2.1), the h^n multipliers derived from those γ_{ij} entries follow the same pattern as those from the β_{ij} entries.

$$\left(\begin{array}{c|ccc|c} & \gamma_{11} & \cdots & \gamma_{ij} & \\ \beta_{ij} & \vdots & \ddots & & \delta_{ij} \\ & \gamma_{1j} & & \gamma_{ij} & \end{array} \right) : \left(\begin{array}{c|ccc|c} & h^{-1} & \cdots & h^{-1} & \\ h^{-j} & \vdots & \ddots & & h^{i-j} \\ & h^{-j} & & h^{-j} & \end{array} \right)$$

The three free variables; the two representing the non-integral step lengths of the previous values, and the free c_2 variable present in most schemes, makes the tableau's very difficult to construct without the help of a symbolic calculator. And even with that, the resultant tableau's are too complex to represent succinctly.

5.4.2 Truncation Error Estimation

The local truncation error is defined as the amount by which the exact solution $y(t_n)$ fails to satisfy the difference equation of the numerical method [39]. An equivalent definition of the local truncation error is the error incurred by taking a single step assuming exact information, that is, a single step using exact past values, solution derivatives and assuming no rounding errors [39].

The latter definition can be very useful in testing schemes which cannot produce their own truncation error estimate. In addition, when the scheme being investigate can produce estimates it can still be advantageous to use this exact truncation error, doing so allows us to concentrate and study other aspects of the integration procedure in isolation.

Embedding

A well established method for producing a truncation error estimate is to take each step with two methods, one of order p and the second of order $p + 1$ producing two estimated solutions y_{n+1} and \hat{y}_{n+1} . Then the difference $|y_{n+1} - \hat{y}_{n+1}|$, is a good estimate of the lower schemes true local truncation error. This estimate can then be used to guide the integrator. It is natural to use the higher order estimate, \hat{y}_{n+1} , to advance the integration. In this case $|y_{n+1} - \hat{y}_{n+1}|$ serves only to control the stepsize. This process is known as “local extrapolation” [20].

Taking each step twice introduces considerable extra computation. To limit this additional cost we try to choose, or design, the two schemes so that much of the work in generating the lower order estimate, y_{n+1} , can be reused for the higher order one, \hat{y}_{n+1} . A common approach to this is known as embedding, whereby the lower order scheme can be written as embedded within the tableau for the higher order scheme.

Classical RK Embedding

When designing an embedded RK we construct two schemes, producing a y_{n+1} and \hat{y}_{n+1} approximation, where the two methods use the same function values [20]. Because of their similar

nature, these two RKs can be written within the same tableau

$$\begin{array}{c|ccc}
 0 & & & \\
 c_2 & a_{21} & & \\
 \vdots & \vdots & \ddots & \\
 c_s & a_{s1} & \cdots & a_{s,s-1} \\
 \hline
 & b_1 & \cdots & b_{s-1} & b_s \\
 \hline
 & \hat{b}_1 & \cdots & \hat{b}_{s-1} & \hat{b}_s
 \end{array} \tag{5.4.6}$$

such that

$$y_{n+1} = y_n + h(b_1 K_1 + \dots + b_s K_s) \tag{5.4.7}$$

$$\hat{y}_{n+1} = y_n + h(\hat{b}_1 K_1 + \dots + \hat{b}_s K_s) \tag{5.4.8}$$

An example of a 2nd and 3rd order embedded RK, also known as a (2,3) pair is the method of Bogacki & Shampine [6]

$$\begin{array}{c|ccc}
 0 & & & \\
 \frac{1}{2} & \frac{1}{2} & & \\
 \frac{3}{4} & 0 & \frac{3}{4} & \\
 1 & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} \\
 \hline
 & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} & 0 \\
 \hline
 & \frac{7}{24} & \frac{1}{4} & \frac{1}{3} & \frac{1}{8}
 \end{array} \tag{5.4.9}$$

This method is familiar from it's use as part of the Matlab ODE suite, where it is referred to by the function name ode23.

ERK Embedding

In adapting the classical notion of embedding to EIs, we can construct a trivial example of ERK embedding by viewing the 1-stage ETD Euler (1.3.13) as embedded within the 2-stage ERK₂ (2.1.19), with $c_2 = 1$,

$$\begin{array}{c|cc|c}
 0 & & & I \\
 1 & 1\varphi_1 & & e^{hL} \\
 \hline
 & \varphi_1 & 0 & \\
 \hline
 & \varphi_1 - \varphi_2 & \varphi_2 &
 \end{array} \tag{5.4.10}$$

This approach embeds a strongly 1st order scheme within a strongly 2nd order scheme. By subtracting the formula for the 1st and 2nd order approximations, \hat{y}_{n+1} and y_{n+1} respectively,

and the difference between the two schemes

$$\begin{aligned}
y_{n+1} &= Y_1 = e^{hL} y_n + h\varphi_1 N_n \\
\hat{y}_{n+1} &= e^{hL} y_n + h\varphi_1 N_n + h\varphi_2 (-N_n + N(t_{n+1}, Y_1)) \\
\rightarrow |y_{n+1} - \hat{y}_{n+1}| &= h\varphi_2 (-N_n + N(t_{n+1}, Y_1))
\end{aligned} \tag{5.4.11}$$

gives us a truncation error estimate for ETD Euler. We have verified, experimentally that this is a robust estimate for guiding an integration controller. In addition to its reliability, the 1-step nature of this embedded scheme means that it can be used for generating starting values to be used subsequently by a multi-step scheme, or to generate derivative approximations for EARKs.

Developing embedded ERKs of higher order proved more troublesome. We discovered that the weak order properties, common to the higher order, meant the truncation errors produced were unreliable.

EAGLM Embedding

Embedding can be extended beyond the (E)RK families of methods to multi-step and multi-value schemes. In the case of the 4th order EARK_{422c2} (3.2.31), it is possible to produce a 3rd order error estimate by embedding a 3rd order scheme, for example EARK_{321c2} (3.2.30), within its tableau.

Scheme EARK_{422c2} requires both N' and N'' , unlike EARK_{321c2} which uses only N' . The 1st stage formula for both schemes is therefore different and limits the potential for reusing a calculation from the 3rd order step to generate the 4th. By constructing a new 3rd order scheme, which has an identical 1st stage to EARK_{422c2}, we can greatly reduce the computational cost of estimating the truncation error.

Such a scheme is the 3rd order method EARK_{422c2}, a 3-stage 3-value EAGLM

c_2	$a_{2,1}$	$w_{2,1}$	$w_{2,2}$	$=$	c_2	$c_2\varphi_{1,2}$	$c_2^2\varphi_{2,2}$	$c_2^3\varphi_{3,2}$
	$b_1 \quad b_2$	$z_1 \quad z_2$				$\varphi_1 - \beta\varphi_4 \quad \beta$	$\varphi_2 - c_2\beta\varphi_4$	$\varphi_3 - \frac{c_2^2\beta}{2}\varphi_4$
	$b_1 \quad b_2$	$z_1 \quad z_2$				$\varphi_1 - \beta\varphi_4 \quad \beta$	$\varphi_2 - c_2\beta\varphi_4$	$\varphi_3 - \frac{c_2^2\beta}{2}\varphi_4$

(5.4.12)

where β is a free parameter. This scheme is embeddable within EARK_{422c2}.

If we choose $\beta = 0$ then the difference between the two schemes, (3.2.31) and (5.4.12), can be calculated explicitly as

$$\hat{y}_{n+1} - y_{n+1} = \frac{6}{c_2^3}\varphi_4 N_n - \frac{6}{c_2^3}\varphi_4 K_2 + \frac{6}{c_2^2}\varphi_4 N'_n + \frac{3}{c_2}\varphi_4 N'_n \tag{5.4.13}$$

$$= \varphi_4 \left(\frac{6}{c_2^3} N_n - \frac{6}{c_2^3} K_2 + \frac{6}{c_2^2} N'_n + \frac{3}{c_2} N'_n \right) \tag{5.4.14}$$

Under both the “Explicit Evaluation” and ‘ $\varphi \times$ vector’ approaches to implementing the integrator, the calculation (5.4.14) is already performed as part of the procedure for computing y_{n+1} . Producing the truncation estimate does not change the scheme’s implementation, or the computational expense summaries as seen in Table 5.10, row 5, columns 2 & 3. In a sense, the truncation error estimate is produced for free as part of the normal calculation of a step.

Numerical Experiments

Looking at the numerical results from fixed-step experiments, we can quickly get an overview of the local error behaviour of a given problem. Figure 5.17 plots the results for four problems. In the plots the grey curves plot a profile of the estimated solution, and the thick black line plot the exact local error.

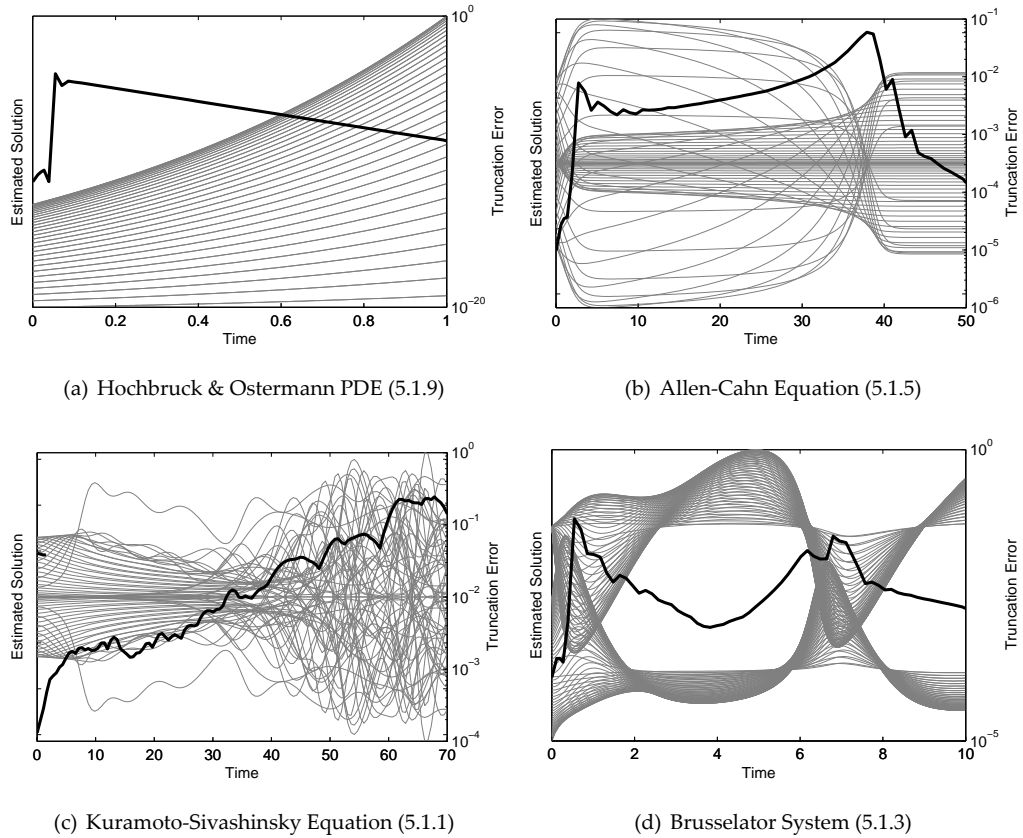


Figure 5.17: Illustration of Local Error Profiles

The Brusselator System (5.1.3), which exhibits erratic behaviour represented the toughest test of a robust integration controller. The Allen-Cahn problem (5.1.5) exhibits interesting behaviour near $t = 37$. There is a spike in the local truncation errors about this point (visible in Figure 5.17(b)) which makes the problem a good test for a stepsize controller.

It is important to note that the low local error exhibited at the beginning of each of the problems in Figure 5.17 is a results of our method for producing the starting values.

5.4.3 Stepsize Adjustment

The goal of an efficient solver is to control the stepsize, h , with the aim of performing the numerical integration in as few time step as possible, while at the same time, keeping the local error below a user specified tolerance. The tolerance value, τ , is derived from two user specified values; and absolute tolerance, τ_{abs} , and a relative tolerance, τ_{rel}

$$\tau = \tau_{\text{abs}} + \|y_n\| \tau_{\text{rel}} \quad (5.4.15)$$

The basic approach to this is to increase h where the local error is below τ , and decrease the stepsize where the error exceeds τ . In practice, a solver will not have access to the *exact* local truncation error, so instead local error estimates are used to guide the algorithm.

Types of Stepsize Controller

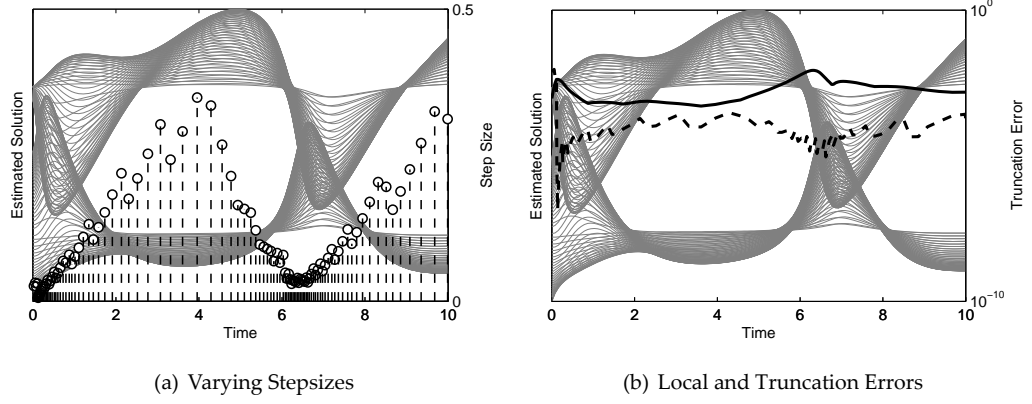
There is no one approach to govern how the stepsize should be adjusted so as to control the truncation error and indeed, designing methods to do so, is a field of study in itself. We looked at a number of different algorithms, referred to as controllers, starting with a very basic algorithm and, from that, constructed increasingly complex procedures in order to more intelligently predict an appropriate subsequent stepsize.

Naive Approach. The simplest approach is to compare the estimated truncation error, e_n , to a user specified tolerance, τ , and increase the step size if the estimation is below tolerance or decrease and repeat the step, if the truncation error exceeds the tolerance. For example,

```

1: if  $\tau < e_n$  then
2:    $h_{n+1} \leftarrow \frac{h_n}{\alpha}$ 
3:   redo
4: else
5:    $h_{n+1} \leftarrow \alpha h_n$ 
6:   continue
7: end if
```


where $\alpha > 1$ is simply some scaling value to adjust how quickly the step size changes, often $\alpha = 2$.



Accepted Steps	Rejected	Total	Global Error
105	26	131	8.5×10^{-3}

(c) Integration Statistics

Figure 5.18: Scheme (3.2.30) with the Stepping Controller for Problem (5.1.3)

Figure 5.18 shows the results of a numerical experiment integrating the Brusselator system (5.1.3) with $\text{EARK}_{321}c_2$, $c_2 = 1$. In Figure 5.18(b) the dashed line is the estimated truncation error, e_n , maintained by the controller, while the solid line plots the exact local error. Table 5.20(c) records the timestep and global error statistics. The visible fluctuation of the stepsize indicates that this approach is prone to oscillating about the optimal step length. Though the global error is higher, it is still within the specified tolerance.

Guided Approach. There is information available to a controller, in the form of the magnitude of the difference between e_n and the desired tolerance, τ . This can be used as a guide to determine how much h should be adjusted by, to get e_{n+1} as close to τ as possible. For example

$$|y_{n+1} - \hat{y}_{n+1}| = e_{n+1} \quad (5.4.16)$$

$$h_{\text{opt}} = h_n \times \left(\frac{\hat{\tau}}{e_n} \right)^{\frac{1}{p+1}} \quad (5.4.17)$$

where p is the order of the scheme used to generate y_{n+1} [20]. An algorithm implementing such a controller could be written as

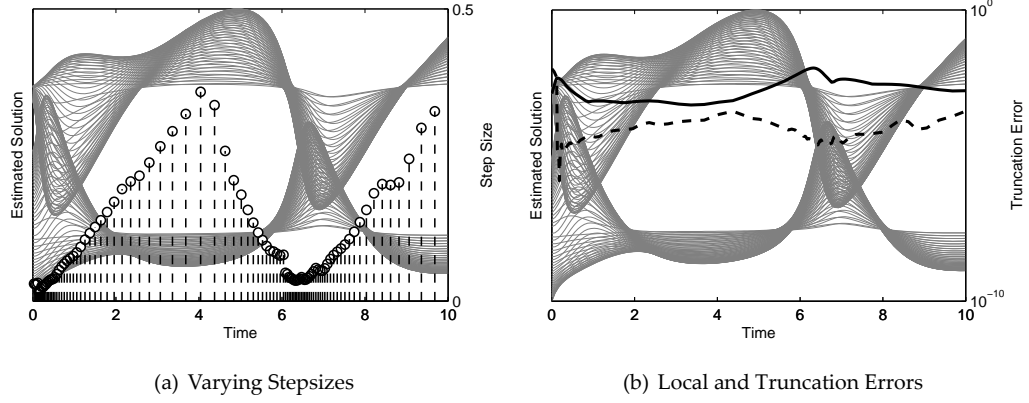
- 1: $\beta \leftarrow \left(\frac{\hat{\tau}}{e_n} \right)^{\frac{1}{p+1}}$
- 2: $h_{n+1} \leftarrow \beta h_n$
- 3: **if** $\tau < e_n$ **then**
- 4: **redo**
- 5: **else**

```

6:   continue
7: end if

```

For our experiments with $\text{EARK}_{321}c_2$ we set $p = 4$. The tolerance value $\hat{\tau}$ is usually set to some value just smaller than the cut-off tolerance, τ . This helps to reduce the number of rejected steps. In our experiments we usually set $\hat{\tau} = 0.8\tau$. An alternative approach is to use a “safety factor”, and we will see this used in the controller we introduce next.



Accepted Steps	Rejected	Total	Global Error
99	23	122	1×10^{-2}

(c) Integration Statistics

Figure 5.19: Scheme (3.2.30) with the Guided Controller for Problem (5.1.3)

Figure 5.19 shows an experiment using the guided adapter. Table 5.19(c) shows an overall improvement in terms of the performance of the scheme. Most interesting here is Figure 5.19(a) where we can see a very smooth and controlled adjustment of the stepsizes. This is a significant improvement over the behaviour seen in Figure 5.18(a).

Established Approaches. A more traditional approach from control theory is the integral or I-controller. The format of the controller makes use of a safety factor, $\gamma < 1$.

$$\log h_{n+1} = \log h_n + \frac{1}{p+1} (\log \|\gamma^{p+1} \xi\| - \log \|e_n\|) \quad (5.4.18)$$

We implement the I-controller with the following algorithm.

```

1:  $e_c \leftarrow \log \|\gamma^{p+1} \tau\| - \log \|e_n\|$ 
2:  $h_{n+1} \leftarrow \exp \left( \log(h_n) + \frac{e_c}{p+1} \right)$ 
3: if  $\tau < e_n$  then
4:   redo
5: else

```

6: **continue**
7: **end if**

Here p is the order of the method, for the experiments in Figure 5.20 we set $p = 4$ and $\gamma = 0.9$.

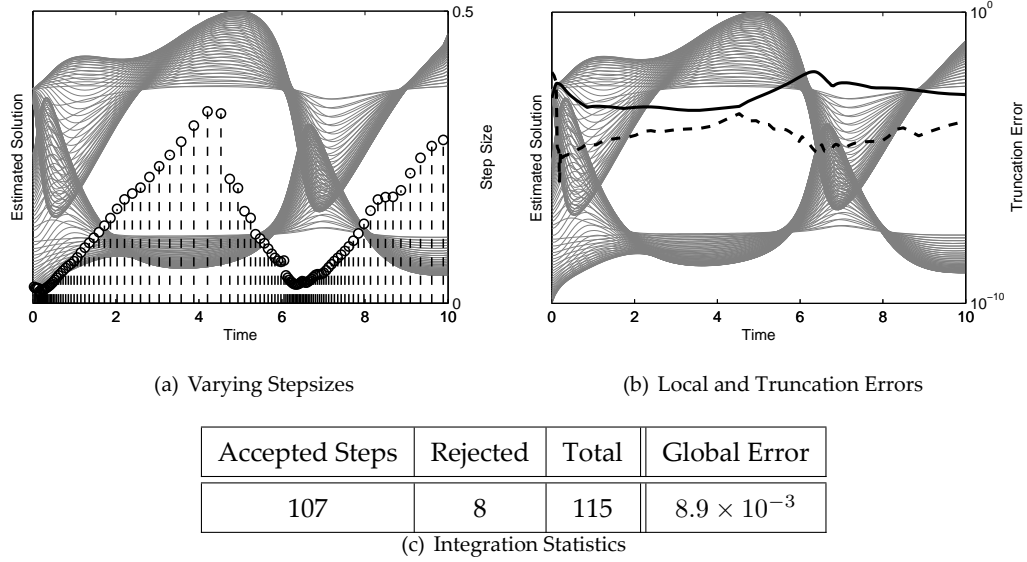


Figure 5.20: Scheme (3.2.30) with the I-controller for Problem (5.1.3)

Table 5.20(c) shows that the number of rejected steps has been significantly decreased, although it did take more accepted steps to complete the integration. Table 5.11 summarises the results for all three controllers run for a tighter tolerance.

Controller	Accepted Steps	Rejected	Total	Global Error
Naive-controller	589	129	718	4.72×10^{-5}
Guided-controller	593	10	603	3.49×10^{-5}
I-controller	634	4	638	2.49×10^{-5}

Table 5.11: Scheme (3.2.30) with all Controllers for Problem (5.1.3)

Step Rejection

Each rejected step represents a waste of computational time, as the estimated solution it produces must be discarded, therefore minimising their occurrences is of high priority. In the controller approaches outlined above, reducing the number of rejected steps was addressed by either using a slightly lower tolerance, $\hat{\tau}$, as in the guided approach, or by using a safety factor, γ , as in the I Controller. Such controllers work by selecting a h_{n+1} , for the next step,

based on the current local error. This means, that for some problems, that when the integrator encounters a period of increasing local error, rejected steps are almost inevitable.

Experiments indicate that rejected steps tend to occur in clusters during the periods where the local truncation error for the solution is increasing. It makes sense, therefore, to *over* compensate for a rejected step with a smaller h_{n+1} than reported as the optimal choice by the controller. One approach is to reduce h_{n+1} by a greater amount than indicated by the controller's basic calculation, such as (5.4.18) for example. This means that, upon encountering a rejected step at time t_r , the controller would reduce the recommended next stepsize h_{r+1} by a factor $\alpha < 1$. To achieve this correctly, a sophisticated controller must avail of time-stepping history. Without history, the following stepsize, h_{r+2} , would no longer take this compensation factor, α , into account.

At some point $t_n > t_r$ the controller will stop taking α into account. To prevent a sudden increase in stepsize, we adopted an algorithm to gradually increase α towards 1 over a number of steps l . As a result, α has less influence over the controller's recommended stepsize the further the integrator has advanced from t_r . For $t_n > t_{r+l}$ we stop taking α into account. We used the following method to control how the influence of α decreases over time.

$$h_{r+i} = \min \left[h_{r+i}, \alpha h_r + \left(h_{r+i} - \alpha h_r \left(\frac{i}{l} \right)^\beta \right) \right] \quad (5.4.19)$$

Here the β exponent controls the behaviour of this interpolation. For $\beta = 1$, the interpolation is linear, while $\beta = 2$ gives a quadratic interpolation. The min function ensures that we never force a higher h_{n+1} than is recommended.

Figure 5.21 illustrates the effects of different values for β , when an artificial rejected step is simulated in a fixed stepsize experiment. In practice the effects different β 's have is subtle and varied depending on the problem, on average settling on the value $\beta = 1.5$ seemed to work well.

5.4.4 Initial Stepsize and Starting Points

We approached the problems of selecting an initial stepsize, and generating starting points, with the use of a one step method. The ideal scheme to use was ERK_2 (2.1.19) with the embedded ETD Euler (1.3.13) as described in Section 5.4.2. Starting with some arbitrary stepsize, we could rely on the controller to ensure the initial steps were of sufficient accuracy. Once enough steps had been generated they would be used to compute the incoming N -function derivatives for EARKs, and be passed to multi-step EGLMs and EAGLMs.

The ERK_2 / ETD Euler scheme is only of 2nd order compared with the 4th order schemes

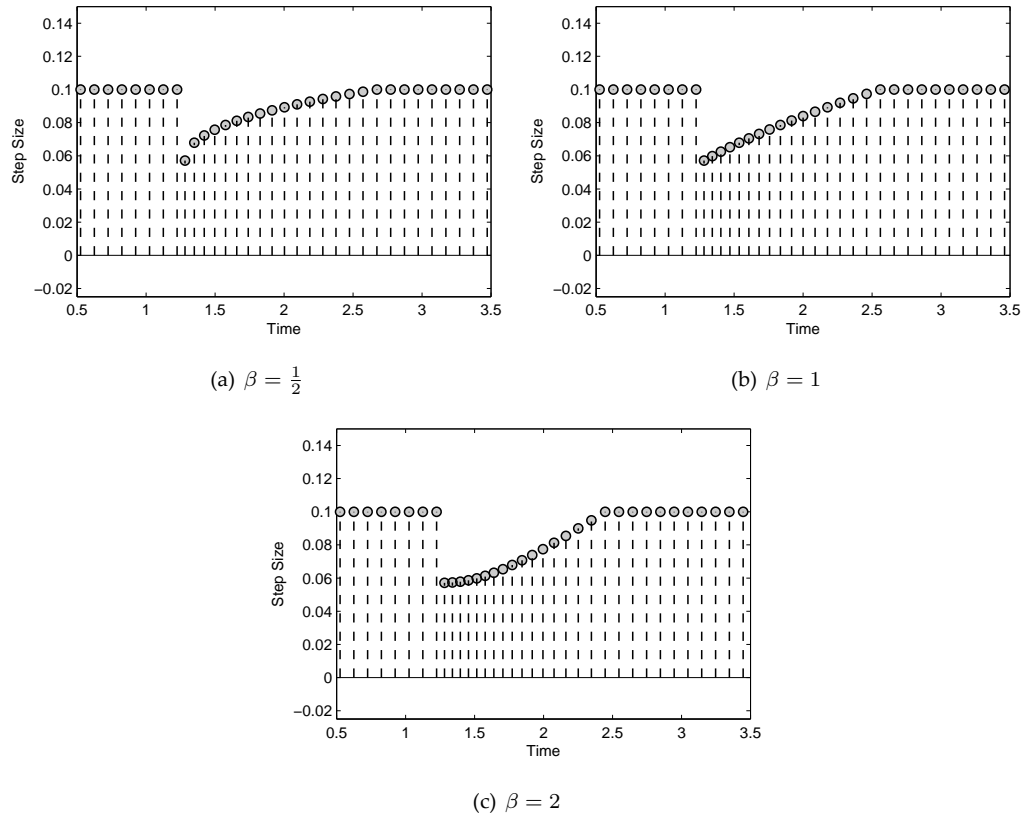


Figure 5.21: Illustration of Different β Exponents with $l = 16$ (5.4.19)

used to complete the integration. This meant that, the initial stepsizes were, unavoidably, much smaller than those needed for the higher order scheme, as such there would be an abrupt increase in stepsizes where the higher order scheme took over. Our experiments tended to show a spike in the local truncation error at this point. While the tolerance threshold would be maintained, this spike often represented the global error for the estimation. There is additional scope for investigation, in the future, to improve the controller's logic.

5.5 Benchmarks

For comparative purposes we will include the built-in Matlab ODE15s in a number of our adaptive stepsize experiments. ODE15s is an implicit solver for stiff systems implemented with backwards differences. It uses a quasi-constant stepsize implementation and it defaults to using 5th order integration. The solver takes advantage of any sparse properties of the Jacobian to accelerate the calculations [43].

5.5.1 Brusselator System

We introduced the Brusselator System in Section 5.1.2 and saw it used in our discussion about local truncation error profiles in Section 5.4.3. We will now run some adaptive stepsize experiments to benchmark EIs.

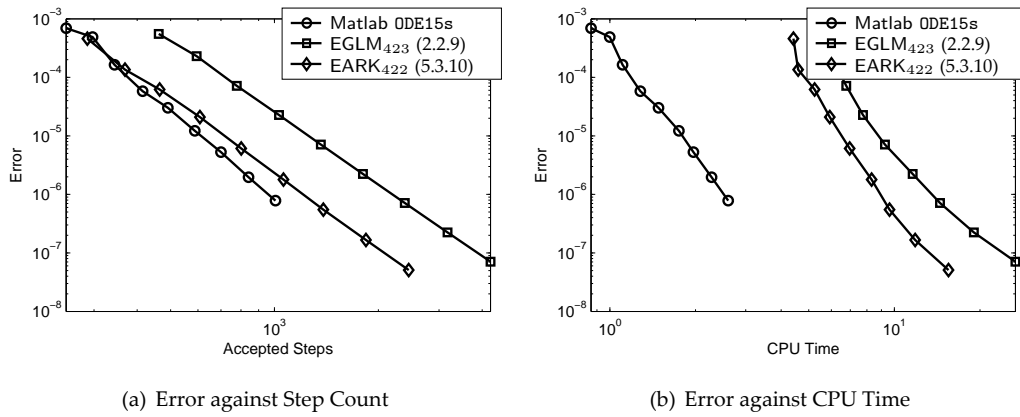


Figure 5.22: Brusselator System, adaptive stepsize, $N = 64$

Figure 5.22 shows that Matlab's ODE15s achieves slightly better step count performance and is faster in CPU costs. The EARK₄₂₂ can be seen to outperform EGLM₄₂₃ by a large margin.

Global Error:	5×10^{-4}	5×10^{-6}	5×10^{-8}
EGLM ₄₂₃	462	1363	4224
EARK ₄₂₂	287	801	2450

Table 5.12: 1D Brusselator System $N = 256$, Step Counts

To get a clearer view of the difference between the EIs we look to Figure 5.23, which does not include ODE15s, and to Table 5.12, which summarizes the step counts at different global errors. We can see that EGLM₄₂₃ requires almost 1.75 times as many steps as EARK₄₂₂ making

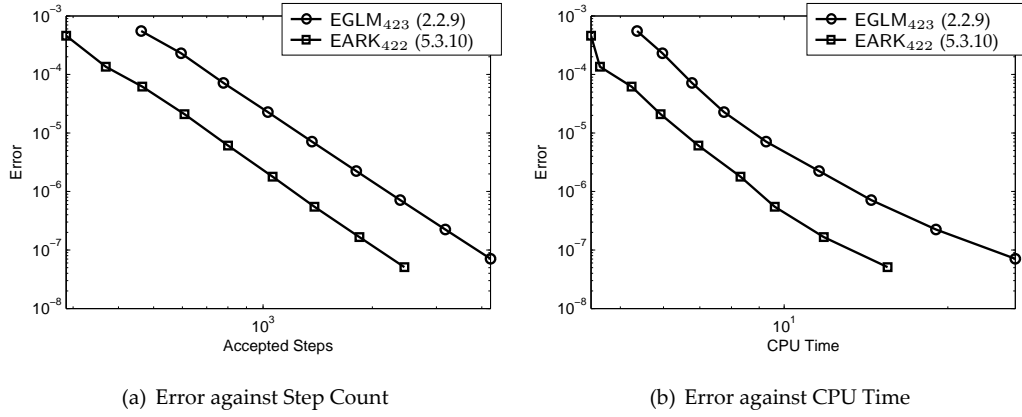


Figure 5.23: Brusselator System, Global Error against CPU Time without ODE15s

EGLM₄₂₃ significantly slower. Appendix B.1 lists the complete integration statistics for three problem sizes, $N = 64, 256$ and 1024 .

5.5.2 Allen-Cahn Equation

Returning to the Allen-Cahn problem from Section 5.1.3 we will now perform some adaptive stepsize experiments against the problem. To include the local truncation error spike (see Figure 5.17(b) at $t = 37$), we run the experiments for $t \in [0, 50]$.

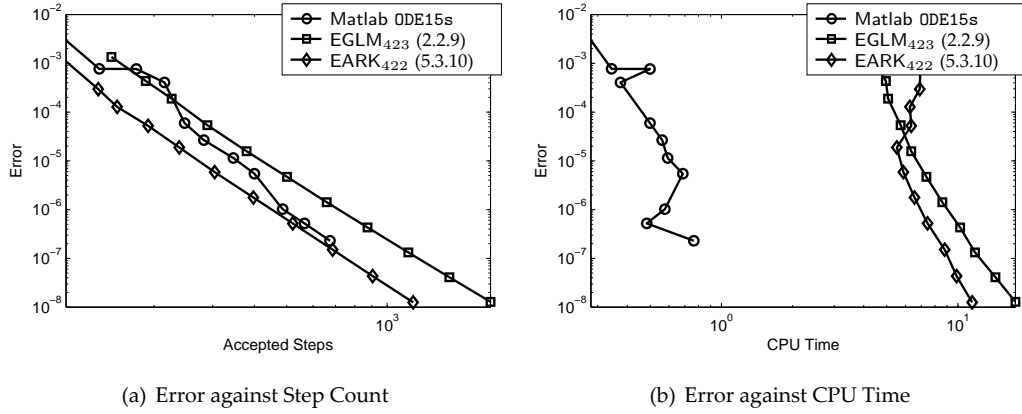


Figure 5.24: Allen-Cahn Equation, adaptive stepsize, $N = 50$, $\varepsilon = 0.001$

Figures 5.24 show the results of an adaptive stepsize experiment. From the results we can see that the step count performance of EARK₄₂₂ is as good as ODE15s. EGLM₄₂₃ is much poorer, needing about 1.75 times more steps.

It is important to highlight that this problem uses a dense, and relatively small, L matrix. It is not the type of problem the underlying `phipm` code is optimised for. Figure 5.25, without

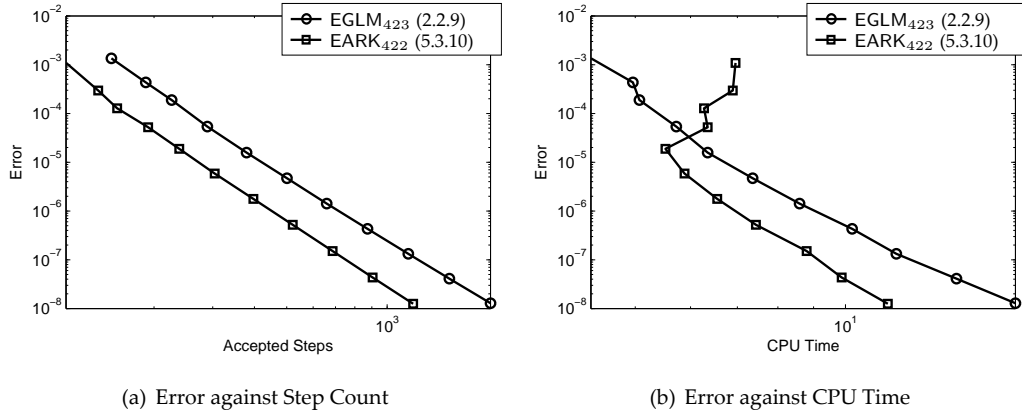


Figure 5.25: Allen-Cahn Equation, adaptive stepsize without ODE15s, $N = 50$, $\varepsilon = 0.001$

ODE15s included, shows the relative performance of the EIs.

Calvo & Portillo [12] preformed variable step experiments with ETDs against the Allen-Cahn problem using a standard three-point finite differences discretisation. To compare our results to those published, we will run some experiments using the same problem discretisation and parameters.

There are however, two possible implementations of the problem. Working with (5.1.6), the term $w(t, x)$ on the right-hand side can be included as part of the linear part L or the non-linear function $N(t, w(t))$. We will perform our experiments twice, once for each implementation. We will see the term “Type 1” when L is a standard three-point finite difference matrix, and “Type 2” for when the $w(t, x)$ is included as part of the L operator.

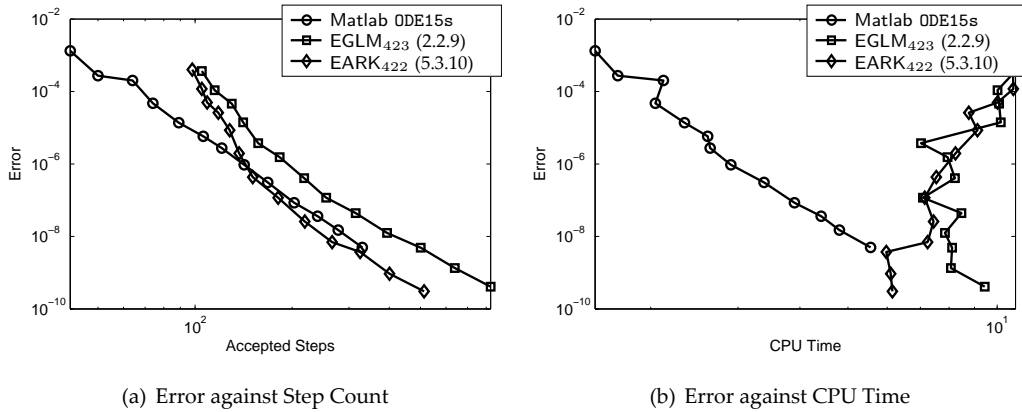


Figure 5.26: Allen-Cahn Equation “Type 1”, $N = 512$, $\varepsilon = 0.001$

For a tolerance of 1×10^{-4} , EGLM₄₂₃ and EARK₄₂₂ take a total of 130 and 109 steps respectively, under the “Type 1” implementation, and 169 and 147 steps for “Type 2”. This is a big improvement over the published 248 steps for ETD3 and 484 for ETD4 [12, Page 635]. For a

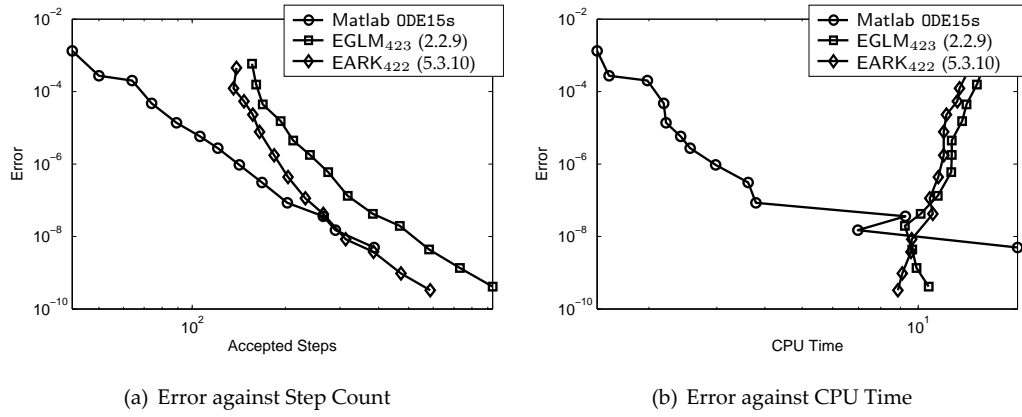


Figure 5.27: Allen-Cahn Equation “Type 2”, $N = 512$, $\varepsilon = 0.001$

tolerance of 1×10^{-8} , EGLM_{423} and EARK_{422} take a total of 315 and 219 steps for “Type 1”, and 383 and 265 steps for “Type 2”. These are significantly better than the published 3105 and 1044 steps for ETD3 and ETD4 [12, Page 635 & 636]. In fact EARK_{422} shows more than a 4-fold improvement over ETD4.

Table 5.13 compares the EIs performance with ODE15s, again it highlights the significant improvement EARK_{422} gives over EGLM_{423} , which requires 1.4 times as many steps under the finite-differences implementations. The tabulated results in Appendix B.2 give the exact step count figures for all experiments.

	Chebyshev	FD “Type 1”	FD “Type 2”
ODE15s	674	168	168
EGLM_{423}	1157	218	275
EARK_{422}	686	151	204

Table 5.13: Allen-Cahn, Global Error = 2×10^{-7} , Step Counts

5.5.3 The Kuramoto-Sivashinsky Equation

The 1D Kuramoto-Sivashinsky Equation was seen in Section 5.1.1 where we performed some fixed stepsize experiments and compared the results with some published in literature. For adaptive stepsize tests, we perform the integration up to $t = 100$. We make this choice after observing (see Figure 5.1) that the solution becomes increasingly chaotic after $t = 40$. A look at local error profile in Figure 5.17(c) shows that over this period a controller must constantly adjust stepsizes and deal with rejected steps.

The results of the adaptive stepsize experiments echo those of the Allen-Cahn problem.

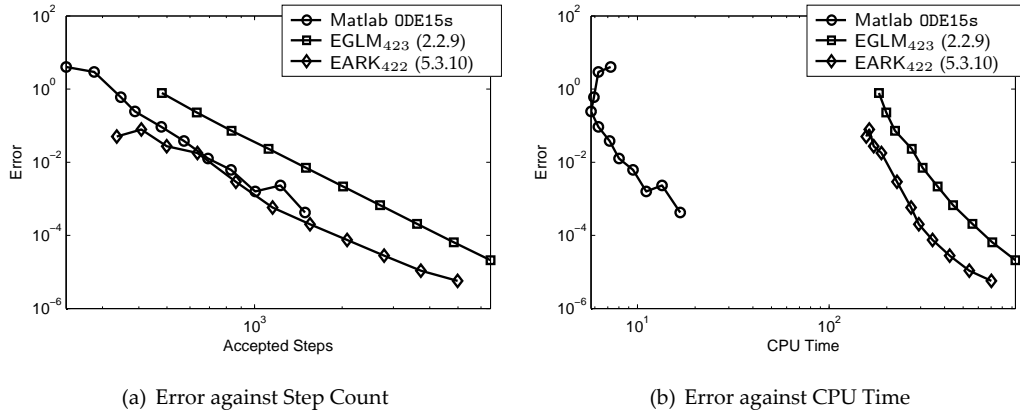


Figure 5.28: Kuramoto-Sivashinsky Equation, $N = 256$

Global Error:	1×10^{-1}	5×10^{-3}	5×10^{-4}
ODE15s	478	1005	1491
EGLM ₄₂₃	833	1501	2701
EARK ₄₂₂	408	862	1551

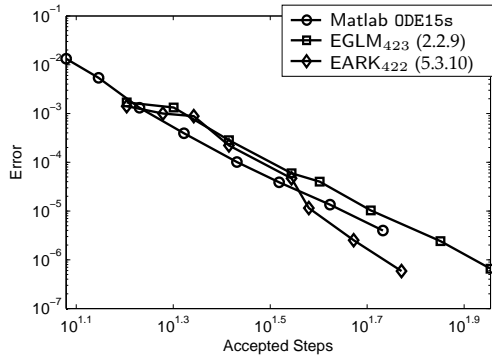
Table 5.14: Kuramoto-Sivashinsky Equation, $N = 256$, Step Counts

Figure 5.28(a) and Table 5.14 demonstrate that the step count performance of EARK₄₂₂ is superior to that of Matlab's ODE15s, we can see that ODE15s requires 1.17 times more steps than EARK₄₂₂ needs to take. EGLM₄₂₃ has fallen much further behind, from the step count performance EGLM₄₂₃ take about 1.75 times more steps than EARK₄₂₂ requires. The CPU Time comparison in Figure 5.28(b) shows that the EAGLM is much better than the EGLM.

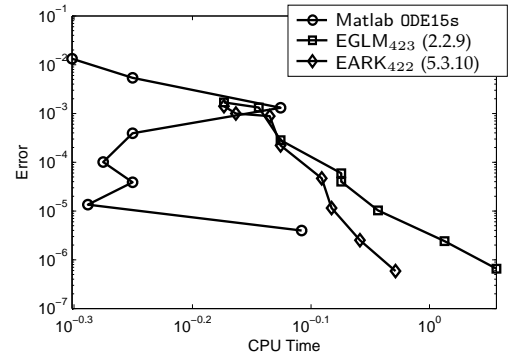
5.5.4 The RDA 2D Equation

The 2D RDA Problem, from Section 5.1.5, produces a very large system of ODEs with a pent-diagonal L matrix. Caliori & Ostermann [10] published the results for some numerical experiments with Rosenbrock-type integrators with the RDA problem. To compare their performance with EAGLMs we will run experiments with the same parameters.

We run a first set of experiments with the parameters $\varepsilon = 0.05$, $\alpha = -1$, $\rho = 1$. Figures 5.29(b) and 5.30(b) illustrate that the step count performance of the three integrators is again largely the same. The CPU comparisons, plotted in Figures 5.29(a) and 5.30(a), are very different from those we saw in the earlier 1D problems. Now ODE15s is significantly slower than either of the EIs. This result clearly highlights the superior performance which EIs schemes can offer for larger problems.

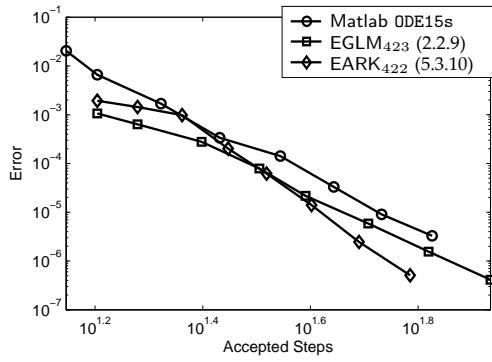


(a) Global Error against Step Count

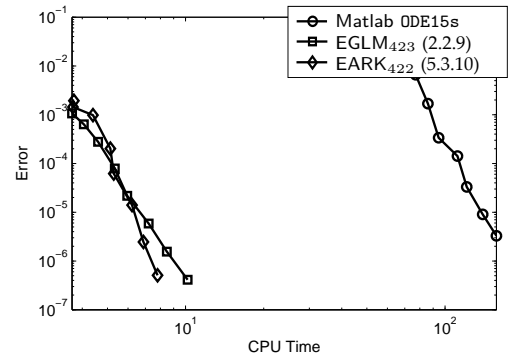


(b) Global Error against CPU Time

Figure 5.29: 2D RDA Equation, $N = 20 \times 20$, $\varepsilon = 0.05$, $\alpha = -1$, $\rho = 1$



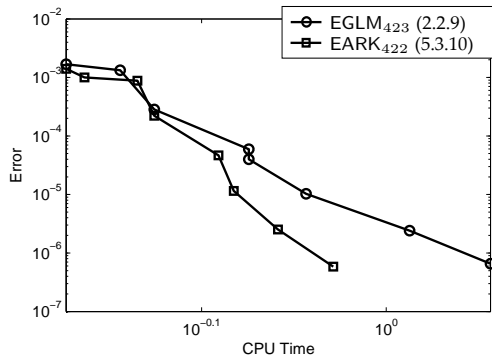
(a) Global Error against Step Count



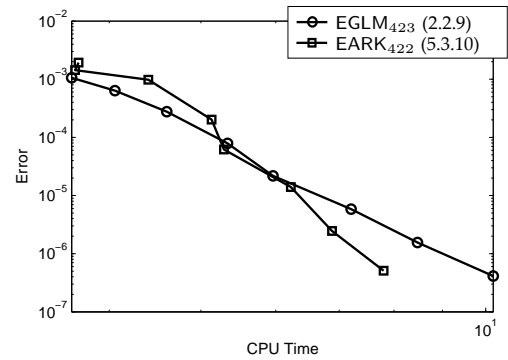
(b) Global Error against CPU Time

Figure 5.30: 2D RDA Equation, $N = 64 \times 64$, $\varepsilon = 0.05$, $\alpha = -1$, $\rho = 1$

Figure 5.31 shows the two “Global Error against CPU Time” plots again, without ODE15s. This makes the difference between EGLM₄₂₃ and EARK₄₂₂ clearer.



(a) $N = 20 \times 20$



(b) $N = 64 \times 64$

Figure 5.31: 2D RDA Equation without ODE15s, $\varepsilon = 0.05$, $\alpha = -1$, $\rho = 1$

The published results from Caliarì & Ostermann [10], while not including any ERKs or

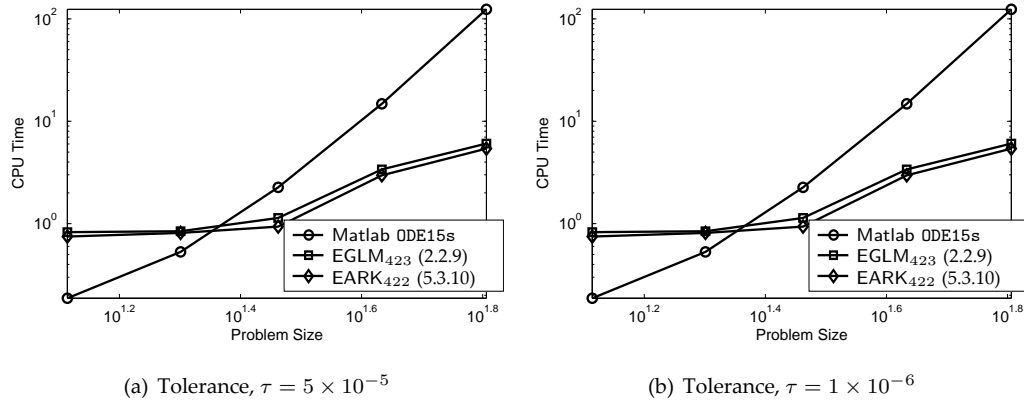


Figure 5.32: 2D RDA Equation, CPU Timings, $\varepsilon = 0.05$, $\alpha = -1$, $\rho = 1$

EGLMs, seem to indicate that Rosenbrock-type integrators can achieve similar levels of accuracy in fewer steps. However, without the inclusion of any established ERKs or EGLMs, it is difficult for us to make a direct comparison with our own results.

To produce a second set of result, we will also repeat the experiments for the parameters $\varepsilon = 0.05$, $\alpha = -1$, $\rho = 100$. Here again we are carrying out a similar experiments to one performed by Caliarì & Ostermann.

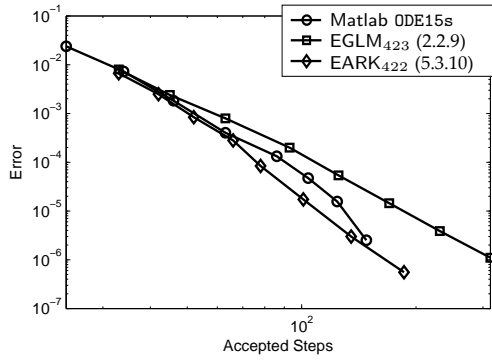
(a) $\rho = 1$				(b) $\rho = 100$			
Global Error:	1×10^{-3}	5×10^{-5}	3×10^{-6}	1×10^{-3}	5×10^{-5}	3×10^{-6}	
ODE15s	21	44	67	46	104	148	
EGLM ₄₂₃	16	39	66	45	125	231	
EARK ₄₂₂	16	33	49	42	78	135	

Table 5.15: 2D RDA Equation, Step Counts, $\varepsilon = 0.05$, $\alpha = -1$

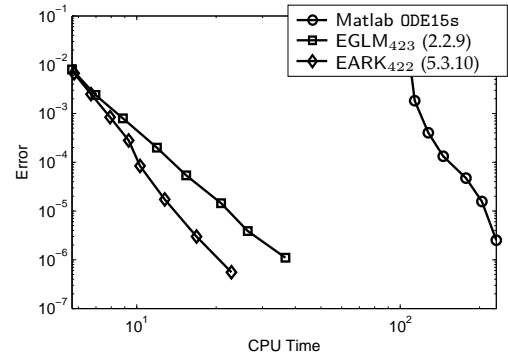
Table 5.15 lists step count statistics. They show that ODE15s takes about 1.3 times more steps than EARK₄₂₂ and EGLM₄₂₃ needs 1.7 times as many.

In Table 5.16 we list the memory required for different problem discretisations. It is clear that memory usage is significantly lower for both the PHIPM and ReLPM approaches to implementing an EI. This is also a crucial result in favour of EIs, as excessive memory requirements can mean integrators such as ODE15s simply cannot be applied to real-world problems in two or more dimensions.

See Appendix B.4 for the full integration statistics across three problem sizes, 32×32 , 45×45 and 64×45 .

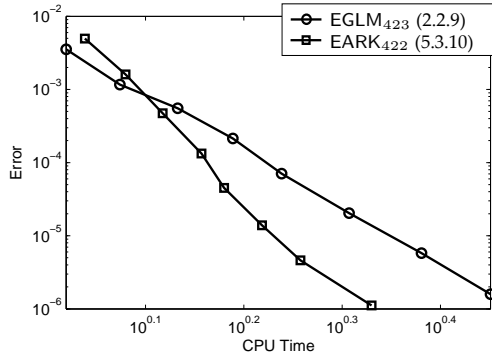


(a) Global Error against Step Count

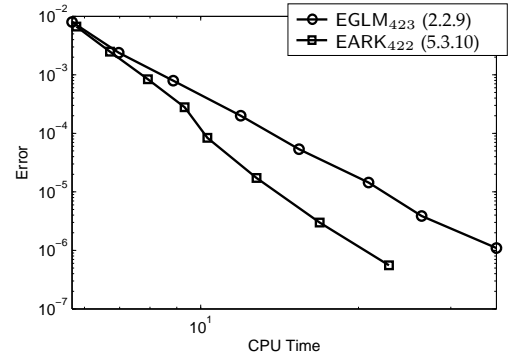


(b) Global Error against CPU Time

Figure 5.33: 2D RDA Equation, $N = 32 \times 32$, $\varepsilon = 0.05$, $\alpha = -1$, $\rho = 100$

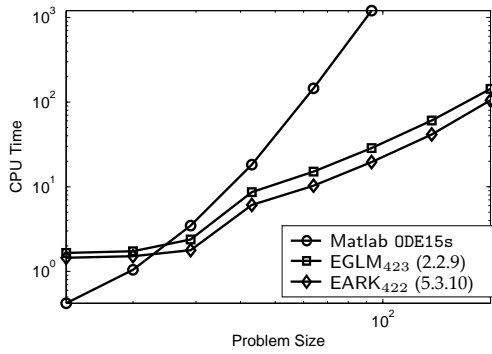


(a) $N = 20 \times 20$

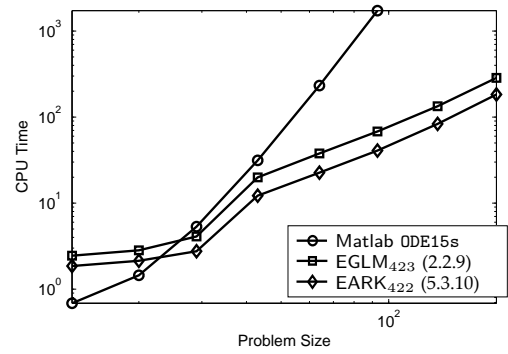


(b) $N = 64 \times 64$

Figure 5.34: 2D RDA Equation without ODE15s, $\varepsilon = 0.05$, $\alpha = -1$, $\rho = 100$



(a) Tolerance, $\tau = 5 \times 10^{-5}$



(b) Tolerance, $\tau = 1 \times 10^{-6}$

Figure 5.35: 2D RDA Equation, CPU Timings, $\varepsilon = 0.05$, $\alpha = -1$, $\rho = 100$

5.5.5 The Gray-Scott Equation

We run a number of numerical experiments with the 1D, 2D and 3D problem implementations provided by the EXPINT package [4, Section 4.2.8]. Due to memory limitations, imposed upon

Dimension	32	64	128	256	512
ODE15s	52	595	3950	N/A	N/A
PHIPM	53	71	170	585	1382
ReLPM	25	63	147	533	1094

Table 5.16: Memory Usage in Megabytes (MBs)

us by Matlab, the largest spacial discretisations we could use were $N = 1024$, $N = 32$ and $N = 8$ for the 1D, 2D and 3D problems respectively.

1D Problem Tests. The L matrix for the Gray-Scott problem is a real diagonal matrix and we ran experiments with the 1D problem for discretisations of $N = 512$ and 1024 giving us L matrices of 1024×1024 and 2048×2048 .

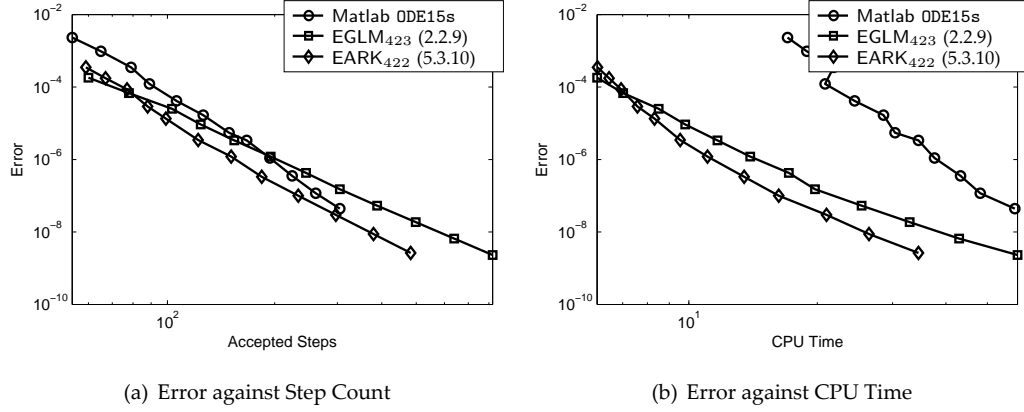


Figure 5.36: 1D Gray-Scott, $N = 512$

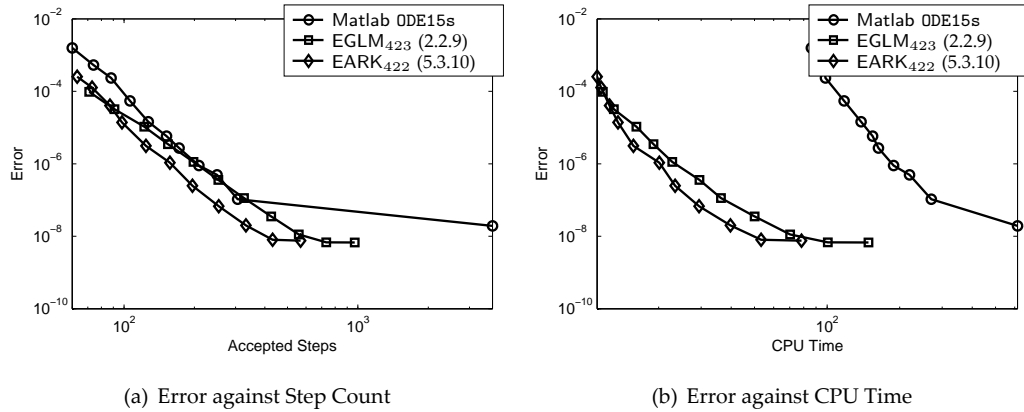


Figure 5.37: 1D Gray-Scott, $N = 1024$

In terms of step count performance (Figures 5.36(a) and 5.37(a)) the three schemes are, again, very similar. Looking at the computational work required per step, the EIs were more

efficient than than ODE15s and so produced faster results. This is in contrast to our earlier 1D Allen-Cahn and Kuramoto-Sivashinsky experiments where ODE15s proved quite efficient.

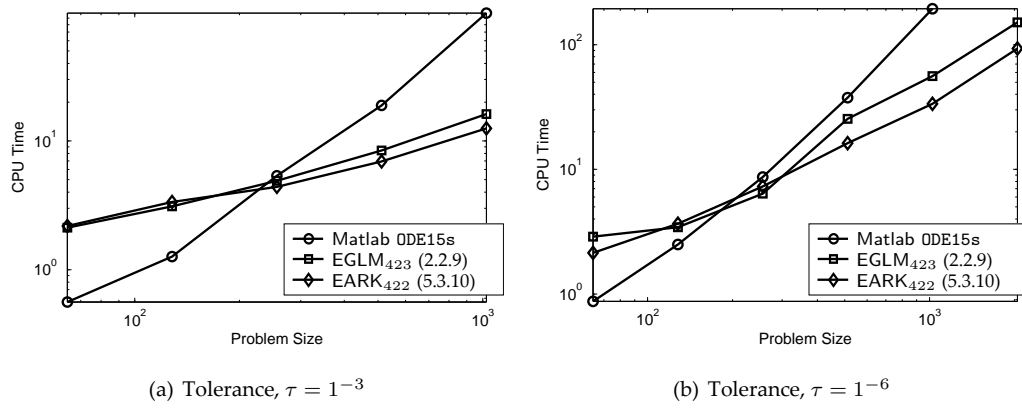


Figure 5.38: 1D Gray-Scott, CPU Timings against Problem Size

In Figures 5.38 we fix the integration tolerance and plot CPU measurement as the problem size increases. Here we observe that, as the size of the problem increases, the CPU timings for the EI schemes increase at a much slower rate over those of ODE15s. Consequently EIs remain practical choices for significantly larger problems.

2D & 3D Problem Tests. Numerical tests with the 2D & 3D Gray-Scott problems repeat the same pattern seen in the 1D tests. For the 2D problem, with $N = 32 \times 32$, we observe from Figure 5.39(a) that step count performances are about equal. In the CPU measurements (Figure 5.39(b)) the EIs are both significantly better than ODE15s, with EARK₄₂₂ again being the top performer.

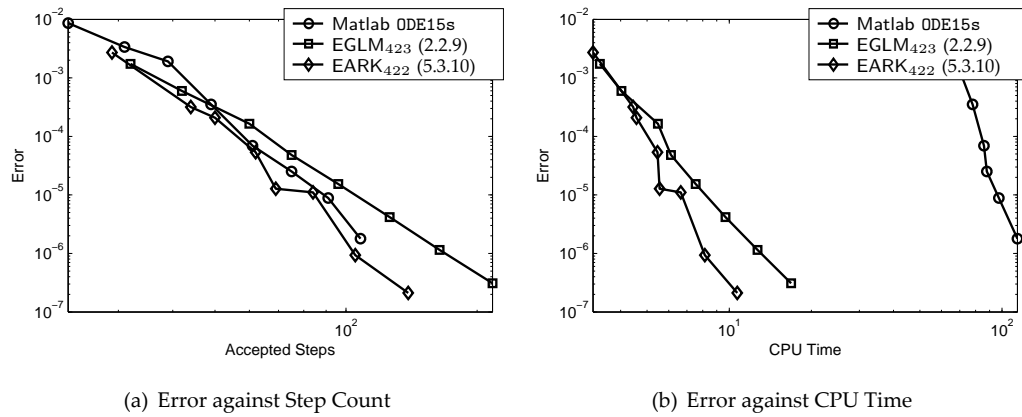


Figure 5.39: 2D Gray-Scott, $N = 32 \times 32$

In the comparison of CPU times against the problem size (Figure 5.40) we can see a much bigger gap between ODE15s and the EIs than the gap observed in the 1D test.

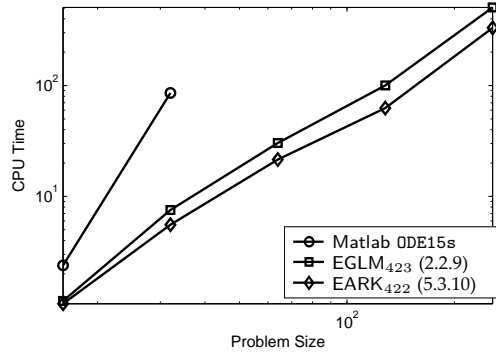


Figure 5.40: 2D Gray-Scott, CPU Timings against Problem Dimension

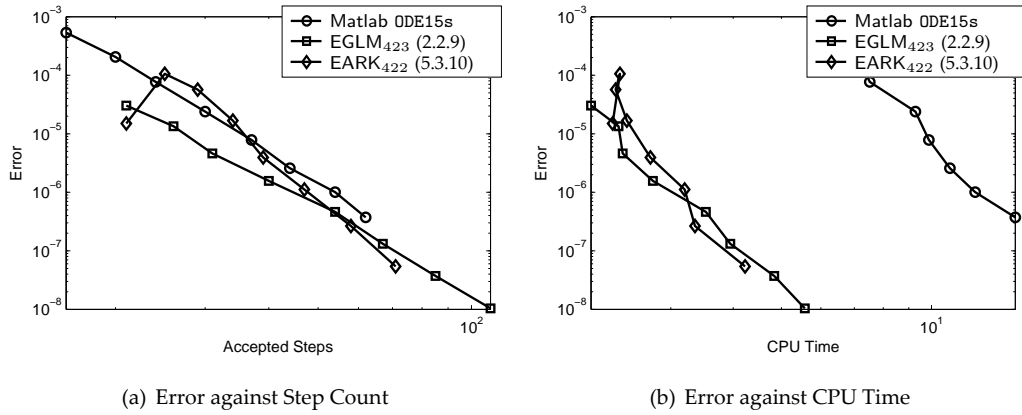


Figure 5.41: 3D Gray-Scott Equation, $n = 8 \times 8 \times 8$

The tests for the 3D problem, shown in Figure 5.41, are more erratic, and this is very likely due to the coarse spatial discretisation. Unfortunately memory constraints meant ODE15s could not be run for a finer mesh. From the CPU timings it is still clear that the EIs are significantly better at solving the problem.

	$N = 512$	$N = 32 \times 32$	$N = 8 \times 8 \times 8$
ODE15s	194	108	54
EGLM ₄₂₃	195	164	40
EARK ₄₂₂	151	105	47

Table 5.17: Gray-Scott Equation, Global Error = 1×10^{-6}

Table 5.17 summarizes some step count statistics for each dimension. On average, the results show that EGLM₄₂₃ consistently requires about 1.6 times as many steps as EARK₄₂₂ takes. Appendices B.5.1, B.5.2 and B.5.3 tabulate the full integration statistics for the experiments.

5.6 ODE15s CPU Cost Scaling

An interesting result observed in some experiments is the rapid growth of the computational cost of ODE15s relative to N , the L -matrix dimension. Figures 5.32 and 5.38 are particularly illustrative of this. To study this behaviour we run some key test problems and measure the relative cost-per-step as N increases.

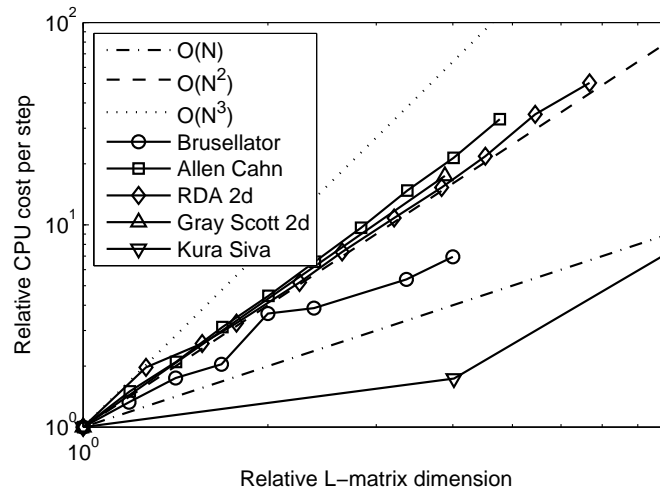


Figure 5.42: Effect of matrix dimension on ODE15s computational cost

The measurements are scaled to align them at the same starting point and then plotted in Figure 5.42, along with reference lines for linear, quadratic and cubic growth rates. The results clearly show that the scaling of ODE15s with respect to the matrix dimension N is almost exactly $O(N^2)$ for most of the problem. For the Kuramoto-Sivashinsky Equation, which has diagonal L -matrix, the graph shows approximately linear scaling for ODE15s.

Chapter 6

Conclusions

From our work we can draw a number of very positive conclusions regarding the EARK family of methods.

- The stability analysis of EARKs showed that they exhibit the same “Runge-Kutta” stability property seen in EGLMs. The large stability region that these two families share with ERKs sets them apart from ETDs and makes them suitable to a wide variety of problems. Like EGLMs, EARKs retain the ETDs’ advantage when it comes to constructing higher-order methods.
- Convergence analysis of EARKs shows that they meet all the same bounds as EGLMs and that existing results proved for EGLMs also hold for EARKs. In addition, we demonstrated an equivalence between EAGLMs and EGLMs when the N function derivatives are constructed using past steps only.
- Having compared accuracy with stepsize over a number of test problems we saw that the 4th order EARKs rank among the top performers out of all the schemes looked at. Only two 4th order ERKs (5.3.5 & 5.3.7) achieve a similar level of performance.
- Looking at CPU cost per step we saw that EARKs are the top performers, matched only by EGLMs when using the `phi_pm` implementation of computing a full linear φ combination in one operation. By comparison, the 4th order ERKs are only half as efficient.

When the $\varphi \times$ vector operations are counted separately, the EARKs are the most efficient schemes in the 3rd and 4th order categories. We also saw that efficiency improvement would be even greater for higher order methods.

- When generating local truncation error approximations, we showed that EARK_{422} (5.3.10) could produce a free estimate from an intermediate result computed as part of a standard implementation of the scheme. Numerical experiments verified that this estimate was robust and very effective at guiding an adaptive stepsize controller.
- Our implementation of an adaptive stepsize integrator was able to combine all of the advantages of EIs and, in particular, EARKs. We carried out a comprehensive comparison of the 4th order EARK with EGLM_{423} (2.2.9) and Matlab's ODE15s. The results showed conclusively that, when comparing step counts, EARK_{422} required the fewest steps to complete the integration of the test problems. Across a range of problems we saw that, on average, EGLM_{423} would require about 1.7 times as many steps as EARK_{422} to achieve the same global error.
- When we looked at large problems, usually of two or more dimensions, we saw that EIs can easily outperform ODE15s with EARK_{422} being the stand-out method.

In the end, it is step count performance where EARKs and EAGLMs demonstrated their superiority over EGLMs and their competitiveness with ODE15s. In terms of computational efficiency we see that simpler structure of an EARK tableau makes them suitable for very efficient implementations.

Appendix A

Definitions

A.1 Parabolic Evolution Equations

The convergence analysis we perform in Section 3.3 occurs in the framework of abstract semilinear parabolic evolution equations. Within this framework we follow the structure of Ostermann, Thälhammer & Wright's proof for [38, Theorem 3.1].

A.1.1 Analytical Framework

Let X be a complex Banach space endowed with the norm $\|\cdot\|_X$ and $D \subset X$ another densely embedded Banach space. For any $0 < v < 1$ we denote by X_v some intermediate space between $D = X_1$ and $X = X_0$ such that the norm in X_v fulfills the relation

$$\|x\|_{X_v} \leq C \|x\|_D^v \|x\|_X^{1-v}, \quad x \in D, \quad 0 < v < 1 \quad (\text{A.1.1})$$

with a constant $C > 0$.

We consider initial value problems of the form (1.3.2) where the right-hand side of the differential equation is defined by a linear operator $L : D \rightarrow X$ and a sufficiently regular nonlinear map

$$N : X_\alpha \rightarrow X : v \mapsto N(v), \quad D \subset X_\alpha \subset X, \quad 0 \leq \alpha < 1 \quad (\text{A.1.2})$$

This requirement together with Hypothesis 17 renders (1.3.2) a semilinear parabolic problem [38].

Hypothesis 17. [38, Hypothesis 3.1] We assume that the closed and densely defined linear operator $L : D \rightarrow X$ is sectorial. Thus, there exist constants $a \in \mathbb{R}$, $0 < \phi < \frac{\pi}{2}$ and ≥ 1 such that L satisfies the resolvent condition

$$\left\| (\lambda I - L)^{-1} \right\|_{X \leftarrow X} \leq \frac{M}{|\lambda - a|}, \quad \lambda \in CS_\phi(a), \quad (\text{A.1.3})$$

on the complement of the sector $S_\phi(a) = \{\lambda \in \mathbb{C} : |\arg(a - \lambda)| \leq \phi\} \cup \{a\}$. Moreover, we suppose that the graph norm of L and the norm in D are equivalent, that is, the estimate

$$C^{-1} \|x\|_D \leq \|x\|_X + \|Lx\|_X \leq C \|x\|_D, \quad x \in D, \quad (\text{A.1.4})$$

is valid for a constant $C > 0$.

Under this Hypothesis, the operator L is the infinitesimal generator of an analytic semi-group $(e^{tL})_{t \geq 0}$ on the underlying Banach space X [38, 25]. For $\omega > -a$, the fractional powers of $\tilde{L} = \omega I - L$ are well-defined [25].

Lemma 18. [25, Lemma 3.1] Under Hypothesis 17 and for fixed $\omega > -a$, the following bounds hold uniformly on $0 \leq t \leq T$.

$$\|e^{tL}\|_{X \leftarrow X} + \|t^\gamma \tilde{L}^\gamma e^{tL}\|_{X \leftarrow X} \leq C, \quad \gamma \geq 0 \quad (\text{A.1.5})$$

$$\left\| hL \sum_{j=1}^{n-1} e^{jhL} \right\|_{X \leftarrow X} \leq C \quad (\text{A.1.6})$$

A.2 B-Convergence

Definition 5 (B-convergent). For an initial value problem (1.1.1) where f satisfies a one-sided Lipschitz condition with a one-sided Lipschitz constant γ and where the solution sought satisfies

$$\left\| \frac{d^i y(t)}{dt^i} \right\| \leq M_i, \quad \forall t \in [0, T] \quad (\text{A.2.1})$$

with bounds M_i , we call a method (producing the sequence $\{\eta_v\}$ with $\eta_v \approx y(t_v)$) *B-convergent* of order p if

$$\|\eta_n - y(t_v)\| \leq C(t_v)h^p, \quad h \in (0, h_0] \quad (\text{A.2.2})$$

holds under the following assumptions:

- The function C depends only on γ (as the only characterisation of f_y), on bounds for certain other derivatives of f ($f_t, f_{yy}, f_{ty}, f_{tt}, \dots$) and on some of the bounds M_i .
- The maximum stepsize h_0 depends on γ and, possibly, on bounds for derivatives of f (except f_y), only.

Definition 6 (B-stable). For a given differential equation (1.1.1) where f satisfies a one-sided Lipschitz condition (5.3.1), with a Lipschitz constant γ , a one-step method is called *B-stable* if an inequality

$$\|\bar{\eta} - \bar{\zeta}\| \leq \phi(h\gamma) \|\eta - \zeta\| \quad (\text{A.2.3})$$

holds, where both $\bar{\eta}$ and $\bar{\zeta}$ are the results (at the point $t + h$) of one step of the method starting from (t, η) and (t, ζ) respectively. $\phi(x)$ is assumed to be a nonnegative monotone increasing function with $\phi(0) = 1$.

Definition 7 (B-consistent). For a given initial value problem (1.1.1), where the solution y satisfies (A.2.1) with bounds M_i , we call a method *B-consistent* of order p if ζ , the result of one step of the method considered (with stepsize h starting from $(t, y(t))$), satisfies

$$\|\zeta - y(t + h)\| \leq Dh^{p+1}, \quad h \in (0, h_0] \quad (\text{A.2.4})$$

D and h_0 are assumed to depend at most on those quantities which are allowed to affect $C(t)$ and h_0 of Definition 5.

Appendix B

Complete Results

B.1 Brusselator System Tables

$N = 64$	Method	Global Error	Steps	CPU Time	Memory
$1.000e - 003$	Matlab ODE 15s	$8.044e - 002$	109	0.499	0.379
$1.000e - 003$	EGLM ₄₂₃ (2.2.9)	$5.092e - 002$	165	7.316	4.813
$1.000e - 003$	EARK ₄₂₂ c_2 (3.2.31)	$3.976e - 002$	128	6.973	0.117
$1.000e - 004$	Matlab ODE 15s	$7.090e - 003$	170	0.686	0.000
$1.000e - 004$	EGLM ₄₂₃ (2.2.9)	$4.728e - 003$	277	5.491	1.063
$1.000e - 004$	EARK ₄₂₂ c_2 (3.2.31)	$5.389e - 003$	184	5.538	0.000
$1.000e - 005$	Matlab ODE 15s	$6.939e - 004$	249	0.858	0.000
$1.000e - 005$	EGLM ₄₂₃ (2.2.9)	$5.507e - 004$	462	5.366	0.738
$1.000e - 005$	EARK ₄₂₂ c_2 (3.2.31)	$4.579e - 004$	287	4.415	-0.031
$1.000e - 006$	Matlab ODE 15s	$1.637e - 004$	344	1.108	0.000
$1.000e - 006$	EGLM ₄₂₃ (2.2.9)	$7.168e - 005$	778	6.770	4.352
$1.000e - 006$	EARK ₄₂₂ c_2 (3.2.31)	$6.213e - 005$	465	5.242	0.000
$1.000e - 007$	Matlab ODE 15s	$3.019e - 005$	491	1.482	0.000
$1.000e - 007$	EGLM ₄₂₃ (2.2.9)	$7.112e - 006$	1363	9.266	4.461
$1.000e - 007$	EARK ₄₂₂ c_2 (3.2.31)	$6.085e - 006$	801	6.958	0.000
$1.000e - 008$	Matlab ODE 15s	$5.278e - 006$	700	1.966	0.000
$1.000e - 008$	EGLM ₄₂₃ (2.2.9)	$7.121e - 007$	2386	14.461	3.422
$1.000e - 008$	EARK ₄₂₂ c_2 (3.2.31)	$5.481e - 007$	1385	9.610	0.000
$1.000e - 009$	Matlab ODE 15s	$7.821e - 007$	1007	2.605	0.000
$1.000e - 009$	EGLM ₄₂₃ (2.2.9)	$7.068e - 008$	4227	26.614	10.473
$1.000e - 009$	EARK ₄₂₂ c_2 (3.2.31)	$5.098e - 008$	2450	15.491	0.000

$N = 256$	Method	Global Error	Steps	CPU Time	Memory
$1.000e - 003$	Matlab ODE 15s	$1.251e - 001$	115	3.401	8.180
$1.000e - 003$	EGLM ₄₂₃ (2.2.9)	$5.094e - 002$	165	41.668	11.277
$1.000e - 003$	EARK _{422C2} (3.2.31)	$3.978e - 002$	128	35.568	9.477
$1.000e - 004$	Matlab ODE 15s	$1.077e - 002$	170	3.838	8.016
$1.000e - 004$	EGLM ₄₂₃ (2.2.9)	$4.720e - 003$	277	44.663	11.145
$1.000e - 004$	EARK _{422C2} (3.2.31)	$5.369e - 003$	184	35.740	11.484
$1.000e - 005$	Matlab ODE 15s	$7.898e - 004$	245	4.961	8.016
$1.000e - 005$	EGLM ₄₂₃ (2.2.9)	$5.503e - 004$	462	40.935	1.070
$1.000e - 005$	EARK _{422C2} (3.2.31)	$4.578e - 004$	287	36.301	19.148
$1.000e - 006$	Matlab ODE 15s	$1.485e - 004$	346	5.897	8.016
$1.000e - 006$	EGLM ₄₂₃ (2.2.9)	$7.167e - 005$	778	43.150	2.988
$1.000e - 006$	EARK _{422C2} (3.2.31)	$6.242e - 005$	465	35.225	12.938
$1.000e - 007$	Matlab ODE 15s	$3.032e - 005$	495	7.847	8.016
$1.000e - 007$	EGLM ₄₂₃ (2.2.9)	$7.111e - 006$	1363	43.368	9.277
$1.000e - 007$	EARK _{422C2} (3.2.31)	$6.087e - 006$	801	42.136	6.543
$1.000e - 008$	Matlab ODE 15s	$5.190e - 006$	708	10.234	8.016
$1.000e - 008$	EGLM ₄₂₃ (2.2.9)	$7.120e - 007$	2386	39.796	31.598
$1.000e - 008$	EARK _{422C2} (3.2.31)	$5.483e - 007$	1385	36.036	20.738
$1.000e - 009$	Matlab ODE 15s	$7.741e - 007$	1030	13.853	8.016
$1.000e - 009$	EGLM ₄₂₃ (2.2.9)	$7.066e - 008$	4224	76.331	55.988
$1.000e - 009$	EARK _{422C2} (3.2.31)	$5.091e - 008$	2450	38.875	49.129

$N = 1024$	Method	Global Error	Steps	CPU Time	Memory
$1.000e - 003$	Matlab ODE 15s	$1.249e - 001$	115	70.809	134.992
$1.000e - 003$	EGLM ₄₂₃ (2.2.9)	$5.094e - 002$	165	941.700	21.777
$1.000e - 003$	EARK _{422C2} (3.2.31)	$3.979e - 002$	128	952.074	20.113
$1.000e - 004$	Matlab ODE 15s	$1.077e - 002$	170	83.195	130.164
$1.000e - 004$	EGLM ₄₂₃ (2.2.9)	$4.720e - 003$	277	1060.027	24.684
$1.000e - 004$	EARK _{422C2} (3.2.31)	$5.368e - 003$	184	984.132	22.371
$1.000e - 005$	Matlab ODE 15s	$1.215e - 003$	245	104.177	131.234
$1.000e - 005$	EGLM ₄₂₃ (2.2.9)	$5.503e - 004$	462	1167.480	26.129
$1.000e - 005$	EARK _{422C2} (3.2.31)	$4.578e - 004$	287	1136.405	39.480
$1.000e - 006$	Matlab ODE 15s	$1.482e - 004$	346	123.958	132.691
$1.000e - 006$	EGLM ₄₂₃ (2.2.9)	$7.167e - 005$	778	1311.344	58.598
$1.000e - 006$	EARK _{422C2} (3.2.31)	$6.242e - 005$	465	1350.547	45.316
$1.000e - 007$	Matlab ODE 15s	$2.787e - 005$	502	167.077	135.172
$1.000e - 007$	EGLM ₄₂₃ (2.2.9)	$7.111e - 006$	1363	1760.487	124.637
$1.000e - 007$	EARK _{422C2} (3.2.31)	$6.087e - 006$	801	1654.001	121.223
$1.000e - 008$	Matlab ODE 15s	$4.800e - 006$	715	217.996	138.879
$1.000e - 008$	EGLM ₄₂₃ (2.2.9)	N/A	N/A	N/A	229.078
$1.000e - 008$	EARK _{422C2} (3.2.31)	$5.482e - 007$	1385	1974.349	275.551

B.2 Allen-Cahn Equation Tables

B.2.1 Chebyshev Differentiation Matrix

$N = 25$	Method	Global Error	Steps	CPU Time	Memory
$1.000e-003$	Matlab ODE 15s	$8.673e-002$	73	0.062	0.000
$1.000e-003$	EGLM ₄₂₃ (2.2.9)	$6.638e-003$	106	2.137	19.770
$1.000e-003$	EARK ₄₂₂ C_2 (3.2.31)	$5.545e-004$	97	2.044	0.000
$1.000e-004$	Matlab ODE 15s	$9.645e-003$	112	0.078	0.000
$1.000e-004$	EGLM ₄₂₃ (2.2.9)	$2.356e-003$	150	2.480	0.133
$1.000e-004$	EARK ₄₂₂ C_2 (3.2.31)	$3.195e-003$	115	2.122	0.000
$1.000e-005$	Matlab ODE 15s	$1.071e-003$	172	0.109	0.000
$1.000e-005$	EGLM ₄₂₃ (2.2.9)	$3.648e-004$	237	3.151	0.492
$1.000e-005$	EARK ₄₂₂ C_2 (3.2.31)	$2.773e-004$	159	2.527	0.000
$1.000e-006$	Matlab ODE 15s	$8.323e-005$	240	0.140	0.000
$1.000e-006$	EGLM ₄₂₃ (2.2.9)	$5.680e-005$	383	4.087	1.980
$1.000e-006$	EARK ₄₂₂ C_2 (3.2.31)	$4.788e-005$	242	3.136	0.000
$1.000e-007$	Matlab ODE 15s	$6.987e-006$	334	0.187	0.000
$1.000e-007$	EGLM ₄₂₃ (2.2.9)	$5.389e-006$	655	5.694	1.289
$1.000e-007$	EARK ₄₂₂ C_2 (3.2.31)	$4.573e-006$	399	4.181	0.000
$1.000e-008$	Matlab ODE 15s	$4.250e-007$	460	0.250	0.000
$1.000e-008$	EGLM ₄₂₃ (2.2.9)	$5.239e-007$	1152	7.894	4.039
$1.000e-008$	EARK ₄₂₂ C_2 (3.2.31)	$4.254e-007$	683	6.006	-0.031
$1.000e-009$	Matlab ODE 15s	$7.100e-008$	646	0.343	0.000
$1.000e-009$	EGLM ₄₂₃ (2.2.9)	$5.163e-008$	2035	11.014	14.949
$1.000e-009$	EARK ₄₂₂ C_2 (3.2.31)	$3.931e-008$	1193	8.377	0.039

$N = 50$	Method	Global Error	Steps	CPU Time	Memory
$1.000e-003$	Matlab ODE 15s	$3.185e-002$	69	0.437	-0.027
$1.000e-003$	EGLM ₄₂₃ (2.2.9)	$6.161e-003$	104	4.430	0.457
$1.000e-003$	EARK ₄₂₂ C_2 (3.2.31)	$1.075e-003$	97	8.112	0.000
$1.000e-004$	Matlab ODE 15s	$2.941e-003$	109	0.281	0.000
$1.000e-004$	EGLM ₄₂₃ (2.2.9)	$1.351e-003$	149	4.321	0.000
$1.000e-004$	EARK ₄₂₂ C_2 (3.2.31)	$1.088e-003$	109	6.958	-0.023
$1.000e-005$	Matlab ODE 15s	$7.636e-004$	177	0.499	0.000
$1.000e-005$	EGLM ₄₂₃ (2.2.9)	$1.879e-004$	226	5.070	-0.020
$1.000e-005$	EARK ₄₂₂ C_2 (3.2.31)	$1.275e-004$	155	6.271	0.230
$1.000e-006$	Matlab ODE 15s	$5.941e-005$	247	0.499	0.000
$1.000e-006$	EGLM ₄₂₃ (2.2.9)	$1.576e-005$	379	6.349	-0.020
$1.000e-006$	EARK ₄₂₂ C_2 (3.2.31)	$1.884e-005$	238	5.522	-0.020
$1.000e-007$	Matlab ODE 15s	$1.144e-005$	346	0.593	0.000
$1.000e-007$	EGLM ₄₂₃ (2.2.9)	$1.414e-006$	659	8.596	0.227
$1.000e-007$	EARK ₄₂₂ C_2 (3.2.31)	$1.776e-006$	397	6.552	-0.016
$1.000e-008$	Matlab ODE 15s	$1.019e-006$	486	0.577	0.000
$1.000e-008$	EGLM ₄₂₃ (2.2.9)	$1.323e-007$	1157	11.825	13.738
$1.000e-008$	EARK ₄₂₂ C_2 (3.2.31)	$1.507e-007$	686	8.798	0.000
$1.000e-009$	Matlab ODE 15s	$2.290e-007$	674	0.764	0.000
$1.000e-009$	EGLM ₄₂₃ (2.2.9)	$1.281e-008$	2041	17.503	68.082
$1.000e-009$	EARK ₄₂₂ C_2 (3.2.31)	$1.254e-008$	1197	11.497	0.000

B.2.2 Finite-Differences

Type 1

$N = 512$	Method	Global Error	Steps	CPU Time	Memory
$1.000e - 003$	Matlab ODE 15s	$1.329e - 003$	41	1.513	8.277
$1.000e - 003$	EGLM ₄₂₃ (2.2.9)	$3.659e - 004$	105	10.702	35.336
$1.000e - 003$	EARK ₄₂₂ c_2 (3.2.31)	$4.007e - 004$	98	10.265	4.617
$1.000e - 004$	Matlab ODE 15s	$2.010e - 004$	64	1.700	7.984
$1.000e - 004$	EGLM ₄₂₃ (2.2.9)	$4.611e - 005$	130	10.670	0.000
$1.000e - 004$	EARK ₄₂₂ c_2 (3.2.31)	$4.979e - 005$	109	9.828	2.688
$1.000e - 005$	Matlab ODE 15s	$1.376e - 005$	89	2.262	7.984
$1.000e - 005$	EGLM ₄₂₃ (2.2.9)	$3.737e - 006$	157	6.895	0.000
$1.000e - 005$	EARK ₄₂₂ c_2 (3.2.31)	$8.518e - 006$	128	8.752	1.176
$1.000e - 006$	Matlab ODE 15s	$2.731e - 006$	121	2.761	7.984
$1.000e - 006$	EGLM ₄₂₃ (2.2.9)	$4.051e - 007$	218	8.346	0.000
$1.000e - 006$	EARK ₄₂₂ c_2 (3.2.31)	$4.352e - 007$	151	7.457	11.039
$1.000e - 007$	Matlab ODE 15s	$3.094e - 007$	168	3.463	9.547
$1.000e - 007$	EGLM ₄₂₃ (2.2.9)	$4.402e - 008$	315	8.299	1.766
$1.000e - 007$	EARK ₄₂₂ c_2 (3.2.31)	$2.581e - 008$	219	7.566	2.184
$1.000e - 008$	Matlab ODE 15s	$3.616e - 008$	240	4.103	7.984
$1.000e - 008$	EGLM ₄₂₃ (2.2.9)	$4.867e - 009$	501	8.923	0.004
$1.000e - 008$	EARK ₄₂₂ c_2 (3.2.31)	$3.730e - 009$	325	6.443	0.000
$1.000e - 009$	Matlab ODE 15s	$4.958e - 009$	330	5.585	7.984
$1.000e - 009$	EGLM ₄₂₃ (2.2.9)	$4.105e - 010$	825	9.766	0.012
$1.000e - 009$	EARK ₄₂₂ c_2 (3.2.31)	$3.040e - 010$	513	6.365	0.000

Type 2

$N = 512$	Method	Global Error	Steps	CPU Time	Memory
$1.000e - 003$	Matlab ODE 15s	$1.329e - 003$	41	1.466	8.234
$1.000e - 003$	EGLM ₄₂₃ (2.2.9)	$5.877e - 004$	156	14.898	3.449
$1.000e - 003$	EARK ₄₂₂ c_2 (3.2.31)	$4.502e - 004$	139	13.619	3.543
$1.000e - 004$	Matlab ODE 15s	$2.010e - 004$	64	1.981	7.984
$1.000e - 004$	EGLM ₄₂₃ (2.2.9)	$4.412e - 005$	169	13.354	0.000
$1.000e - 004$	EARK ₄₂₂ c_2 (3.2.31)	$5.402e - 005$	147	12.636	0.000
$1.000e - 005$	Matlab ODE 15s	$1.376e - 005$	89	2.215	7.984
$1.000e - 005$	EGLM ₄₂₃ (2.2.9)	$4.459e - 006$	212	12.230	2.473
$1.000e - 005$	EARK ₄₂₂ c_2 (3.2.31)	$7.705e - 006$	165	11.653	0.000
$1.000e - 006$	Matlab ODE 15s	$2.731e - 006$	121	2.558	7.984
$1.000e - 006$	EGLM ₄₂₃ (2.2.9)	$5.990e - 007$	275	12.184	5.883
$1.000e - 006$	EARK ₄₂₂ c_2 (3.2.31)	$4.386e - 007$	204	11.279	-0.008
$1.000e - 007$	Matlab ODE 15s	$3.094e - 007$	168	3.619	7.984
$1.000e - 007$	EGLM ₄₂₃ (2.2.9)	$4.220e - 008$	383	10.156	-0.008
$1.000e - 007$	EARK ₄₂₂ c_2 (3.2.31)	$4.222e - 008$	265	10.920	1.160
$1.000e - 008$	Matlab ODE 15s	$3.616e - 008$	265	9.266	7.984
$1.000e - 008$	EGLM ₄₂₃ (2.2.9)	$4.396e - 009$	582	9.656	0.000
$1.000e - 008$	EARK ₄₂₂ c_2 (3.2.31)	$3.754e - 009$	385	9.578	4.012
$1.000e - 009$	Matlab ODE 15s	$4.958e - 009$	387	18.112	7.984
$1.000e - 009$	EGLM ₄₂₃ (2.2.9)	$4.142e - 010$	933	10.655	0.000
$1.000e - 009$	EARK ₄₂₂ c_2 (3.2.31)	$3.289e - 010$	587	8.861	13.516

B.3 1D Kuramoto-Sivashinsky Equation Tables

$N = 128$	Method	Global Error	Steps	CPU Time	Memory
$1.000e - 002$	Matlab ODE 15s	$4.855e + 000$	207	2.387	1.516
$1.000e - 002$	EGLM ₄₂₃ (2.2.9)	$2.076e + 000$	405	46.098	30.363
$1.000e - 002$	EARK ₄₂₂ c_2 (3.2.31)	$4.096e - 001$	303	38.345	2.781
$1.000e - 003$	Matlab ODE 15s	$2.045e + 000$	300	1.763	0.000
$1.000e - 003$	EGLM ₄₂₃ (2.2.9)	$1.441e - 001$	703	65.505	17.664
$1.000e - 003$	EARK ₄₂₂ c_2 (3.2.31)	$6.680e - 002$	438	46.566	0.488
$1.000e - 004$	Matlab ODE 15s	$1.918e - 002$	447	2.309	0.000
$1.000e - 004$	EGLM ₄₂₃ (2.2.9)	$1.431e - 002$	1256	99.092	44.621
$1.000e - 004$	EARK ₄₂₂ c_2 (3.2.31)	$8.995e - 003$	718	63.196	0.285
$1.000e - 005$	Matlab ODE 15s	$7.674e - 003$	632	3.089	0.000
$1.000e - 005$	EGLM ₄₂₃ (2.2.9)	$1.365e - 003$	2265	163.083	4.797
$1.000e - 005$	EARK ₄₂₂ c_2 (3.2.31)	$3.633e - 004$	1298	101.791	7.438
$1.000e - 006$	Matlab ODE 15s	$3.645e - 003$	925	4.415	0.000
$1.000e - 006$	EGLM ₄₂₃ (2.2.9)	$1.296e - 004$	4061	292.845	8.309
$1.000e - 006$	EARK ₄₂₂ c_2 (3.2.31)	$5.010e - 005$	2339	170.727	4.395
$1.000e - 007$	Matlab ODE 15s	$1.757e - 003$	1352	6.224	0.000
$1.000e - 007$	EGLM ₄₂₃ (2.2.9)	$1.360e - 005$	7260	224.735	36.082
$1.000e - 007$	EARK ₄₂₂ c_2 (3.2.31)	$7.982e - 006$	4187	303.672	10.184

$N = 256$	Method	Global Error	Steps	CPU Time	Memory
$1.000e - 002$	Matlab ODE 15s	$4.017e + 000$	225	7.254	4.645
$1.000e - 002$	EGLM ₄₂₃ (2.2.9)	$7.816e - 001$	480	181.882	79.352
$1.000e - 002$	EARK ₄₂₂ c_2 (3.2.31)	$5.022e - 002$	336	156.235	13.266
$1.000e - 003$	Matlab ODE 15s	$5.986e - 001$	347	5.928	0.000
$1.000e - 003$	EGLM ₄₂₃ (2.2.9)	$7.230e - 002$	833	220.585	4.117
$1.000e - 003$	EARK ₄₂₂ c_2 (3.2.31)	$2.735e - 002$	499	170.681	0.602
$1.000e - 004$	Matlab ODE 15s	$9.267e - 002$	478	6.256	0.000
$1.000e - 004$	EGLM ₄₂₃ (2.2.9)	$7.068e - 003$	1501	307.322	12.801
$1.000e - 004$	EARK ₄₂₂ c_2 (3.2.31)	$2.910e - 003$	862	225.827	9.578
$1.000e - 005$	Matlab ODE 15s	$1.265e - 002$	692	8.018	0.000
$1.000e - 005$	EGLM ₄₂₃ (2.2.9)	$6.672e - 004$	2701	443.261	16.035
$1.000e - 005$	EARK ₄₂₂ c_2 (3.2.31)	$2.016e - 004$	1551	293.438	10.445
$1.000e - 006$	Matlab ODE 15s	$1.585e - 003$	1005	11.107	0.000
$1.000e - 006$	EGLM ₄₂₃ (2.2.9)	$6.455e - 005$	4839	710.132	50.551
$1.000e - 006$	EARK ₄₂₂ c_2 (3.2.31)	$2.788e - 005$	2789	425.134	44.480
$1.000e - 007$	Matlab ODE 15s	$4.208e - 004$	1491	16.708	0.000
$1.000e - 007$	EGLM ₄₂₃ (2.2.9)	N/A	N/A	N/A	138.156
$1.000e - 007$	EARK ₄₂₂ c_2 (3.2.31)	$5.724e - 006$	4986	701.817	192.371

B.4 RDA 2D Equation Tables

Parameters $\varepsilon = 0.05, \alpha = -1, \rho = 1$

$N = 32 \times 32$	Method	Global Error	Steps	CPU Time	Memory
$1.000e-002$	Matlab ODE 15s	$1.183e-002$	12	2.387	24.734
$1.000e-002$	EGLM ₄₂₃ (2.2.9)	$1.107e-003$	16	1.841	3.133
$1.000e-002$	EARK _{422C2} (3.2.31)	$2.996e-003$	16	1.841	0.000
$2.683e-003$	Matlab ODE 15s	$3.986e-003$	14	2.574	24.734
$2.683e-003$	EGLM ₄₂₃ (2.2.9)	$1.041e-003$	19	1.544	0.066
$2.683e-003$	EARK _{422C2} (3.2.31)	$1.169e-003$	19	1.685	0.895
$7.197e-004$	Matlab ODE 15s	$1.079e-003$	17	2.652	24.734
$7.197e-004$	EGLM ₄₂₃ (2.2.9)	$3.936e-004$	25	1.716	0.066
$7.197e-004$	EARK _{422C2} (3.2.31)	$7.495e-004$	24	1.529	0.895
$1.931e-004$	Matlab ODE 15s	$3.363e-004$	22	3.011	24.734
$1.931e-004$	EGLM ₄₂₃ (2.2.9)	$1.029e-004$	31	1.872	0.066
$1.931e-004$	EARK _{422C2} (3.2.31)	$2.497e-004$	26	1.622	0.895
$5.179e-005$	Matlab ODE 15s	$9.418e-005$	29	3.338	24.734
$5.179e-005$	EGLM ₄₂₃ (2.2.9)	$2.650e-005$	39	1.607	0.066
$5.179e-005$	EARK _{422C2} (3.2.31)	$7.556e-005$	32	1.825	1.031
$1.389e-005$	Matlab ODE 15s	$2.672e-005$	35	3.635	24.734
$1.389e-005$	EGLM ₄₂₃ (2.2.9)	$6.701e-006$	50	1.342	0.027
$1.389e-005$	EARK _{422C2} (3.2.31)	$1.199e-005$	38	1.264	-0.039
$3.728e-006$	Matlab ODE 15s	$1.212e-005$	44	4.384	24.734
$3.728e-006$	EGLM ₄₂₃ (2.2.9)	$1.699e-006$	65	1.591	2.059
$3.728e-006$	EARK _{422C2} (3.2.31)	$2.102e-006$	47	1.295	0.000
$1.000e-006$	Matlab ODE 15s	$4.370e-006$	56	5.054	24.734
$1.000e-006$	EGLM ₄₂₃ (2.2.9)	$4.365e-007$	86	1.950	0.031
$1.000e-006$	EARK _{422C2} (3.2.31)	$4.906e-007$	59	1.435	0.895

$N = 64 \times 64$	Method	Global Error	Steps	CPU Time	Memory
$1.000e-002$	Matlab ODE 15s	$2.046e-002$	14	75.957	450.953
$1.000e-002$	EGLM ₄₂₃ (2.2.9)	$1.061e-003$	16	3.650	6.309
$1.000e-002$	EARK _{422C2} (3.2.31)	$1.931e-003$	16	3.713	5.516
$2.683e-003$	Matlab ODE 15s	$6.592e-003$	16	77.018	450.723
$2.683e-003$	EGLM ₄₂₃ (2.2.9)	$6.359e-004$	19	4.056	3.625
$2.683e-003$	EARK _{422C2} (3.2.31)	$1.434e-003$	19	3.682	3.348
$7.197e-004$	Matlab ODE 15s	$1.688e-003$	21	85.941	450.391
$7.197e-004$	EGLM ₄₂₃ (2.2.9)	$2.772e-004$	25	4.602	2.172
$7.197e-004$	EARK _{422C2} (3.2.31)	$9.786e-004$	23	4.399	1.855
$1.931e-004$	Matlab ODE 15s	$3.353e-004$	27	94.537	450.953
$1.931e-004$	EGLM ₄₂₃ (2.2.9)	$7.885e-005$	32	5.335	1.695
$1.931e-004$	EARK _{422C2} (3.2.31)	$2.018e-004$	28	5.132	2.695
$5.179e-005$	Matlab ODE 15s	$1.413e-004$	35	111.744	450.953
$5.179e-005$	EGLM ₄₂₃ (2.2.9)	$2.173e-005$	39	5.959	2.027
$5.179e-005$	EARK _{422C2} (3.2.31)	$6.194e-005$	33	5.288	3.566
$1.389e-005$	Matlab ODE 15s	$3.305e-005$	44	121.338	452.273
$1.389e-005$	EGLM ₄₂₃ (2.2.9)	$5.848e-006$	51	7.207	3.145
$1.389e-005$	EARK _{422C2} (3.2.31)	$1.401e-005$	40	6.224	3.020
$3.728e-006$	Matlab ODE 15s	$9.051e-006$	54	139.746	452.363
$3.728e-006$	EGLM ₄₂₃ (2.2.9)	$1.557e-006$	66	8.471	3.563
$3.728e-006$	EARK _{422C2} (3.2.31)	$2.455e-006$	49	6.880	4.824
$1.000e-006$	Matlab ODE 15s	$3.291e-006$	67	157.717	452.160
$1.000e-006$	EGLM ₄₂₃ (2.2.9)	$4.124e-007$	86	10.187	4.879
$1.000e-006$	EARK _{422C2} (3.2.31)	$5.093e-007$	61	7.800	3.980

Parameters $\varepsilon = 0.05, \alpha = -1, \rho = 100$

$N = 32 \times 32$	Method	Global Error	Steps	CPU Time	Memory
$1.000e-002$	Matlab ODE 15s	$2.915e-002$	24	3.229	24.734
$1.000e-002$	EGLM ₄₂₃ (2.2.9)	$7.871e-003$	34	1.420	1.500
$1.000e-002$	EARK _{422C2} (3.2.31)	$6.968e-003$	32	2.402	0.000
$2.683e-003$	Matlab ODE 15s	$8.985e-003$	35	3.572	24.734
$2.683e-003$	EGLM ₄₂₃ (2.2.9)	$2.732e-003$	47	1.560	0.066
$2.683e-003$	EARK _{422C2} (3.2.31)	$2.092e-003$	48	1.576	0.664
$7.197e-004$	Matlab ODE 15s	$2.499e-003$	46	3.962	24.734
$7.197e-004$	EGLM ₄₂₃ (2.2.9)	$9.814e-004$	66	1.810	0.066
$7.197e-004$	EARK _{422C2} (3.2.31)	$6.232e-004$	61	1.622	1.402
$1.931e-004$	Matlab ODE 15s	$5.513e-004$	63	4.368	24.734
$1.931e-004$	EGLM ₄₂₃ (2.2.9)	$2.625e-004$	97	1.888	3.055
$1.931e-004$	EARK _{422C2} (3.2.31)	$2.212e-004$	69	1.622	-0.016
$5.179e-005$	Matlab ODE 15s	$1.730e-004$	84	4.976	24.820
$5.179e-005$	EGLM ₄₂₃ (2.2.9)	$7.349e-005$	131	2.262	12.176
$5.179e-005$	EARK _{422C2} (3.2.31)	$6.362e-005$	86	1.872	0.000
$1.389e-005$	Matlab ODE 15s	$9.893e-005$	115	6.349	24.734
$1.389e-005$	EGLM ₄₂₃ (2.2.9)	$2.019e-005$	179	2.824	2.887
$1.389e-005$	EARK _{422C2} (3.2.31)	$1.556e-005$	106	2.044	2.738
$3.728e-006$	Matlab ODE 15s	$4.355e-005$	129	6.770	24.734
$3.728e-006$	EGLM ₄₂₃ (2.2.9)	$5.505e-006$	244	3.245	1.695
$3.728e-006$	EARK _{422C2} (3.2.31)	$3.165e-006$	143	2.434	2.199
$1.000e-006$	Matlab ODE 15s	$6.900e-006$	150	7.831	26.129
$1.000e-006$	EGLM ₄₂₃ (2.2.9)	$1.489e-006$	335	4.290	1.605
$1.000e-006$	EARK _{422C2} (3.2.31)	$5.870e-007$	196	2.995	2.770

$N = 64 \times 64$	Method	Global Error	Steps	CPU Time	Memory
$1.000e-002$	Matlab ODE 15s	$2.381e-002$	24	83.835	453.063
$1.000e-002$	EGLM ₄₂₃ (2.2.9)	$7.995e-003$	33	5.678	16.277
$1.000e-002$	EARK _{422C2} (3.2.31)	$6.674e-003$	33	5.788	5.129
$2.683e-003$	Matlab ODE 15s	$7.263e-003$	34	109.950	450.953
$2.683e-003$	EGLM ₄₂₃ (2.2.9)	$2.389e-003$	45	6.973	5.586
$2.683e-003$	EARK _{422C2} (3.2.31)	$2.484e-003$	42	6.708	3.633
$7.197e-004$	Matlab ODE 15s	$1.827e-003$	46	113.491	452.012
$7.197e-004$	EGLM ₄₂₃ (2.2.9)	$7.949e-004$	63	8.861	4.754
$7.197e-004$	EARK _{422C2} (3.2.31)	$8.388e-004$	52	7.925	4.457
$1.931e-004$	Matlab ODE 15s	$4.019e-004$	63	127.718	452.625
$1.931e-004$	EGLM ₄₂₃ (2.2.9)	$1.996e-004$	93	11.918	5.934
$1.931e-004$	EARK _{422C2} (3.2.31)	$2.799e-004$	66	9.313	4.664
$5.179e-005$	Matlab ODE 15s	$1.323e-004$	86	145.611	453.332
$5.179e-005$	EGLM ₄₂₃ (2.2.9)	$5.369e-005$	125	15.397	5.070
$5.179e-005$	EARK _{422C2} (3.2.31)	$8.377e-005$	78	10.296	4.055
$1.389e-005$	Matlab ODE 15s	$4.710e-005$	104	177.638	453.332
$1.389e-005$	EGLM ₄₂₃ (2.2.9)	$1.446e-005$	170	20.904	26.273
$1.389e-005$	EARK _{422C2} (3.2.31)	$1.731e-005$	101	12.776	4.340
$3.728e-006$	Matlab ODE 15s	$1.569e-005$	124	204.112	454.652
$3.728e-006$	EGLM ₄₂₃ (2.2.9)	$3.874e-006$	231	26.395	30.500
$3.728e-006$	EARK _{422C2} (3.2.31)	$2.988e-006$	135	16.864	4.984
$1.000e-006$	Matlab ODE 15s	$2.513e-006$	148	231.568	454.914
$1.000e-006$	EGLM ₄₂₃ (2.2.9)	$1.099e-006$	314	36.660	32.641
$1.000e-006$	EARK _{422C2} (3.2.31)	$5.551e-007$	186	22.870	45.180

B.5 Gray-Scott Equation Tables

B.5.1 1-Dimensional Problem

$N = 512$	Method	Global Error	Steps	CPU Time	Memory
$1.000e - 002$	Matlab ODE 15s	$2.320e - 003$	54	17.035	67.543
$1.000e - 002$	EGLM ₄₂₃ (2.2.9)	$1.801e - 004$	60	6.100	18.910
$1.000e - 002$	EARK ₄₂₂ c_2 (3.2.31)	$3.505e - 004$	59	6.115	1.262
$3.511e - 003$	Matlab ODE 15s	$9.686e - 004$	65	18.892	64.031
$3.511e - 003$	EGLM ₄₂₃ (2.2.9)	$6.795e - 005$	78	7.020	0.000
$3.511e - 003$	EARK ₄₂₂ c_2 (3.2.31)	$1.743e - 004$	67	6.505	0.000
$1.233e - 003$	Matlab ODE 15s	$3.505e - 004$	79	21.840	64.031
$1.233e - 003$	EGLM ₄₂₃ (2.2.9)	$2.505e - 005$	103	8.502	17.566
$1.233e - 003$	EARK ₄₂₂ c_2 (3.2.31)	$8.641e - 005$	77	6.942	0.004
$4.329e - 004$	Matlab ODE 15s	$1.217e - 004$	89	20.857	64.039
$4.329e - 004$	EGLM ₄₂₃ (2.2.9)	$9.231e - 006$	124	9.812	34.383
$4.329e - 004$	EARK ₄₂₂ c_2 (3.2.31)	$2.911e - 005$	88	7.582	0.000
$1.520e - 004$	Matlab ODE 15s	$4.156e - 005$	106	24.445	64.105
$1.520e - 004$	EGLM ₄₂₃ (2.2.9)	$3.357e - 006$	154	11.669	22.707
$1.520e - 004$	EARK ₄₂₂ c_2 (3.2.31)	$1.338e - 005$	99	8.315	0.000
$5.337e - 005$	Matlab ODE 15s	$1.672e - 005$	126	28.564	64.035
$5.337e - 005$	EGLM ₄₂₃ (2.2.9)	$1.202e - 006$	195	13.931	2.590
$5.337e - 005$	EARK ₄₂₂ c_2 (3.2.31)	$3.462e - 006$	122	9.547	0.852
$1.874e - 005$	Matlab ODE 15s	$5.452e - 006$	149	30.405	69.590
$1.874e - 005$	EGLM ₄₂₃ (2.2.9)	$4.257e - 007$	245	17.145	9.652
$1.874e - 005$	EARK ₄₂₂ c_2 (3.2.31)	$1.202e - 006$	151	11.060	0.000
$6.579e - 006$	Matlab ODE 15s	$3.368e - 006$	167	34.507	64.102
$6.579e - 006$	EGLM ₄₂₃ (2.2.9)	$1.507e - 007$	305	19.750	11.824
$6.579e - 006$	EARK ₄₂₂ c_2 (3.2.31)	$3.341e - 007$	184	13.478	0.000
$2.310e - 006$	Matlab ODE 15s	$1.100e - 006$	194	37.627	65.934
$2.310e - 006$	EGLM ₄₂₃ (2.2.9)	$5.309e - 008$	388	25.397	16.750
$2.310e - 006$	EARK ₄₂₂ c_2 (3.2.31)	$1.005e - 007$	233	16.240	0.238
$8.111e - 007$	Matlab ODE 15s	$3.536e - 007$	224	43.290	67.055
$8.111e - 007$	EGLM ₄₂₃ (2.2.9)	$1.867e - 008$	498	32.901	19.199
$8.111e - 007$	EARK ₄₂₂ c_2 (3.2.31)	$2.958e - 008$	297	21.013	21.891
$2.848e - 007$	Matlab ODE 15s	$1.170e - 007$	261	48.126	67.621
$2.848e - 007$	EGLM ₄₂₃ (2.2.9)	$6.565e - 009$	639	42.963	24.754
$2.848e - 007$	EARK ₄₂₂ c_2 (3.2.31)	$8.745e - 009$	379	26.411	39.254
$1.000e - 007$	Matlab ODE 15s	$4.407e - 008$	305	57.986	73.109
$1.000e - 007$	EGLM ₄₂₃ (2.2.9)	$2.307e - 009$	819	58.828	27.617
$1.000e - 007$	EARK ₄₂₂ c_2 (3.2.31)	$2.629e - 009$	482	34.507	36.906

$N = 1024$	Method	Global Error	Steps	CPU Time	Memory
$1.000e - 002$	Matlab ODE 15s	$1.582e - 003$	60	88.297	259.625
$1.000e - 002$	EGLM ₄₂₃ (2.2.9)	$9.644e - 005$	71	11.762	57.789
$1.000e - 002$	EARK _{422C2} (3.2.31)	$2.538e - 004$	63	10.842	4.109
$3.511e - 003$	Matlab ODE 15s	$6.037e - 004$	73	96.877	256.039
$3.511e - 003$	EGLM ₄₂₃ (2.2.9)	$3.549e - 005$	90	13.260	1.367
$3.511e - 003$	EARK _{422C2} (3.2.31)	$8.653e - 005$	73	11.435	-0.426
$1.233e - 003$	Matlab ODE 15s	$1.905e - 004$	85	96.658	256.691
$1.233e - 003$	EGLM ₄₂₃ (2.2.9)	$1.302e - 005$	117	15.241	0.539
$1.233e - 003$	EARK _{422C2} (3.2.31)	$3.734e - 005$	84	12.605	0.574
$4.329e - 004$	Matlab ODE 15s	$6.433e - 005$	100	112.789	258.871
$4.329e - 004$	EGLM ₄₂₃ (2.2.9)	$4.745e - 006$	144	17.925	1.836
$4.329e - 004$	EARK _{422C2} (3.2.31)	$2.088e - 005$	93	13.307	0.438
$1.520e - 004$	Matlab ODE 15s	$2.973e - 005$	118	127.874	258.820
$1.520e - 004$	EGLM ₄₂₃ (2.2.9)	$1.692e - 006$	179	21.372	27.086
$1.520e - 004$	EARK _{422C2} (3.2.31)	$5.739e - 006$	114	15.210	1.570
$5.337e - 005$	Matlab ODE 15s	$1.321e - 005$	140	143.022	260.434
$5.337e - 005$	EGLM ₄₂₃ (2.2.9)	$6.025e - 007$	226	26.271	26.203
$5.337e - 005$	EARK _{422C2} (3.2.31)	$1.779e - 006$	140	17.737	1.949
$1.874e - 005$	Matlab ODE 15s	$3.668e - 006$	156	146.859	260.902
$1.874e - 005$	EGLM ₄₂₃ (2.2.9)	$2.146e - 007$	282	32.479	40.770
$1.874e - 005$	EARK _{422C2} (3.2.31)	$5.015e - 007$	171	21.419	37.477
$6.579e - 006$	Matlab ODE 15s	$1.928e - 006$	183	173.208	261.848
$6.579e - 006$	EGLM ₄₂₃ (2.2.9)	$7.560e - 008$	357	40.389	41.504
$6.579e - 006$	EARK _{422C2} (3.2.31)	$1.519e - 007$	215	25.023	34.336
$2.310e - 006$	Matlab ODE 15s	$5.518e - 007$	220	194.876	262.629
$2.310e - 006$	EGLM ₄₂₃ (2.2.9)	$2.658e - 008$	458	55.334	38.582
$2.310e - 006$	EARK _{422C2} (3.2.31)	$4.473e - 008$	273	31.949	55.301
$8.111e - 007$	Matlab ODE 15s	$3.402e - 007$	262	244.110	263.664
$8.111e - 007$	EGLM ₄₂₃ (2.2.9)	$9.348e - 009$	587	72.603	72.035
$8.111e - 007$	EARK _{422C2} (3.2.31)	$1.326e - 008$	348	47.128	76.070
$2.848e - 007$	Matlab ODE 15s	$9.387e - 008$	312	259.320	265.883
$2.848e - 007$	EGLM ₄₂₃ (2.2.9)	$3.286e - 009$	751	102.555	109.785
$2.848e - 007$	EARK _{422C2} (3.2.31)	$3.951e - 009$	446	57.018	81.410
$1.000e - 007$	Matlab ODE 15s	$1.964e - 008$	3769	626.048	373.090
$1.000e - 007$	EGLM ₄₂₃ (2.2.9)	$1.155e - 009$	969	151.181	164.125
$1.000e - 007$	EARK _{422C2} (3.2.31)	$1.197e - 009$	569	75.224	135.133

B.5.2 2-Dimensional Problem

$N = 32 \times 32$	Method	Global Error	Steps	CPU Time	Memory
$1.000e - 002$	Matlab ODE 15s	$8.642e - 003$	23	59.873	256.418
$1.000e - 002$	EGLM ₄₂₃ (2.2.9)	$1.738e - 003$	32	3.354	14.105
$1.000e - 002$	EARK ₄₂₂ c_2 (3.2.31)	$2.712e - 003$	29	3.167	5.633
$2.683e - 003$	Matlab ODE 15s	$3.364e - 003$	31	62.135	256.031
$2.683e - 003$	EGLM ₄₂₃ (2.2.9)	$5.991e - 004$	42	4.025	9.113
$2.683e - 003$	EARK ₄₂₂ c_2 (3.2.31)	$3.179e - 004$	44	4.446	13.641
$7.197e - 004$	Matlab ODE 15s	$1.915e - 003$	39	67.954	256.031
$7.197e - 004$	EGLM ₄₂₃ (2.2.9)	$1.645e - 004$	60	5.476	22.816
$7.197e - 004$	EARK ₄₂₂ c_2 (3.2.31)	$2.079e - 004$	50	4.571	3.199
$1.931e - 004$	Matlab ODE 15s	$3.527e - 004$	49	78.001	256.031
$1.931e - 004$	EGLM ₄₂₃ (2.2.9)	$4.819e - 005$	75	6.115	3.965
$1.931e - 004$	EARK ₄₂₂ c_2 (3.2.31)	$5.353e - 005$	62	5.460	1.969
$5.179e - 005$	Matlab ODE 15s	$6.941e - 005$	61	85.816	258.070
$5.179e - 005$	EGLM ₄₂₃ (2.2.9)	$1.536e - 005$	96	7.535	19.266
$5.179e - 005$	EARK ₄₂₂ c_2 (3.2.31)	$1.275e - 005$	69	5.554	5.238
$1.389e - 005$	Matlab ODE 15s	$2.521e - 005$	75	87.907	258.008
$1.389e - 005$	EGLM ₄₂₃ (2.2.9)	$4.173e - 006$	126	9.688	1.121
$1.389e - 005$	EARK ₄₂₂ c_2 (3.2.31)	$1.106e - 005$	84	6.646	27.629
$3.728e - 006$	Matlab ODE 15s	$8.844e - 006$	91	97.345	258.945
$3.728e - 006$	EGLM ₄₂₃ (2.2.9)	$1.146e - 006$	164	12.698	0.926
$3.728e - 006$	EARK ₄₂₂ c_2 (3.2.31)	$9.356e - 007$	105	8.143	0.961
$1.000e - 006$	Matlab ODE 15s	$1.788e - 006$	108	113.803	259.539
$1.000e - 006$	EGLM ₄₂₃ (2.2.9)	$3.093e - 007$	217	16.864	1.012
$1.000e - 006$	EARK ₄₂₂ c_2 (3.2.31)	$2.114e - 007$	139	10.702	0.770

$N = 64 \times 64$	Method	Global Error	Steps	CPU Time	Memory
$1.000e - 002$	Matlab ODE 15s	N/A	N/A	N/A	4158.961
$1.000e - 002$	EGLM ₄₂₃ (2.2.9)	N/A	41	17.269	77.797
$1.000e - 002$	EARK ₄₂₂ c_2 (3.2.31)	N/A	43	11.123	36.516
$2.683e - 003$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$2.683e - 003$	EGLM ₄₂₃ (2.2.9)	N/A	66	14.165	24.898
$2.683e - 003$	EARK ₄₂₂ c_2 (3.2.31)	N/A	58	13.744	36.301
$7.197e - 004$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$7.197e - 004$	EGLM ₄₂₃ (2.2.9)	N/A	82	17.394	14.504
$7.197e - 004$	EARK ₄₂₂ c_2 (3.2.31)	N/A	70	15.475	75.980
$1.931e - 004$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$1.931e - 004$	EGLM ₄₂₃ (2.2.9)	N/A	102	20.545	108.430
$1.931e - 004$	EARK ₄₂₂ c_2 (3.2.31)	N/A	73	15.444	47.000
$5.179e - 005$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$5.179e - 005$	EGLM ₄₂₃ (2.2.9)	N/A	140	30.358	117.430
$5.179e - 005$	EARK ₄₂₂ c_2 (3.2.31)	N/A	95	21.513	160.012
$1.389e - 005$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$1.389e - 005$	EGLM ₄₂₃ (2.2.9)	N/A	174	39.359	114.762
$1.389e - 005$	EARK ₄₂₂ c_2 (3.2.31)	N/A	103	22.901	137.160
$3.728e - 006$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$3.728e - 006$	EGLM ₄₂₃ (2.2.9)	N/A	224	56.160	190.895
$3.728e - 006$	EARK ₄₂₂ c_2 (3.2.31)	N/A	136	30.966	151.008
$1.000e - 006$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$1.000e - 006$	EGLM ₄₂₃ (2.2.9)	N/A	305	77.314	243.012
$1.000e - 006$	EARK ₄₂₂ c_2 (3.2.31)	N/A	176	41.886	301.766

B.5.3 3-Dimensional Problem

$N = 8 \times 8 \times 8$	Method	Global Error	Steps	CPU Time	Memory
$1.000e-002$	Matlab ODE 15s	$5.347e-004$	16	6.053	70.660
$1.000e-002$	EGLM ₄₂₃ (2.2.9)	$3.036e-005$	21	2.075	6.977
$1.000e-002$	EARK _{422C2} (3.2.31)	$1.496e-005$	21	2.293	1.723
$2.683e-003$	Matlab ODE 15s	$2.059e-004$	20	7.114	64.031
$2.683e-003$	EGLM ₄₂₃ (2.2.9)	$1.346e-005$	26	2.356	1.379
$2.683e-003$	EARK _{422C2} (3.2.31)	$1.058e-004$	25	2.371	0.773
$7.197e-004$	Matlab ODE 15s	$7.707e-005$	24	7.519	64.031
$7.197e-004$	EGLM ₄₂₃ (2.2.9)	$4.610e-006$	31	2.402	0.484
$7.197e-004$	EARK _{422C2} (3.2.31)	$5.684e-005$	29	2.324	0.871
$1.931e-004$	Matlab ODE 15s	$2.399e-005$	30	9.282	64.035
$1.931e-004$	EGLM ₄₂₃ (2.2.9)	$1.567e-006$	40	2.761	2.758
$1.931e-004$	EARK _{422C2} (3.2.31)	$1.682e-005$	34	2.449	0.000
$5.179e-005$	Matlab ODE 15s	$7.836e-006$	37	9.875	64.031
$5.179e-005$	EGLM ₄₂₃ (2.2.9)	$4.620e-007$	54	3.526	9.449
$5.179e-005$	EARK _{422C2} (3.2.31)	$3.944e-006$	39	2.730	0.000
$1.389e-005$	Matlab ODE 15s	$2.586e-006$	44	10.889	64.031
$1.389e-005$	EGLM ₄₂₃ (2.2.9)	$1.322e-007$	67	3.947	8.953
$1.389e-005$	EARK _{422C2} (3.2.31)	$1.116e-006$	47	3.198	1.715
$3.728e-006$	Matlab ODE 15s	$1.004e-006$	54	12.230	64.031
$3.728e-006$	EGLM ₄₂₃ (2.2.9)	$3.721e-008$	85	4.836	18.844
$3.728e-006$	EARK _{422C2} (3.2.31)	$2.657e-007$	58	3.354	0.977
$1.000e-006$	Matlab ODE 15s	$3.725e-007$	62	14.726	64.113
$1.000e-006$	EGLM ₄₂₃ (2.2.9)	$1.041e-008$	109	5.569	34.168
$1.000e-006$	EARK _{422C2} (3.2.31)	$5.408e-008$	71	4.228	3.906

$N = 16 \times 16 \times 16$	Method	Global Error	Steps	CPU Time	Memory
$1.000e-002$	Matlab ODE 15s	N/A	N/A	N/A	4101.867
$1.000e-002$	EGLM ₄₂₃ (2.2.9)	N/A	56	52.026	60.184
$1.000e-002$	EARK _{422C2} (3.2.31)	N/A	47	17.862	47.305
$2.683e-003$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$2.683e-003$	EGLM ₄₂₃ (2.2.9)	N/A	70	16.692	27.230
$2.683e-003$	EARK _{422C2} (3.2.31)	N/A	65	20.483	143.340
$7.197e-004$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$7.197e-004$	EGLM ₄₂₃ (2.2.9)	N/A	99	24.664	106.770
$7.197e-004$	EARK _{422C2} (3.2.31)	N/A	74	18.798	46.227
$1.931e-004$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$1.931e-004$	EGLM ₄₂₃ (2.2.9)	N/A	128	31.465	109.066
$1.931e-004$	EARK _{422C2} (3.2.31)	N/A	88	24.071	147.148
$5.179e-005$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$5.179e-005$	EGLM ₄₂₃ (2.2.9)	N/A	156	36.660	114.695
$5.179e-005$	EARK _{422C2} (3.2.31)	N/A	110	27.331	161.758
$1.389e-005$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$1.389e-005$	EGLM ₄₂₃ (2.2.9)	N/A	205	50.903	192.574
$1.389e-005$	EARK _{422C2} (3.2.31)	N/A	131	32.557	218.172
$3.728e-006$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$3.728e-006$	EGLM ₄₂₃ (2.2.9)	N/A	273	73.898	232.445
$3.728e-006$	EARK _{422C2} (3.2.31)	N/A	170	44.616	298.668
$1.000e-006$	Matlab ODE 15s	N/A	N/A	N/A	N/A
$1.000e-006$	EGLM ₄₂₃ (2.2.9)	N/A	365	107.828	317.359
$1.000e-006$	EARK _{422C2} (3.2.31)	N/A	224	64.272	345.898

Appendix C

Alternative Order Condition Proofs

C.1 3-Stage, 6th Order EAGLMs

What follows is an alternative proof of EAGLM order conditions in a grid format similar to the ERK proof of Theorem 2.

Theorem 19. *3-stage EAGLMs of the form*

$$\begin{array}{c|ccc|ccc|cc}
 0 & & & & & & & & & \\
 c_2 & a_{21} & & & e^{c_2 h L} & u_{21} & u_{22} & p_{21} & p_{22} & \\
 c_3 & a_{31} & a_{32} & & e^{c_3 h L} & u_{31} & u_{32} & p_{31} & p_{32} & \\
 \hline
 & b_1 & b_2 & b_3 & e^{h L} & v_1 & v_2 & q_1 & q_2 &
 \end{array} \tag{C.1.1}$$

$$K_1 = N$$

$$K_2 = N \left(t_n + c_2 h, e^{c_2 h L} y_n + h \left[a_{21} K_1 + u_{21} N_{t_{n-1}} + u_{22} N_{t_{n-2}} + p_{21} N' + p_{22} N'' \right] \right)$$

$$K_3 = N \left(t_n + c_3 h, e^{c_3 h L} y_n + h \left[a_{31} K_1 + a_{32} K_2 + u_{31} N_{t_{n-1}} + u_{32} N_{t_{n-2}} + p_{31} N' + p_{32} N'' \right] \right)$$

$$y_{n+1} = e^{h L} y_n + h \left(b_1 K_1 + b_2 K_2 + b_3 K_3 + v_1 N_{t_{n-1}} + v_2 N_{t_{n-2}} + q_1 N' + q_2 N'' \right)$$

that meet the consistency conditions (3.2.4), can achieve the following listed orders if they also satisfy the accompanying conditions.

2nd Order

$$b_3 c_3 - 2 v_2 + b_2 c_2 - v_1 + q_1 = \varphi_2 \tag{C.1.2}$$

3rd Order

$$b_3 c_3^2 + 4 v_2 + 2 q_2 + b_2 c_2^2 + v_1 = 2 \varphi_3 \quad (\text{C.1.3a})$$

$$b_2 (p_{21} - u_{21} - 2u_{22} - c_2^2 \varphi_{22}) + b_3 (c_2 a_{32} + p_{31} - u_{31} - 2u_{32} - c_3^2 \varphi_{23}) = 0 \quad (\text{C.1.3b})$$

4th Order

$$b_3 c_3^3 - 8 v_2 + b_2 c_2^3 - v_1 = 3! \varphi_4 \quad (\text{C.1.4a})$$

$$p_{21} - u_{21} - 2u_{22} - c_2^2 \varphi_{22} = 0 \quad (\text{C.1.4b})$$

$$c_2 a_{32} + p_{31} - u_{31} - 2u_{32} - c_3^2 \varphi_{23} = 0 \quad (\text{C.1.4c})$$

$$b_2 (p_{22} + \frac{1}{2} u_{21} + \frac{4}{2!} u_{22} - c_2^3 \varphi_{32}) + b_3 (\frac{1}{2} c_2^2 a_{32} + p_{32} + \frac{1}{2} u_{31} + \frac{4}{2!} u_{32} - c_3^3 \varphi_{33}) = 0 \quad (\text{C.1.4d})$$

5th Order

$$b_3 c_3^4 + 16 v_2 + b_2 c_2^4 + v_1 = 4! \varphi_5 \quad (\text{C.1.5a})$$

$$p_{22} + \frac{1}{2} u_{21} + \frac{4}{2!} u_{22} - c_2^3 \varphi_{32} = 0 \quad (\text{C.1.5b})$$

$$\frac{1}{2} c_2^2 a_{32} + p_{32} + \frac{1}{2} u_{31} + \frac{4}{2!} u_{32} - c_3^3 \varphi_{33} = 0 \quad (\text{C.1.5c})$$

$$b_2 (-\frac{1}{3!} u_{21} - \frac{8}{3!} u_{22} - c_2^4 \varphi_{42}) + b_3 (\frac{1}{3!} c_2^3 a_{32} - \frac{1}{3!} u_{31} - \frac{8}{3!} u_{32} - c_3^4 \varphi_{43}) = 0 \quad (\text{C.1.5d})$$

6th Order

$$b_3 c_3^5 - 32 v_2 + b_2 c_2^5 - v_1 = 5! \varphi_6 \quad (\text{C.1.6a})$$

$$-\frac{1}{3!} u_{21} - \frac{8}{3!} u_{22} - c_2^4 \varphi_{42} = 0 \quad (\text{C.1.6b})$$

$$\frac{1}{3!} c_2^3 a_{32} - \frac{1}{3!} u_{31} - \frac{8}{3!} u_{32} - c_3^4 \varphi_{43} = 0 \quad (\text{C.1.6c})$$

$$b_2 (\frac{1}{4!} u_{21} + \frac{16}{4!} u_{22} - c_2^5 \varphi_{52}) + b_3 (\frac{1}{4!} c_2^4 a_{32} + \frac{1}{4!} u_{31} + \frac{16}{4!} u_{32} - c_3^5 \varphi_{53}) = 0 \quad (\text{C.1.6d})$$

Proof. The R_2 grid for 6th order EAGLMs is much the same as that of EGLMs though in this case extended to h^5 . We introduce a new notation here of labelling the occupied grid locations so that referencing them later is much clearer.

R_2	1	h	h^2	h^3	h^4	h^5
N						
N'			$(p_{21} - u_{21} \quad [\mathbf{A}]$ $-2u_{22} - c_2^2 \varphi_{22})$			
N''				$(p_{22} + \frac{1}{2}u_{21} \quad [\mathbf{B}]$ $+ \frac{4}{2!}u_{22} - c_2^3 \varphi_{32})$		
N'''					$(-\frac{1}{3!}u_{21} \quad [\mathbf{C}]$ $-\frac{8}{3!}u_{22} - c_2^4 \varphi_{42})$	
N''''						$(\frac{1}{4!}u_{21} \quad [\mathbf{D}]$ $+ \frac{16}{4!}u_{22} - c_2^5 \varphi_{52})$
N'''''						

(C.1.7)

In addition, to keep the terms manageable we will simply refer to entries in the R_2 grid rather than write out the expressions explicitly.

K_2	1	h	h^2	h^3	h^4	h^5
N	1					
N'		c_2	$[R_2, \mathbf{A}] N_u \quad [\mathbf{A}]$	$c_2 [R_2, \mathbf{A}] N'_u \quad [\mathbf{E}]$	$\frac{c_2^2}{2} [R_2, \mathbf{A}] N''_u \quad [\mathbf{H}]$	$\frac{c_2^3}{6} [R_2, \mathbf{A}] N'''_u \quad [\mathbf{J}]$
N''			$\frac{1}{2} c_2^2$	$[R_2, \mathbf{B}] N_u \quad [\mathbf{B}]$	$c_2 [R_2, \mathbf{B}] N'_u \quad [\mathbf{F}]$	$\frac{c_2^2}{2} [R_2, \mathbf{B}] N''_u \quad [\mathbf{I}]$
N'''				$\frac{1}{6} c_2^2$	$[R_2, \mathbf{C}] N_u \quad [\mathbf{C}]$	$c_2 [R_2, \mathbf{C}] N'_u \quad [\mathbf{G}]$
N''''					$\frac{1}{4!} c_2^4$	$[R_2, \mathbf{D}] N_u \quad [\mathbf{D}]$
N'''''						$\frac{1}{5!} c_2^5$

K_2	h^4	h^5
$(N')^2$	$([R_2, \mathbf{A}])^2 N_{uu} \quad [\mathbf{K}]$	$c_2 ([R_2, \mathbf{A}])^2 N'_{uu} \quad [\mathbf{M}]$
$N' N''$		$[R_2, \mathbf{A}] \times [R_2, \mathbf{B}] N_{uu} \quad [\mathbf{L}]$

(C.1.8)

The K_2 grid now contains two new rows, this is because to achieve 6th order we cannot ignore the R_i^2 terms from the K_i expansion (2.1.6). Specifically for K_2 we have

$$R_2^2 = [R_2, \mathbf{A}]^2 (N')^2 h^4 + ([R_2, \mathbf{A}] + [R_2, \mathbf{B}]) N' N'' h^5 + O(h^6) \quad (\text{C.1.9})$$

The R_3 and K_3 grids also simply reference entries in the previous K_2 and R_3 grids respectively.

R_3	1	h	h^2	h^3	h^4	h^5
N						
N'			$(c_2 a_{32} - c_3^2 \varphi_{23} \quad [\text{A}]$ $+ p_{31} - u_{31} - 2u_{32})$	$a_{32} [K_2, \text{A}] \quad [\text{E}]$	$a_{32} [K_2, \text{E}] \quad [\text{H}]$	$a_{32} [K_2, \text{H}] \quad [\text{J}]$
N''				$(\frac{1}{2} c_2^2 a_{32} - c_3^3 \varphi_{33} \quad [\text{B}]$ $+ p_{32} + \frac{1}{2} u_{31} + 2u_{32})$	$a_{32} [K_2, \text{B}] \quad [\text{F}]$	$a_{32} [K_2, \text{F}] \quad [\text{I}]$
N'''					$(\frac{1}{6} c_2^3 a_{32} - c_3^4 \varphi_{43} \quad [\text{C}]$ $-\frac{1}{6} u_{31} - \frac{4}{3} u_{32})$	$a_{32} [K_2, \text{C}] \quad [\text{G}]$
N''''						$(\frac{1}{4!} c_2^4 a_{32} - c_3^5 \varphi_{53} \quad [\text{D}]$ $+ \frac{1}{4!} u_{31} + \frac{2}{3} u_{32})$
N'''''						

R_3	h^4	h^5
$(N')^2$		$a_{32} [K_2, \text{K}] \quad [\text{K}]$
$N' N''$		

(C.1.10)

As was required for K_2 we must preserve the R_3^2 terms in expansion of K_3 .

$$R_3^2 = [R_3, \text{A}]^2 (N')^2 h^4 + ([R_3, \text{A}] + [R_3, \text{E}]) (N')^2 h^5 ([R_3, \text{A}] + [R_3, \text{B}]) N' N'' h^5 + O(h^6) \quad (\text{C.1.11})$$

K_3	1	h	h^2	h^3	h^4	h^5
N	1					
N'		c_3	$[\text{A}]$ $[R_3, \text{A}] N_u$	$[\text{E}]$ $c_3 [R_3, \text{A}] N'_u + [R_3, \text{E}] N_u$	$[\text{H}]$ $\frac{1}{2} c_3^2 [R_3, \text{A}] N''_u + c_3 [R_3, \text{E}] N'_u + [R_3, \text{H}] N_u$	$[\text{J}]$ $[R_3, \text{J}] N_u + c_3 [R_3, \text{H}] N'_u + \frac{1}{2} c_3^2 [R_3, \text{E}] N''_u + \frac{1}{6} c_3^3 [R_3, \text{A}] N'''_u$
N''			$\frac{1}{2} c_3^2$	$[\text{B}]$ $[R_3, \text{B}] N_u$	$[\text{F}]$ $c_3 [R_3, \text{B}] N'_u + [R_3, \text{F}] N_u$	$[\text{I}]$ $\frac{1}{2} c_3^2 [R_3, \text{B}] N''_u + c_3 [R_3, \text{F}] N'_u + [R_3, \text{I}] N_u$
N'''				$\frac{1}{6} c_3^3$	$[\text{C}]$ $[R_3, \text{C}] N_u$	$[\text{G}]$ $c_3 [R_3, \text{C}] N'_u + [R_3, \text{G}] N_u$
N''''					$\frac{1}{4!} c_3^4$	$[\text{D}]$ $[R_3, \text{D}] N_u$
N'''''						$\frac{1}{5!} c_3^5$

K_3	h^4	h^5
$(N')^2$	$([R_3, \text{A}])^2 N_{uu} \quad [\text{K}]$	$[R_3, \text{K}] N_u + c_3 ([R_3, \text{A}])^2 N'_{uu} + [R_3, \text{A}] \times [R_3, \text{E}] N_{uu} \quad [\text{M}]$
$N' N''$		$[R_3, \text{A}] \times [R_3, \text{B}] N_{uu} \quad [\text{L}]$

(C.1.12)

Summarizing Conditions

The grids cells along the diagonal, $(h^i, N^{(i)})$ for $i \geq 1$, recover respective the first condition of each i^{th} order set of conditions. That is, the necessary 2nd order condition (C.1.2) as well as the additional conditions (C.1.3a), (C.1.4a), (C.1.5a) and (C.1.6a).

3rd Order

The $h^2 N' N_u$ cells imply the condition $b_3 [R_3, A] + b_2 [R_2, A] = 0$ which, with condition In addition to (C.1.3a), recovers the remaining 3rd order condition (C.1.3b).

4th Order

We get the following conditions from the cells;

- $h^3 N' N'_u \Rightarrow b_3 c_3 [R_3, A] + b_2 c_2 [R_2, A] = 0$
- $h^3 N' (N_u)^2 \Rightarrow b_3 a_{32} [R_2, A] = 0$
- $h^3 N'' N_u \Rightarrow b_3 [R_3, B] + b_2 [R_2, B] = 0$

The requirement $b_3 a_{32} [R_2, A] = 0$ together with the earlier $b_3 [R_3, A] + b_2 [R_2, A] = 0$ implies both $[R_2, A]$ and $[R_3, A] = 0$ recovering conditions (C.1.4b) and (C.1.4c). From $b_3 [R_3, B] + b_2 [R_2, B] = 0$ we get (C.1.4d).

5th Order

Here the h^4 cells imply

- $h^4 N' N''_u \Rightarrow \frac{b_3 c_3^2 [R_3, A]}{2} + \frac{b_2 c_2^2 [R_2, A]}{2} = 0$
- $h^4 N' N'_u N_u \Rightarrow b_3 a_{32} (c_3 + c_2) [R_2, A] = 0$
- $h^4 N'' N'_u \Rightarrow b_3 c_3 [R_3, B] + b_2 c_2 [R_2, B] = 0$
- $h^4 N'' (N_u)^2 \Rightarrow b_3 a_{32} [R_2, B] = 0$
- $h^4 N''' N_u \Rightarrow b_3 [R_3, C] + b_2 [R_2, C] = 0$
- $h^4 (N')^2 N_{uu} \Rightarrow b_3 ([R_3, A])^2 + b_2 ([R_2, A])^2 = 0$

Like before, the requirement $b_3 a_{32} [R_2, \mathbb{B}] = 0$ together with the earlier $b_3 [R_3, \mathbb{B}] + b_2 [R_2, \mathbb{B}] = 0$ implies both $[R_2, \mathbb{B}]$ and $[R_3, \mathbb{B}] = 0$ recovering conditions (C.1.5b) and (C.1.5c). From $b_3 [R_3, \mathbb{C}] + b_2 [R_2, \mathbb{C}] = 0$ we get (C.1.5d).

6th Order

The cells

- $h^5 N' N_u''' \Rightarrow \frac{1}{6} b_3 c_3^3 [R_3, \mathbb{A}] + \frac{1}{6} b_2 c_2^3 [R_2, \mathbb{A}] = 0$
- $h^5 N' N_u'' N_u \Rightarrow b_3 a_{32} \left(\frac{c_3^2 + c_2^2}{2} \right) [R_2, \mathbb{A}] = 0$
- $h^5 N' (N_u')^2 \Rightarrow c_2 c_3 b_3 a_{32} [R_2, \mathbb{A}] = 0$
- $h^5 N'' N_u'' \Rightarrow \frac{1}{2} b_3 c_3^2 [R_3, \mathbb{B}] + \frac{1}{2} b_2 c_2^2 [R_2, \mathbb{B}] = 0$
- $h^5 N'' N_u N_u' \Rightarrow b_3 (c_3 + c_2) a_{32} [R_2, \mathbb{B}] = 0$
- $h^5 N''' N_u' \Rightarrow b_3 c_3 [R_3, \mathbb{C}] + b_2 c_2 [R_2, \mathbb{C}] = 0$
- $h^5 N''' (N_u')^2 \Rightarrow b_3 a_{32} [R_2, \mathbb{C}] = 0$
- $h^5 N'''' N_u \Rightarrow b_3 [R_3, \mathbb{D}] + b_2 [R_2, \mathbb{D}] = 0$
- $h^5 (N')^2 N_u N_{uu} \Rightarrow b_3 a_{32} [R_2, \mathbb{A}] ([R_3, \mathbb{A}] + [R_2, \mathbb{A}]) = 0$
- $h^5 (N')^2 N_{uu}' \Rightarrow b_3 c_3 ([R_3, \mathbb{A}])^2 + b_2 c_2 ([R_2, \mathbb{A}])^2 = 0$
- $h^5 N' N'' N_{uu} \Rightarrow b_3 ([R_3, \mathbb{A}] \times [R_3, \mathbb{B}]) + b_2 ([R_2, \mathbb{A}] \times [R_2, \mathbb{B}]) = 0$

Though there are a lot more 6th orders conditions from the h^5 cells there is a lot of repetition. The requirement $b_3 a_{32} [R_2, \mathbb{C}] = 0$ together with the earlier $b_3 [R_3, \mathbb{C}] + b_2 [R_2, \mathbb{C}] = 0$ implies both $[R_2, \mathbb{B}]$ and $[R_3, \mathbb{B}] = 0$. This satisfies a number of the cells conditions and recovers the required order conditions (C.1.6b) and (C.1.6c). Finally, the requirement $b_3 [R_3, \mathbb{D}] + b_2 [R_2, \mathbb{D}] = 0$ gives condition (C.1.6d). \square

Appendix D

Acronyms

AMAB	Adams-Moulton / Adams-Bashforth Method	5
ARK	Almost Runge-Kutta Method	29
EAGLM	Exponential Almost General Linear Method	29
EARK	Exponential Almost Runge-Kutta Method	29
EI	Exponential Time-Integrator	1
EGLM	Exponential General Linear Method	6
ELP	<i>Exact</i> treatment of the <i>Linear Part, L</i> , Method	5
ETD	Exponential Time Differencing Method	5
ETDRK	ETD Runge-Kutta Method	5
ERK	Exponential Runge-Kutta Method	5
GLM	General Linear Method	6
I-EM	mixed Implicit-Explicit Method	5
IF	Integrating Factor Method	4
IFAB	Integrating Factor/Adams-Bashford Method	60
IFRK	Integrating Factor/Classical Runge-Kutta Method	60
RDA	Reaction Diffusion Advection	73
ReLPM	Real Leja Points Method	80
RK	Runge-Kutta Method	4

Bibliography

- [1] G. Akrivis and Y.-S. Smyrlis, *Implicit-explicit BDF methods for the Kuramoto-Sivashinsky equation*, Applied Numerical Mathematics **51** (2004), no. 2-3, 151 – 169.
- [2] L. Bergamaschi, M. Caliori, and A. Martínez and M. Vianello, *Comparing Leja and Krylov approximations of large scale matrix exponentials*, International Conference on Computational Science (4), 2006, pp. 685–692.
- [3] L. Bergamaschi and M. Vianello, *Efficient computation of the exponential operator for large, sparse, symmetric matrices*, Numer. Linear Algebra Appl **7**, 27–45.
- [4] H. Berland, B. Skaflestad, and W. M. Wright, *Expint — a Matlab package for exponential integrators*, ACM Trans. Math. Softw. **33** (2007), no. 1, 4.
- [5] G. Beylkin, J.M. Keiser, and L. Vozovoi, *A new class of time discretization schemes for the solution of nonlinear PDEs*, J. Comput. Phys. **147** (1998), no. 2, 362–387.
- [6] P. Bogacki and L.F. Shampine, *A 3(2) pair of Runge-Kutta formulas*, Applied Mathematics Letters **2** (1989), no. 4, 321 – 325.
- [7] J.C. Butcher, *On the convergence of numerical solutions to ordinary differential equations*, Math. Comp. **Volume 20** (1966), 1–10.
- [8] ———, *General linear methods*, Computers & Mathematics with Applications **Volume 31** (1996), no. 4-5, 105–112.
- [9] ———, *An introduction to “almost Runge-Kutta” methods*, Appl. Numer. Math. **24** (1997), no. 2-3, 331–342.
- [10] M. Caliori and A. Ostermann, *Implementation of exponential Rosenbrock-type integrators*, Appl. Numer. Math. **59** (2009), no. 3-4, 568–581.
- [11] M. Caliori, M. Vianello, and L. Bergamaschi, *Interpolating discrete advection-diffusion propagators at Leja sequences*, J. Comput. Appl. Math. **172** (2004), no. 1, 79–99.
- [12] M.P. Calvo and A.M. Portillo, *Variable step implementation of ETD methods for semilinear problems*, Applied Mathematics and Computation **196** (2008), no. 2, 627 – 637.
- [13] S.M. Cox and P.C. Matthews, *Exponential time differencing for stiff systems*, J. Comput. Phys. **176** (2002), no. 2, 430–455.

- [14] C.F. Curtiss and J.O. Hirschfelder, *Integration of stiff equations*, Proceedings of the National Academy of Sciences **38** (1952), 235–243.
- [15] Qiang D. and Wenxiang Z., *Analysis and applications of the exponential time differencing schemes and their contour integration modifications*, BIT Numerical Mathematics **45** (2005), 307–328, 10.1007/s10543-005-7141-8.
- [16] R. Frank, J. Schneid, and C.W. Ueberhuber, *The concept of b-convergence*, SIAM Journal on Numerical Analysis **18** (1981), no. 5, pp. 753–780 (English).
- [17] R.A. Friesner, L.S. Tuckerman, B.C. Dornblaser, and T.V. Russo, *A method for exponential propagation of large systems of stiff nonlinear differential equations*, J. Sci. Comput. **4** (1989), no. 4, 327–354.
- [18] E. Gallopoulos and Y. Saad, *Efficient solution of parabolic equations by Krylov approximation methods*, SIAM J. Sci. Stat. Comput. **13** (1992), no. 5, 1236–1264.
- [19] C.W. Gear, *Numerical initial value problems in ordinary differential equations*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1971.
- [20] E. Hairer, S.P. Norsett, and G. Wanner, *Solving Ordinary Differential Equations I, Nonstiff problems, Second Revised Edition with 135 Figures, Vol.: 1*, 2Ed. Springer-Verlag, 2000, 2000.
- [21] ———, *Solving Ordinary Differential Equations II. stiff and Differential-Algebraic Problems, Second Revised Edition with 137 Figures, Vol.: 2*, 2Ed. Springer-Verlag, 2002, 2002.
- [22] J.K. Hale, L.A. Peletier, and W.C. Troy, *Stability and instability in the Gray-Scott model: The case of equal diffusivities*, Applied Mathematics Letters **12** (1999), no. 4, 59 – 65.
- [23] D. Henry, *Geometric theory of semilinear parabolic equations*, Lecture Notes in Mathematics, vol. 840, Springer Berlin / Heidelberg, 1981, 10.1007/BFb0089648.
- [24] M. Hochbruck, C. Lubich, and H. Selhofer, *Exponential integrators for large systems of differential equations*, SIAM J. Sci. Comput. **19** (1998), no. 5, 1552–1574.
- [25] M. Hochbruck and A. Ostermann, *Explicit exponential Runge-Kutta methods for semilinear parabolic problems*, SIAM J. Numer. Anal. **43** (2005), no. 3, 1069–1090.
- [26] A.-K. Kassam and L.N. Trefethen, *Fourth-order time-stepping for stiff PDEs*, SIAM J. Sci. Comput. **26** (2005), no. 4, 1214–1233.
- [27] S. Krogstad, *Generalized integrating factor methods for stiff PDEs*, J. Comput. Phys. **203** (2005), no. 1, 72–88.
- [28] J.D. Lawson, *Generalized Runge-Kutta processes for stable systems with large Lipschitz constants*, SIAM Journal on Numerical Analysis **4** (1967), no. 3, 372–380.
- [29] W. Liniger and R.A. Willoughby, *Efficient integration methods for stiff systems of ordinary differential equations*, SIAM Journal on Numerical Analysis **7** (1970), no. 1, pp. 47–66 (English).
- [30] Y.Y. Lu, *Exponentials of symmetric matrices through tridiagonal reductions*, Linear Algebra and its Applications **279** (1998), 317–324.

- [31] Y.Y. Lu, *Computing a matrix function for exponential integrators*, J. Comput. Appl. Math. **161** (2003), no. 1, 203–216.
- [32] S. Maset and M. Zennaro, *Unconditional stability of explicit exponential Runge-Kutta methods for semi-linear ordinary differential equations*, Math. Comp. **Volume 78** (2009), 957–967.
- [33] B.V. Minchev, *Exponential integrators for semilinear problems*, Ph.D. thesis, University of Bergen, Department of Informatics, 2004.
- [34] B.V. Minchev and W.M. Wright., *A review of exponential integrators for first order semi-linear problems*, Tech. Report 2/05 (2005).
- [35] C. Moler and C. Van Loan, *Nineteen dubious ways to compute the exponential of a matrix*, SIAM Review **20** (1978), 801–836.
- [36] C. Moler and C. Van Loan, *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Review **45** (2003), no. 1, 3–49.
- [37] J. Niesen and W.M. Wright, *A Krylov subspace algorithm for evaluating the φ -functions appearing in exponential integrators*, (2009).
- [38] A. Ostermann, M. Thalhammer, and W.M. Wright, *A class of explicit exponential general linear methods*, BIT **46** (2006), no. 2, 409–431.
- [39] K. Radhakrishnan and A.C. Hindmarsh, *LSODE: Livermore solver for ordinary differential equations*, (1987).
- [40] N. Rattenbury, *Almost Runge-Kutta methods for stiff and non-stiff problems*, Ph.D. thesis, The University of Auckland, 2005.
- [41] T. Schmelzer, *Talbot quadratures and rational approximations*, Bit Numerical Mathematics, 653–670(18).
- [42] T. Schmelzer and L.N. Trefethen., *Evaluating matrix functions for exponential integrators via Carathéodory-Fejér approximations and contour integrals*, Published Online (2007).
- [43] L.F. Shampine and M.W. Reichelt, *The matlab ode suite*, SIAM J. Sci. Comput. **18** (1997), 1–22.
- [44] R.B. Sidje, ACM Trans. Math. Softw., no. 1, 130–156.
- [45] A. Talbot, *The accurate numerical inversion of Laplace transforms*, IMA J Appl Math **23** (1979), no. 1, 97–120.
- [46] J.A.C. Weideman, *Optimizing Talbot’s contours for the inversion of the Laplace transform*, SIAM J. Numer. Anal. **44** (2006), no. 6, 2342–2362.
- [47] J.A.C. Weideman and L.N. Trefethen, *Parabolic and hyperbolic contours for computing the Bromwich integral*, Mathematics of Computation **76** (2007), 1341–1356.