ASYNCHRONOUS AND EXPONENTIAL BASED NUMERICAL SCHEMES FOR POROUS MEDIA FLOW

by

Daniel Stone

Submitted for the degree of
Doctor of Philosophy

Department of Mathematics
School of Mathematical and Computer Sciences
Heriot-Watt University

September 2015

Abstract

A great many physical phenomena are modelled by partial differential equations (PDEs), and numerical schemes often have to be employed to approximate the solutions to these equations where analytical solutions cannot be found. We develop and analyse here new schemes belonging to two broad classes, schemes that are asynchronous, and exponential integrators. We apply these schemes to test models of advection-diffusion-reaction processes that occur in porous media flow.

Asynchronous schemes allow different parts of the physical domain to evolve at different rates. We develop a class of asynchronous schemes that progress by discrete events, where a single event is the transfer of a unit of mass through the domain, according to the local flux. These schemes are intended to focus computational effort where it is most needed, as a high local flux will cause the algorithm to automatically take more events in that part of the domain. We develop the simplest version of this scheme, and then develop further schemes by adding modifications to address potential shortcomings. Numerical experiments indicate a number of interesting relations between the parameters of these schemes. Particularly, the error of the schemes seems to be first order with respect to a control parameter we call the mass unit. Some analysis is conducted which can pave the way towards robust theoretical understanding of these schemes in the future.

Exponential integrators are time stepping schemes which exactly solve the linear part of a semilinear ODE system. This class of schemes requires the approximation of a matrix exponential in every step, and one successful modern method is the Krylov subspace projection method. We investigate, through analysis and experiment, the effect of breaking down a single timestep into multiple substeps, recycling the Krylov subspace to minimise costs. Our results indicate that this can increase accuracy and efficiency.

We show the results of an investigation into developing a class of 'semi-exponential' Runge-Kutta type schemes, which use an exponential integrator for the initial stage and then essentially fulfil classical order conditions for the remaining stages.

Finally, we return to the concept of asynchronicity in a different form. With the advent of massively parallel machines, there is increasing interest in developing domain-decomposition type schemes that are robust to random failures or delays in communication between processing elements. This is because in massively parallel machines, communication between processors is likely to be the significant bottleneck in execution time. Recently the effect of such communication delay with a simple domain-decomposed Euler timestepping solver applied to a linear PDE has been investigated with promising results. Here, inspired by exponential integrators, we investigate the natural extension of this, by replacing the Euler timestepping with the evaluation of the appropriate matrix exponential on the sub-domain. We have performed experiments simulating the communication delay and the results are also promising.

Acknowledgements

For my parents, who gave me everything. May this make you proud.

I am extremely grateful to my supervisors, Professors Gabriel Lord and Sebastian Geiger, for all their support and guidance.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Partial differential equations (PDEs) are used in physics, chemistry, biology, engineering and applied mathematics to model a vast array of behaviour. They express the temporal and spatial rates of change of some quantity of interest, and their solution subject to specific domains and initial and boundary conditions allows the quantity of interest to be usefully predicted in real world scenarios. Examples of PDEs include the Navier Stokes equations for fluid flow, where the quantity of interest may be the velocity or density of the fluid, the equations of electromagnetism, and the equations modelling the flow of fluid through a porous medium. The latter are of primary interest here, see §1.2.

Closed form solutions cannot normally be analytically derived for many useful PDE models, thus it is necessary to use algorithms to approximate the solutions, given specific domains and initial and boundary conditions. The development, analysis, implementation and testing of the approximation algorithms constitutes the field of numerical analysis.

A plethora of other approaches exist, but a traditional approach to numerical approximation of a PDE is as follows. First the domain is gridded or meshed into discrete cells, volumes or nodes. An approximation method for the spatial derivatives in the PDE is applied and the result is a system of ordinary differential equations (ODEs); this is called the method of lines. Spatial discretisation methods include the simplest finite difference method ([2], Chapter 12), the finite element method [3], and the finite volume method, which we discus further in §1.3. These are the classical 'big three' spatial discretisation methods; an example of a more modern development is the mimetic finite difference method [4]. An ODE has only temporal derivatives, and the ODE system is called the semidiscretised equation.

Schemes for ODEs range from the simplest Euler method, which approximates the temporal derivative by the simplest possible finite difference, to Runge-Kutta schemes (e.g. [5] Chapter 3), to modern schemes such as exponential integrators, which are introduced in §1.4. Some of our contributions are schemes within this class.

In this thesis our contribution will be to new schemes for approximating semidiscretised PDE systems. In this chapter we introduce necessary concepts. In Chapters 2 through 4 we develop new schemes within the class of Asynchronous schemes which proceed by discrete events at different parts of the domain, according to the local rate of activity. Chapter 2 describes the schemes, Chapter 3 reports our numerical experiments with these schemes, and Chapter 4 contains some analysis. Chapter 5 contains a new type of scheme within the class of exponential integrators, where an approximation of part of the solution is recycled across multiple substeps in order to increase accuracy. In Chapter 6 we develop a class of Runge-Kutta type schemes which share some properties with exponential integrator schemes but sacrifice others for faster run time. Chapter 7 deals with schemes which are asynchronous in a different way to those of Chapter 2 through 4. In Chapter 7 we look at schemes which allow asynchronicity as a result of communication delay in a parallel system.

This introductory chapter is organised as follows. In §1.2 we introduce the basic concepts of porous media flow and in §1.2.1 we introduce the model equation which we will use a primary test for our new schemes. In §1.3 we introduce the Finite Volume method for spatial discretisation, which we will make use of for our tests. Exponential integrators are introduced in §1.4, and some important numerical analysis concepts are discussed in §1.5.

1.2 Porous Media Flow

Here we introduce porous media flow, as described in, for example, [6]. A porous medium is a solid material with connected internal void spaces (pores) through which a fluid can flow. The complete connected space of voids is called the pore geometry. One example is soil, which consists of solid soil grains packed together. The gaps between the grains form a pore geometry and allow the bulk movement of fluid through a solid mass of soil, for example groundwater flow. Geological reservoirs are rock formations with connected void spaces in which water, oil and natural gas can exist and flow. The extraction of fluid hydrocarbon resources from the earth is thus a problem of porous media flow. A hydrothermal reservoir is a geological reservoir containing heat; pumping a fluid into, through, and out of a hydrothermal reservoir for the purpose of geothermal energy generation is also a porous media flow problem. Other materials, including bones and the electrolyte in hydrogen fuel cells, can be characterised as porous media.

The void spaces in porous media are not simply or linearly connected, and the flowing fluid will traverse a tortuous path. Also, the scale of the void spaces can be microscopic while the complete porous medium of interest may range in scale up to kilometres (e.g. an entire geological reservoir). Thus modelling and simulating flow through a realistic medium by taking into account every void space is generally impossible. It is extremely difficult to measure the pore geometry or the state of a fluid at any specific point of the geometry. Typically, bulk scale parameters of the flow or the medium are observed and used, and empirical models for the bulk scale flow developed based on these observations. The most famous such model is Darcy's law, which we will introduce shortly. Another way to develop models is to consider the flow through some tractable but representative pore geometry and then upscale this system into an averaged bulk scale model; see for example [7] and the references therein. At the bulk scale the properties of the flow are expressed by continuous parameters; this is called the continuum approximation. The first is the porosity $\phi(\mathbf{x})$, where $\mathbf{x} = (x, y, z)^T$ is the coordinate for a spatial position in the domain. The porosity is defined as the ratio of the void space to the total volume, within a small

Representative Elementary Volume (REV) centred at $\mathbf{x}$. The REV must be small enough for the value of $\phi(\mathbf{x})$ to be specific to $\mathbf{x}$, but not so small that the definition stops making sense. Consider decreasing the size of the REV. As the size is reduced, ϕ will initially appear to approach a limiting value. However, as the volume passes below a certain critical value, the porosity will start to fluctuate erratically due to the fine scale of the pore geometry, before approaching unity or zero as the volume reaches zero. See Figure 1.3.2 in [6] or Figure 5 in [8] for example. The true value of ϕ can be taken to be the value at the critical volume, or the extrapolated limit as the volume goes to zero, if the fluctuation effects were not present.

The second continuum parameter for the medium is the permeability $\mathbf{k}(\mathbf{x})$. This quantifies the extent to which a pressure gradient in the fluid in the medium will induce flow in each of the three principal directions (i.e., the x, y, z directions in three dimensions). The permeability is a square matrix, also referred to as a tensor in this context, with size equal to the dimensions of the domain. In three dimensions, $\mathbf{k}(\mathbf{x}) \in \mathbb{R}^{3\times 3}$. If $\mathbf{k}(\mathbf{x}) = k(\mathbf{x})I$, where I is the identity matrix and $k(\mathbf{x})$ is a scalar, then the permeability field is called isotropic and flow is not favoured in any particular direction, at any point in the domain.

Variables typical of fluid flow models are also continuous variables of the flowing fluid in a porous media flow models. At a point $\mathbf{x}$, the pressure of the fluid is $p(\mathbf{x})$; the velocity is $\mathbf{v}(\mathbf{x})$; the density is $\rho(\mathbf{x})$, and the viscosity of the fluid is μ.

We now introduce the most famous and basic equation of porous media flow, Darcy's law, discovered in 1856. See also [8]. Darcy's law typifies the empirical and continuum approach discussed above. Darcy was an engineer; the experiment and equation were reported in four pages of an appendix to a 647-page report on modifications to the public water systems of Dijon, France. He was trying to design a filter. The experiments consisted of observing the bulk flow rate of water through a column of packed sand. The flow rate was related empirically to the length of the column and the pressure at each end, and a coefficient K, which describes how properties of the medium (the sand in this case) and the fluid affect the average flow, and later practitioners showed can be related to the permeability $\mathbf{k}$ of the medium and the density ρ and viscosity μ of the fluid. The original expression is,

$$q = -\frac{K}{l}(h_2 - h_1),$$

where q is the rate of flow per unit volume, l is the length of the column, and h_1 and h_2 are the heights of water in manometers above and below the column. A manometer measures pressure, so the $(h_2 - h_1)$ part describes a pressure difference. This is a scalar equation since the bulk flow in only one direction is considered. Subsequent work by practitioners led to expressions for velocity fields in three dimensional domains, in terms of the gradient of the pressure field, such as,

$$\mathbf{q}(\mathbf{x}) \equiv \mathbf{v}(\mathbf{x})\phi(\mathbf{x}) = -\frac{\mathbf{k}(\mathbf{x})}{\mu}(\nabla p(\mathbf{x}) + \rho(\mathbf{x})\mathbf{g}), \tag{1.1}$$

where $\mathbf{q} \equiv \mathbf{v}\phi$ is the Darcy velocity. The vector $\mathbf{g}$ is the vector representing acceleration due to gravity. We see how the matrix $\mathbf{k}(\mathbf{x})$ controls how the gradient of $p(\mathbf{x})$ relates to the flow $\mathbf{q}(\mathbf{x})$, as well as the effect of gravity $\rho(\mathbf{x})\mathbf{g}(\mathbf{x})$. Both $\nabla p(\mathbf{x})$ and $\rho(\mathbf{x})\mathbf{g}(\mathbf{x})$ describe forces on the fluid and (1.1) describes how these forces, at some point $\mathbf{x}$, affect the flow velocity $\mathbf{q}(\mathbf{x})$ at that point. The matrix form of $\mathbf{k}(\mathbf{x})$ helps describe the tortuous paths in the porous medium, as, for example, a large gradient of $p(\mathbf{x})$ in one direction may induce a large flow velocity $\mathbf{q}(\mathbf{x})$ in a completely different direction, depending on the components of $\mathbf{k}(\mathbf{x})$. This would represent flow through the medium being much easier in the other direction due to the pore geometry.

1.2.1 Conservation Equation and the Advection Diffusion Reaction (ADR) Equation

Conservation equations are important for modelling flows in porous media, and a particular example we shall use often in our tests is the Advection Diffusion Reaction (ADR) equation. Other examples of conservation equations include the relation of current density to charge density in electromagnetism, the heat equation, and the relation of fluid density to velocity in fluid flow models (we note a porous media equivalent of this shortly, where the velocity is given by Darcy's law).

Let U be some conserved extensive property of a substance, such as mass, amount, or heat. Define u with respect to U by

$$\int_V \phi u dV = \text{Sum of all } U \text{ in } V,$$

for some closed volume V. Note the inclusion of the porosity term ϕ, ensuring that only the pore space within V is taken into account. Then u is a density or concentration function corresponding to U. Let the evolution of U be determined by a flux $\mathbf{J}(x,y,z) = (J_x(x,y,z), J_y(x,y,z), J_z(x,y,z))^T$ and a non-conserving source or reaction term Q. The flux models the flow and transport of U; it is defined such that in unit time the amount of U that will flow through a surface S is

$$\oint_S \mathbf{J}(x,y,z) d\mathbf{S}.$$

The source or reaction term models any other processes that affect U. For example, the mass could increase or decrease at a certain rate due to chemical or physical processes; or radioactive decay may destroy mass or generate heat; or the substance may be simply inserted to or removed from the system at certain parts of the domain, as with wells in geological reservoirs. Per unit time, the rate of change of U at a point due to non-conserving processes is given by $Q(x,y,z,t,u)$.

The evolution of u, the density of U, is then given by the conservation (also called continuity) equation,

$$\frac{d(\phi u(x,y,z,t))}{dt} = \nabla \cdot (\mathbf{J}(x,y,z,t)) + \phi(x,y,z) Q(x,y,z,t,u). \tag{1.2}$$

We show a standard derivation of (1.2) here; another derivation specific to porous media flow can be found in §4.2 of [6]. For the derivation we consider an infinitesimal cube of sides Δx, Δy, Δz as shown in Figure 1.1 a). First we note that the definitions of u, Q and $\mathbf{J}$ give us, for the point $(x_c, y_c, z_c)^T$ at the centroid at the cube,

$$\frac{d}{dt}\int_V \phi u dV = \oint_S \mathbf{J}(x_c, y_c, z_c) \cdot d\mathbf{S} + \int_V \phi Q dV, \tag{1.3}$$

where V is now the cube and S its surface. We can express the term $\oint_S \mathbf{J}(x_c, y_c, z_c) \cdot d\mathbf{S}$ as a volume integral as follows. Note this is essentially a proof of the divergence theorem.

The surface integral over the cube is the sum of the surface integrals over its six faces. We consider the three pairs of opposite faces - show in Figure 1.1 b), c) and d). The sum of the surface integrals over the two faces perpendicular to the x axis is (highlighted in plot b),

$$\int_{z_c-\Delta z}^{z_c+\Delta z} \int_{y_c-\Delta y}^{y_c+\Delta y} \left(J_x(x_c + \Delta x, y_c, z_c) - J_x(x - \Delta x_c, y_c, z_c)\right) dydz.$$

We can invoke the fundamental theorem of calculus on J_x,

$$J_x(x_c + \Delta x, y_c, z_c) - J_x(x_c - \Delta x, y_c, z_c) = \int_{x_c-\Delta x}^{x_c+\Delta x} \frac{dJ_x}{dx} dx.$$

Thus we have that the sum of the surface integrals over these two faces is

$$\int_{z_c-\Delta z}^{z_c+\Delta z} \int_{y_c-\Delta y}^{y_c+\Delta y} \int_{x_c-\Delta x}^{x_c+\Delta x} \frac{dJ_x}{dx} dxdydz = \int_V \frac{dJ_x}{dx} dV.$$

Using the same argument, we see that the two faces highlighted in Figure 1.1 c) yield a total surface integral of $\oint_V \frac{J_y}{dy} dV$, and the faces highlighted in d) yield $\oint_V \frac{dJ_z}{dz} dV$ so that the total surface integral is

$$\oint_S \mathbf{J}(x_c, y_c, z_c) d\mathbf{S} = \int_V \nabla \mathbf{J}(x_c, y_c, z_c, t) dV.$$

Inserting this into (1.3) and rearranging gives us

$$\int_V \left(\frac{d(\phi u)}{dt} - \nabla \mathbf{J}(x_c, y_c, z_c, t) - \phi Q\right) dV = 0.$$

The integrand must be zero, and by taking $(x_c, y_c, z_c)^T$ to be an arbitrary point (x, y, z), (1.2) follows.

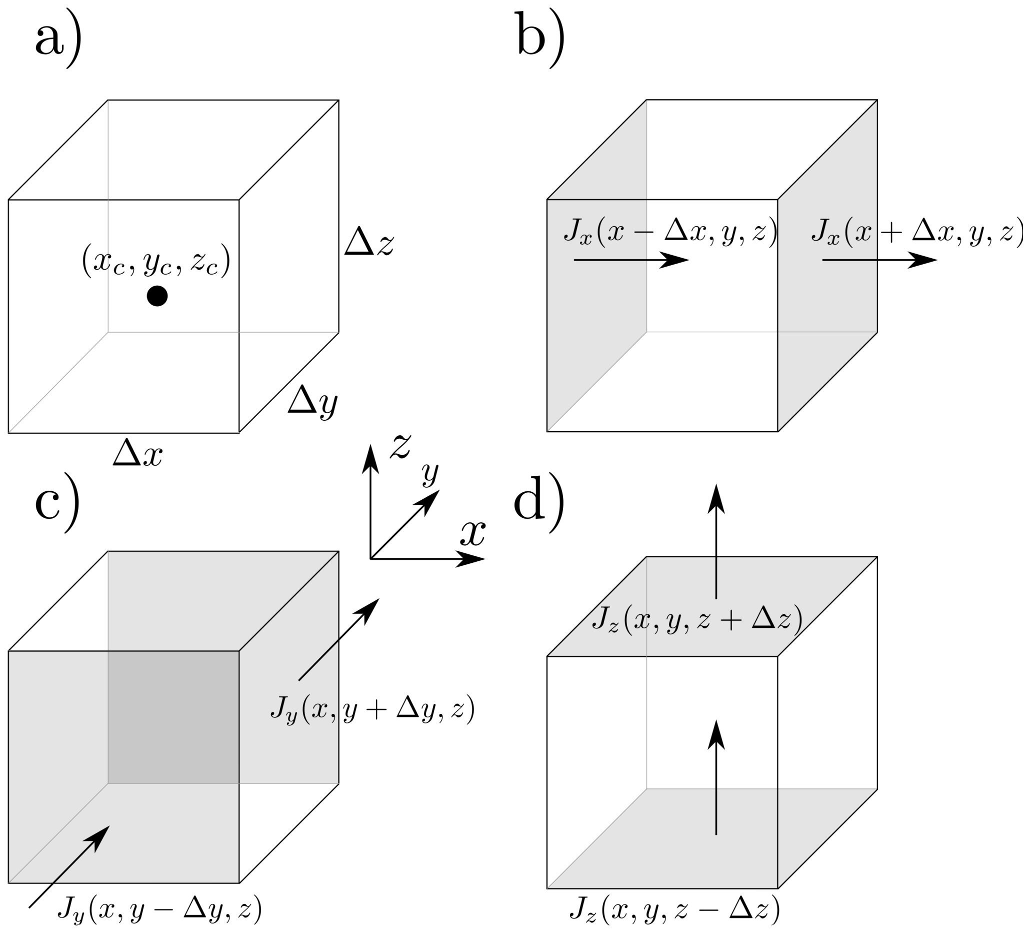

a)
b)
c)
d)
(x_c, y_c, z_c)
Δz
Δy
Δx
$J_x(x - \Delta x, y, z)$
$J_x(x + \Delta x, y, z)$
z
y
x
$J_y(x, y + \Delta y, z)$
$J_y(x, y - \Delta y, z)$
$J_z(x, y, z + \Delta z)$
$J_z(x, y, z - \Delta z)$

Figure 1.1: The representative elementary volume used to derive the conservation equation. a) The dimensions and centroid of the volume. b) Flow in and out on faces perpendicular to x. c) Flow in and out on faces perpendicular to y. d) Flow in and out on faces perpendicular to y.

We use the Advection Diffusion Reaction (ADR) equation model to test our schemes, which we derive from (1.2). This models the transport of a dissolved species in a solute flowing in a porous medium. To (1.2) we apply the following. We let U be the mass, m of some dissolved species. From this it follows that u is the concentration, c. We let Q be some arbitrary reaction term R. Two physical processes define the flux $\mathbf{J}$.

The first is diffusion, modelled by Fick's law (see e.g., [9] §1.2 or [10] §16.2),

$$\mathbf{J}_D = -\mathbf{D}\nabla c,$$

where the matrix $\mathbf{D}$ is the diffusivity matrix. Moving forward we will consider the simplified case where $\mathbf{D} = D\mathbf{I}$, where $\mathbf{I}$ is the 3×3 identity matrix. Diffusion is the net movement of the solute from regions of high to low concentration. The solute molecules are constantly undergoing Brownian motion; those whose velocities happen to be in the direction of a lower concentration have a smaller probability of collision, thus longer mean free paths, thus the net movement of molecules is in the direction of lower concentration. The second physical process is advection. Let the solvent be flowing with velocity field $\mathbf{v}$. Then solvent molecules may collide with solute molecules and move them in the direction of the flow. Collisions are more likely with higher concentration, so the flow of the solvent causes a flux,

$$\mathbf{J}_A = c\mathbf{v}.$$

The total flux of the dissolved species is $\mathbf{J} = \mathbf{J}_D + \mathbf{J}_A$. Inserting this into (1.2) gives us

$$\frac{d\phi c}{dt} = \nabla \cdot (D\nabla c + \phi c\mathbf{v}) + \phi R(c), \tag{1.4}$$

the advection diffusion reaction equation. A simplification of (1.4) which we will make use of is to set $\phi = 1$ everywhere; this is done in our tests.

Using (1.1) we can write a conservation equation for the mass of the flowing fluid, as in §1.2.1, letting the density be $u \equiv \rho$ and the flux be $\mathbf{J} \equiv \mathbf{q}$ in (1.2),

$$\frac{d\phi\rho}{dt} = \nabla \cdot \mathbf{q} + Q = -\nabla \cdot \left(\frac{\mathbf{K}}{\mu}(\nabla p + \rho\mathbf{g})\right) + Q, \tag{1.5}$$

where Q models any sources or sinks of the fluid. A simplification of (1.5) for the case of no sources or sinks ($Q \equiv 0$), negligible gravitational effects ($\rho\mathbf{g} \approx 0$), and steady state ($\frac{d\phi\rho}{dt} = 0$), is

$$0 = -\nabla \cdot \left(\frac{\mathbf{K}}{\mu}\nabla p\right). \tag{1.6}$$

1.3 The Finite Volume method

We introduce here the Finite Volume spatial discretisation method. This method is appropriate for porous media flow as it closely follows the physics underlying the PDE, as will be seen. The method also conserves mass and flux. By mass conservation it is meant that the total mass in the system will not be spuriously changed by the method. By flux conservation it is meant that the approximated flux approximation across one face is consistent for both of the cells adjacent to that face. For more detail see also the expository paper [11] and references therein, or textbooks such as [12, 13, 14].

The method is appropriate for general conservation equations (1.2); we will focus on (1.4) in our description. It is necessary to ensure that (1.4) is well posed, that is, the solution c exists and is unique for given initial data $c_0(\mathbf{x}) \equiv c(0,\mathbf{x})$. For theory of well-posed PDEs see [15, 16, 17], for example. Typically it is necessary to assume that the non-linearity satisfies a Lipschitz condition.

Definition 1. Lipschitz Continuity *A function $F(x)$ is said to be Lipschitz continuous if there exists positive $C_L \in \mathbb{R}$ such that*

$$||F(y) - F(x)|| \leq C_L||y - x||.$$

There are also requirements on the linear part of (1.4). Setting the operator

$$A\cdot \equiv \nabla D \nabla \cdot + \nabla v \cdot,$$

so that (1.4) can be written $\frac{dc}{dt} = Ac + Q$, it is required that A be the generator of a semigroup. See for example [17] Definition 1.3.3 or [16] Chapter 4 for this definition and conditions for A to have this property. We will assume throughout that sufficient conditions are fulfilled such that (1.4) is well posed. Similar conditions are required for convergence proofs for finite volume discretisations; see for example [11] or [18] and the references therein.

We now describe the finite volume discretisation of (1.4).

Consider the spatial domain Ω divided into a mesh of non overlapping polyhedral volumes; we refer to the volumes as cells. Let there be J cells in total and let each cell have unique index $j \in \{1, 2, \ldots, J\} = \mathcal{C}$, where $\mathcal{C}$ is simply the set of all cell indexes. We denote the volume of a cell j by V_j and the cell itself by ω_j, such that $\Omega = \bigcup_{j=1}^{J} \omega_j$. Let K be the total number of faces in the grid and let each face have a unique index $k \in \{1, 2, \dots K\} = \mathcal{F}$. For a cell $j \in \mathcal{C}$, let the set $\mathcal{F}_j \subset \mathcal{F}$ be the set of faces for that cell. For a face $\mathfrak{f}_k$ with index k let A_k be its area.

We consider integrating the conservation law (1.2) over one cell j.

$$\int_{\omega_j} \frac{d\phi u(x,y,z,t)}{dt} dV = \int_{\omega_j} \nabla \cdot (\mathbf{J}(x,y,z,t)) dV + \int_{\omega_j} \phi Q(x,y,z,t,U) dV. \tag{1.7}$$

We invoke the divergence theorem on the first term on the right hand side,

$$\int_{\omega_j} \nabla \cdot (\mathbf{J}(x,y,z,t)) dV = \int_A \mathbf{J}(x,y,z,t) \cdot dS,$$

where the surface integral is over the surface of ω_j. Since the cell is polyhedral the surface integral can be expressed as the sum of the surface integrals on each of the faces,

$$\int_A \mathbf{J}(x,y,z,t) \cdot dS = \sum_{k \in \mathcal{F}_j} \int_{\mathfrak{f}_k} \mathbf{J}(x,y,z,t) \cdot dS.$$

Let $\hat{Q} \equiv \phi Q$, and assume that u and $\hat{Q}$ are constant in the cell. That is, let $\mathbf{x}_j$ be the vector of (x,y,z) at the centre of the cell j, then inside the cell we make the approximation, $u(x,y,z,t) \approx u(\mathbf{x}_j, t)$ and $\hat{Q}(x,y,z,t) \approx \hat{Q}(\mathbf{x}_j, t)$. Then (1.7) can be written as

$$\frac{d\phi u(\mathbf{x}_j, t)}{dt} = \frac{1}{V_j} \sum_{k \in \mathcal{F}_j} \int_{\mathfrak{f}_k} \mathbf{J}(x,y,z,t) \cdot dS + \hat{Q}(\mathbf{x}_j, t, U),$$

where we have divided through by the volume of V_j. An approximation for the flux terms on each face in $\mathcal{F}_j$ needs to be found. That is, we must approximate on each face,

$$\mathbf{J}(x,y,z,t) \cdot \mathbf{n},$$

where $\mathbf{n}$ is the outward unit normal vector of the face. The simplest methods are the two point and upwind methods, which we now describe.

Consider the ADR equation so that

$$J = D\nabla u + \phi u \mathbf{v}.$$

The two point approximation for the term $D\nabla u \cdot \mathbf{n}$ is essentially a finite difference approximation for the spatial derivative term. If j_1, j_2 are the two cells adjacent to the face, then we approximate,

$$D\nabla u \cdot \mathbf{n} \approx \bar{D}\frac{u_{j_2} - u_{j_1}}{\Delta x},$$

where Δx is the length of the line joining the two centroids of cells j_1 and j_2, u_{j_1} and u_{j_2} are the values of u at these centroids, and $\bar{D}$ is an average value D on the face. Note that we only consider Cartesian grids here (in which cells are simple cubes), so that the line joining the centroids of cells j_1 and j_2 is always perpendicular to the face it passes through. Also note that for $\bar{D}$, in order to ensure convergence to the underlying PDE as the cell volume is reduced to zero, a harmonic mean is used;

$$\bar{D} = \frac{2D_{j_1}D_{j_2}}{D_{j_2} + D_{j_2}}.$$

To approximate the advection term $u\mathbf{v} \cdot \mathbf{n}$ the upwind approximation is used; see for example [13, §5.6]. See also [14, §8.3.4] for a justification of this method in terms of stability. Let $\mathbf{v}$ be the velocity at the centre of the face. This may need to be interpolated if velocities are only available at cell centroids. The approximation then depends on the direction of the velocity vector with respect to the normal vector to the face. If $\mathbf{n}$ is in the direction away from cell j_1 and towards j_2, then,

$$u\mathbf{v} \cdot \mathbf{n} \approx \begin{cases} u_{j_1}\mathbf{v} \cdot \mathbf{n} \text{ if } \mathbf{v} \cdot \mathbf{n} > 0 \\ u_{j_2}\mathbf{v} \cdot \mathbf{n} \text{ otherwise,} \end{cases} \tag{1.8}$$

and vice versa if $\mathbf{n}$ points from j_2 to j_1.

The two point flux approximations fail to consistently approximate the underlying PDE unless the mesh is regular and Cartesian. For more complicated grids, approximations for the flux using data from more than two cells must be used (so called Multi Point Flux Approximation schemes, MPFA, see for example [19] and the references therein).

Let the approximation of the flux on face k be $\tilde{J}_k$. The approximation of (1.7) is then

$$\frac{d\phi u(\mathbf{x}_j,t)}{dt} = \frac{1}{V_j}\sum_{k\in\mathcal{F}_j} A_k\tilde{J}_k + \hat{Q}(\mathbf{x}_j,t,U). \tag{1.9}$$

The approximation of the flux $\tilde{J}_k$ is a linear combination of the values of u in cell j and others. We can accumulate the approximation (1.9) into an ODE system. Let $\mathbf{u} \in \mathbb{R}^J$ be the vector with jth entry $\phi u(\mathbf{x}_j,t)$. Similarly let $\hat{\mathbf{Q}} \in \mathbb{R}^J$ be the vector with jth entry $\hat{Q}(\mathbf{x}_j,t)$. Let the matrix $L \in \mathbb{R}^{J\times J}$ be the matrix representing the linear combinations from the flux approximation parts of (1.9). Then the finite volume method yields a semilinear ODE system,

$$\frac{d\mathbf{u}}{dt} = L\mathbf{u} + \hat{\mathbf{Q}}. \tag{1.10}$$

1.4 Exponential Integrators

A PDE such as the ADR (1.4) can be semidiscretised into an semilinear ODE system using a spatial discretisation such as the Finite Volume method, as seen in the previous section. The resulting ODE system can be expressed as

$$\frac{d\mathbf{u}}{dt} = L\mathbf{u} + F(\mathbf{u},t). \tag{1.11}$$

If the spatial grid has J cells or nodes, then $\mathbf{u} \in \mathbb{R}^J$ is a vector of the approximation to u in each cell (from now on in this section we will say cell instead of cell or node as we mainly deal with Finite Volumes). The matrix $L \in \mathbb{R}^{J\times J}$ represents the linear parts of the discretised equations, while the vector $F \in \mathbb{R}^J$ is a vector of the non-linear contributions in each cell. For example when the ADR (1.4) is approximated by (1.11) the approximation of the diffusion and advection terms produce linear equations, while the approximation of the reaction term produce a nonlinear term. In a general conservation equation (1.2), the flux term can be approximated as a linear combination of data from two or more cells, while the reaction term produces a nonlinear contribution to the ODE semidiscretised system.

Exponential integrators are a class of schemes which approximate (1.11) by exactly solving the linear part. This means that if F were zero or constant, then the exponential integrator is exact for the ODE system. We give details on the construction of these schemes shortly in §1.4.1. A characterising aspect of exponential integrators is that they require the evaluation or approximation of a matrix exponential function of L at each timestep. The concept of exponential integrators has existed since at least the 1960s, but were considered impractical until the start of the current century. This is because methods to efficiently approximate the matrix exponential functions were not available until then.

Exactly solving the linear part of (1.11) grants exponential integrators some beneficial properties. Classical CFL conditions (See e.g., §4.2 in [20]) are avoided since $e^{\Delta t L}$ is dense even when L is typically sparse; thus at each timestep each cell updates based on information from every other cell (that is, the physical domain of dependence is always within the domain of dependence of the scheme). Exponential integrators tend to have greater stability regions than other explicit schemes, see for example the analysis in §3 in [21]. Exponential integrators are also often useful for 'stiff' problems. A problem is termed stiff if the timestep is limited by stability requirements, or if there are different processes being modelled that evolve at vastly different rates. The timestep in that case would need to be small enough to resolve the fastest of these processes accurately. Either way there is a timestep restriction. If the stiffness is in the processes modelled in the semilinear part of the ODE (e.g., advection or diffusion), then the exponential integrator will not have problems with stiffness as the linear part is solved exactly. If the problem is stiff in the sense of a stability restriction; exponential integrators will benefit from their large regions of stability.

An overview for exponential integrators has been published in Acta Numerica [22]. Another overview is [23], and useful references can be found in [24]. A major class of exponential integrators are the multistep Exponential Time Differencing (ETD) schemes, first developed in [21], which also contains some analysis to justify the claims about stability. Other classes include the Exponential Euler Midpoint method [25] and Rosenbrock type methods [26, 27].

Extending the concept Runge-Kutta (RK) schemes to exponential integrators has also been a significant area of research. Analogous to how classical RK schemes use multiple internal stages in order to attain a higher order of accuracy, Exponential Runge Kutta (ERK) schemes use multiple internal stages, thus requiring multiple evaluations of the matrix exponential function typically. Hochbruck and Osterman [28, 29, 30] have derived conditions for ERK schemes which guarantee the stiff order of the scheme (see §1.5.3). Tokman has developed a class of ERK schemes [31, 32, 33] which instead satisfy classical order conditions but require fewer matrix exponential function evaluations. We investigate a different approach to the same objective in Chapter 4.

Exponential integrators have been successfully applied to geological reservoirs, see for example [34]. Lord [35] and Tambue [36, 37, 38, 39] have developed exponential integrators for stochastic ODE systems, used to model, for example, geological reservoirs with highly heterogeneous permeability. See also the Thesis [40].

A parallel in time exponential integrator has been developed, named Paraexp [41] and based on the parallel in time scheme Parareal [42]. This requires the reaction term to be autonomous of u, i.e. $F = F(t)$. Another report on parallelization is [43], where the matrix-matrix and matrix-vector multiplications necessary for the Leja-point interpolation [44] method for approximating the matrix exponential, are performed in parallel in a standard way.

Packages released for exponential integrators include [45] and the expint package [46].

1.4.1 φ-Functions

A class of exponential functions called $\varphi-$functions are key to the formulation of exponential integrators. They also appear in some of our Asynchronous schemes in Chapter 2. We provide a definition here.

The first $\varphi-$function is $\varphi_0(\cdot)$ and is defined as simply the exponential function

$$\varphi_0(z) \equiv e^z.$$

The argument z can be any operator for which the exponential can be defined, but for our purposes it will always be a matrix; $z \in \mathbb{R}^{n\times n}$. The exponential is defined as the series,

$$e^z \equiv \sum_{i=0}^{\infty} \frac{z^i}{i!}. \tag{1.12}$$

The rest of the $\varphi-$functions are defined iteratively,

$$\varphi_{n+1}(z) \equiv z^{-1}\left(\varphi_n(z) - \frac{1}{n!}\right), \tag{1.13}$$

where I is the $n \times n$ identity. Of particular interest is $\varphi_1(\cdot)$;

$$\varphi_1(z) = z^{-1}\left(e^z - I\right) = \sum_{i=0}^{\infty} \frac{z^i}{(i+1)!}. \tag{1.14}$$

The series expression in (1.14) follows from the series expression of the exponential (1.12). Indeed, an inductive argument using (1.12) and (1.13) shows that for general $\varphi-$functions,

$$\varphi_m(z) = \sum_{i=0}^{\infty} \frac{z^i}{(i+m)!}. \tag{1.15}$$

We show how $\varphi-$functions relate to (1.11). First we follow the approach of [21]. Using an integrating factor $e^{-\Delta t L}$, the exact solution to (1.11) can be written as

$$\mathbf{u}(t+\Delta t) = e^{\Delta t L}\mathbf{u}(t) + \int_t^{t+\Delta t} e^{(t+\Delta t - s)L} F(\mathbf{u}(s), s)ds. \tag{1.16}$$

ETD schemes are derived for (1.11) by approximating the integral in (1.16). Consider the approximation that $F(\mathbf{u}(s), s)$ is constant over $[t, t+\Delta t]$, that $F(\mathbf{u}(s), s) \approx F(\mathbf{u}(t), t)$. Then integration gives

$$\mathbf{u}(t+\Delta t) \approx e^{\Delta t L}\mathbf{u}(t) + \Delta t \varphi_1(\Delta t L) F(\mathbf{u}(t), t),$$

which is equivalent to Equation (4) in [21] after applying the definition of φ_1 (the authors of [21] do not use $\varphi-$functions). Using this we can define the approximation method ETD1. Let $t_n \equiv n\Delta t$, define the approximation $\mathbf{u}_n \approx \mathbf{u}(t_n)$, and let $F_n \equiv$

$F(t_n, \mathbf{u}_n)$. Then ETD1 is the scheme,

$$\mathbf{u}_{n+1} = e^{\Delta t L}\mathbf{u}_n + \Delta t \varphi_1(\Delta t L)F_n, \tag{1.17}$$

ETD1 can be shown to be locally second order in Δt and globally first order in Δt ([21] and §1.5.1 here).

By approximating $F(\mathbf{u}(s), s)$ in (1.16) by polynomials of increasing degree, more accurate multistep methods can be derived. Consider the approximation $F(\mathbf{u}(s), s) \approx F(\mathbf{u}(t), t) + s\frac{F(\mathbf{u}(t),t) - F(\mathbf{u}(t-\Delta t), t-\Delta t)}{\Delta t}$ (i.e., a second order Taylor series approximation, with a finite difference approximation for the first derivative). Inserting this into (1.16) and integrating gives us,

$$\begin{aligned}\mathbf{u}(t + \Delta t) \approx e^{\Delta t L}\mathbf{u}(t) &+ \Delta t \varphi_1(\Delta t L)F(\mathbf{u}(t), t) \\ &+ \Delta t \varphi_2(\Delta t L)\left(F(\mathbf{u}(t), t) - F(\mathbf{u}(t - \Delta t), t - \Delta t)\right).\end{aligned}$$

This can be seen to be equivalent to Equation (6) in [21] after some rearrangement and application of the definitions of φ_1 and φ_2. Using the same notation as before, this approximation is the basis for the scheme ETD2,

$$\mathbf{u}_{n+1} = e^{\Delta t L}\mathbf{u}_n + \Delta t \varphi_1(\Delta t L)F_n + \Delta t \varphi_2(\Delta t L)\left(F_n - F_{n-1}\right). \tag{1.18}$$

ETD2 can be shown to be locally third and globally second order, [21].

We see how ETD schemes constructed in this manner include the exact solution of the linear part, i.e. $e^{\Delta t L}\mathbf{u}(t)$, while the approximation of the nonlinear part includes $\varphi-$functions. An alternative framework for deriving exponential integrators can be found by using the Taylor series expansion of F in (1.16). It can be shown [22] that evaluating the integral for each term results in an infinite series of $\varphi-$functions;

$$\mathbf{u}(t + \Delta t) = e^{\Delta t L}\mathbf{u}(t) + \sum_{k=1}^{\infty} \Delta t^k \varphi_k(\Delta t L)F^{(k-1)}(t, u(t)) + O(\Delta t^k). \tag{1.19}$$

From this we see that ETD1 and ETD2 are attempts to match or approximate the first one or two terms in the sum in (1.19), respectively.

Approximating the matrix exponential and functions of it like $\varphi-$functions is a notorious problem [47]. A classical technique is Padé approximation, which is only efficient for small matrices. Modern methods range from Taylor series methods making sophisticated use of scaling and squaring for efficiency, [48], to approximation with Faber or Chebyschev polynomials [49], [22] §4.1, interpolation on Leja points [50, 44, 43, 51, 52], to Krylov subspace projection techniques [53, 54, 45, 55, 56]. We are particularly interested in the Krylov subspace projection technique and introduce this in detail in Chapter 3.

A variation on ETD1 is to replace L with the Jacobian of the right hand side of (1.11), that is, the matrix J defined as

$$J \equiv \frac{\partial}{\partial u}\left(Lu + F(u,t)\right) = L + \frac{\partial F(u,t)}{\partial u}.$$

Equation (1.11) can then be re-written as

$$\frac{d\mathbf{u}}{dt} = J\mathbf{u} + (L\mathbf{u} + F(\mathbf{u},t) - J\mathbf{u}) \tag{1.20}$$

We then apply ETD1 (1.17) to (1.20), obtaining

$$\begin{aligned} \mathbf{u}_{n+1} &\approx e^{\Delta tJ}\mathbf{u}_n + \Delta t\varphi_1(\Delta tJ)\left(L\mathbf{u}_n + F(\mathbf{u}_n,t) - J\mathbf{u}_n\right) \\ &= \mathbf{u}_n + \Delta t\varphi_1(\Delta tJ)\left(L\mathbf{u}_n + F(\mathbf{u}_n,t)\right), \end{aligned} \tag{1.21}$$

where we have used $e^{\Delta tJ}\mathbf{u}_n = \mathbf{u}_n + \Delta t\varphi_1(\Delta tJ)J\mathbf{u}_n$, which follows from the $\varphi-$function definition at the start of this section. The scheme (1.21) is essentially the simplest Rosenbrock-type exponential method [26, 27], and is second order globally. We will refer to it as ROS2.

1.5 Numerical Analysis

1.5.1 Local and Global Error

We now discuss types of error that can be examined to analyse a numerical scheme; the local error and the global error; and the relationship between them. See also for example [57, §9.5] for a treatment specifically for linear systems, or [5, §2.11-2.12] for an example specifically for the Euler method.

Throughout the thesis we will use $||\cdot||$ to denote a generic norm. When the object x in the norm $||x||$ is a vector, then $||x||$ is understood to be a vector norm. If x is instead an operator or a matrix, then $||x||$ denotes the corresponding induced norm. More specific norms will be denoted as needed, for instance $||\cdot||_2$ is the standard Euclidean norm, or the corresponding induced operator or matrix norm.

Consider the ODE system (1.11) and its approximation $u_n \approx u(t_n)$, where $t_n = n\Delta t$, produced by a scheme with update rule,

$$u_{n+1} = \mathcal{L}u_n + \mathcal{N}(u_n), \tag{1.22}$$

where $\mathcal{L}$ is a linear operator and $\mathcal{N}(\cdot)$ is nonlinear. Consider the global error E_n defined as,

$$E_n \equiv u(t_n) - u_n. \tag{1.23}$$

Typically an essential step in examining the global error is to start with the local error, defined as,

$$\hat{E}_n = u(t_n) - \mathcal{L}u(t_{n-1}) - \mathcal{N}(u(t_{n-1})), \tag{1.24}$$

i.e. the error that would accumulate in one step of the method if $u(t_{n-1})$ were used instead of the approximation u_{n-1}. The global error can then be expressed iteratively, for we can combine (1.22), (1.23) and (1.24) to obtain

$$E_n = \hat{E}_n + \mathcal{L}E_{n-1} + \mathcal{N}(u(t_{n-1})) - \mathcal{N}(u_{n-1}).$$

From this it follows

$$E_n = \sum_{i=1}^{n} \mathcal{L}^{n-i} \hat{E}_i + \sum_{i=1}^{n-1} \mathcal{L}^{n-1-i} \left(\mathcal{N}(u(t_i)) - \mathcal{N}(u_i)\right).$$

An attempt is then made to bound the norm of E_n, i.e.,

$$||E_n|| = \sum_{i=1}^{n} ||\mathcal{L}^{n-i}|| \, ||\hat{E}_i|| + \sum_{i=1}^{n-1} ||\mathcal{L}^{n-1-i}|| \, ||\mathcal{N}(u(t_i)) - \mathcal{N}(u_i)||. \tag{1.25}$$

A bound for $||\mathcal{L}^{n-i}||$ is searched for. A bound for $||\hat{E}_i||$ can be found by examining the local error. For the nonlinear part, a bound can usually be found of the form $||\mathcal{N}(u(t_i)) - \mathcal{N}(u_i)|| \leq N(\Delta t, t_i)||F(u(t_i)) - F(u_i)||$, where $N(\Delta t, t_i)$ is a positive function specific to the scheme. For a bound on $||F(u(t_i)) - F(u_i)||$ it is usually necessary to assume that F is Lipschitz continuous, see Definition 1. If F is Lipschitz continuous, equation (1.25) will yield the bound,

$$||E_n|| = \sum_{i=1}^{n} ||\mathcal{L}^{n-i}|| \, ||\hat{E}_i|| + C_L \sum_{i=1}^{n-1} ||\mathcal{L}^{n-1-i}|| \, ||N(\Delta t, t_i)|| \, ||E_i||. \tag{1.26}$$

The goal is to put (1.26) in a form to which we can apply a discrete Gronwall Lemma. A result is described as being a discrete Gronwall lemma if it has the following properties. Given some nonnegative series $\{\epsilon_n\}$ and a bound on ϵ_n in terms of one or more ϵ_i where $i < n$, then there exists a bound on ϵ_n independent of any other members of the sequence $\{\epsilon_n\}$. For our purposes we are best served by simplified form of Lemma 4 from [28].

Lemma 1.5.1. Discrete Gronwall Lemma, [28], Lemma 4, simplified. *Given $\Delta t > 0$, $t_n = n\Delta t$, $a, b \geq 0$, and a sequence of nonnegative ϵ_n satisfying*

$$\epsilon_n \leq a\Delta t \sum_{i=1}^{n-1} \epsilon_i + b,$$

then ϵ_n satisfies

$$\epsilon_n \leq C_g b,$$

where C_g depends on all the parameters and on a $T \geq t_n$.

Proof. This is Lemma 4 from [28] with with $\rho, \delta = 0$. $\square$

Other forms of discrete Gronwall lemma are given in the paper [58]. Also see for example A.10 in [59]. Note that for the bound $T \geq t_n$ in Lemma 1.5.1 it is natural to simply use the final time of the simulation.

Example 1: Forward Euler

As a simple example of finding a bound on the global error, consider the forward Euler method, for which $\mathcal{L} = I$, the identity operator, and $\mathcal{N}(\cdot) = \Delta t F(\cdot)$. Assuming that F is Lipschitz continuous (Definition 1), then in (1.25) we have $||\mathcal{N}(u(t_i)) - \mathcal{N}(u_i))|| \leq \Delta t C_L ||u(t_i) - (u_i)|| = \Delta t C_L ||E_i||$, and (1.25) leads to the estimate,

$$||E_n|| \leq \Delta t C_L \sum_{i=1}^{n-1} ||E_i|| + \sum_{i=1}^{n} ||\hat{E}_i||,$$

and so,

$$||E_n|| \leq \Delta t C_L \sum_{i=0}^{n-1} ||E_i|| + \frac{T}{\Delta t} \max_i ||\hat{E}_i||,$$

where we have used $\sum_{i=1}^{n} 1 = n = \frac{t_n}{\Delta t} \leq \frac{T}{\Delta t}$. Applying Lemma 1.5.1 to this gives

$$||E_n|| \leq C_g \frac{T}{\Delta t} \max_i ||\hat{E}_i||.$$

The local error of the Euler method can be shown to be bounded by

$$||\hat{E}_i|| \leq \frac{\Delta t^2}{2} \max_t ||\frac{dF(u(t), t)}{dt}||,$$

by Taylor series expansion. That is, it is locally second order in Δt. The global error is first order in Δt,

$$||E_n|| \leq C_g \frac{T}{2} \Delta t \max_t ||\frac{dF(u(t), t)}{dt}||.$$

Example 2: ETD1

As another example we consider the global error of ETD1, see also for example Theorem 4.1 from [29]. For ETD1 we have $\mathcal{L} = e^{\Delta t L}$ and $\mathcal{N}(\cdot) = \Delta t \varphi_1(\Delta t L) F(\cdot)$. The global errors then satisfy the recursion,

$$E_n = \Delta t \sum_{i=0}^{n-1} e^{(n-i-1)\Delta t L} \varphi_1(\Delta t L) \left(F(t_i, u(t_i)) - F(t_i, u_i)\right) + \sum_{i=0}^{n-1} e^{i \Delta t L} \hat{E}_i.$$

Taking a norm gives us,

$$||E_n|| \leq \Delta t C_L \sum_{i=0}^{n-1} ||e^{(n-i-1)\Delta t L}|| \, ||\varphi_1(\Delta t L)|| \, ||E_i|| + \max_i ||\hat{E}_i|| \sum_{i=0}^{n-1} ||e^{i\Delta t L}||. \quad (1.27)$$

Bounds need to be found for the norms of the exponential functions in (1.27). Different approaches may be taken, and the particular form of the bound will depend on the approach and the norms used. We sketch standard ideas behind straightforward bounds in Remarks 1.5.2, 1.5.3 and 1.5.4. For a full discussion see [29].

Remark 1.5.2. Bound for $||e^{tL}||$

We assume there exists a positive C such that $||e^{tL}|| \leq C$. Possible justifications for this include the following.

1) We could assume that L is diagonalizable, then $e^{tL} = Ve^{tD}V^{-1}$, where D is the diagonal matrix of the eigenvalues of L. Then in the norm induced by the Euclidean 2-norm, $||e^{tL}||_2 \leq C_1 \max_i |e^{t\lambda_i}|$, where $||V||_2||V^{-1}||_2 \leq C_1$. If all the eigenvalues of L are negative, then $|e^{t\lambda_i}| \leq 1$. Otherwise $\max_i |e^{t\lambda_i}| \leq \max_i |e^{T\lambda_i}|$. Either way there exists a C_2 such that $||e^{tL}||_2 \leq C_2$. Equivalence of norms then guarantees a C such that $||e^{tL}|| \leq C$ in any norm.

2) Various approaches can be found in, for example, [60, §2] and the references therein. For example, consideration of the power series definition of e^{tL} leads to $||e^{tL}|| \leq e^{t||L||}$ which is bounded by a constant if $||L||$ is.

Remark 1.5.3. Bound for $||\varphi_1(\Delta t L)||$

We assume there exists a positive C such that $||\varphi_1(\Delta t L)|| \leq C$. Possible justifications for this include the following.

1) We could assume that L is diagonalizable, then $\varphi_1(\Delta t L) = V\varphi_1(\Delta t D)V^{-1}$, where D is the diagonal matrix of the eigenvalues of L. Then like in Remark 1.5.2, $||\varphi_1(\Delta t L)||_2 \leq C_1 \max_i |\varphi_1(\Delta t \lambda_i)|$. The term $\max_i |\varphi_1(\Delta t \lambda_i)|$ is bounded above in the range $\Delta t \in [0, T]$ (Δt is confined to this range because there will be between 0 and ∞ steps), because $\varphi_1(\cdot)$ is continuous. Thus $\varphi_1(\Delta t L)$ is bounded by a constant in the 2-norm, and thus is bounded by a constant in any norm by the equivalence of norms.

2) Another approach is to use the power series definition as in 1.5.2 2), leading to $||\varphi_1(\Delta t L)|| \leq \varphi_1(\Delta t ||L||)$, which is bounded by a constant if $||L||$ is.

Remark 1.5.4. Bound for $|| \sum_{i=0}^{n-1} e^{i\Delta t L}||$

Given Remark 1.5.2, there exists a positive C_4 such that $|| \sum_{i=0}^{n-1} e^{i\Delta t L}|| \leq \frac{C_4}{\Delta t}$.
We have that $|| \sum_{i=0}^{n-1} e^{i\Delta t L}|| \leq Cn = C\frac{t_n}{\Delta t} \leq \frac{CT}{\Delta t}$, where C is the bound on $||e^{tL}||$ from Remark 1.5.2.

Applying these bounds with (1.27) leads to the estimate,

$$E_n \leq C\Delta t \sum_{i=0}^{n-1} ||E_i|| + \max_i ||\hat{E}_i|| \frac{C}{\Delta t},$$

where we have replaced all products of various constants with the generic constant C. Taylor series expansion shows that the bound on the local error of ETD1 is

$$\max_i ||\hat{E}_i|| \leq \frac{\Delta t^2}{2} \max_t ||\frac{dF(u(t))}{dt}||,$$

so that by applying Lemma 1.5.1, we see that ETD1 is globally first order in Δt. Using similar methods, local errors of the globally second order methods ETD2 (1.18) and ROS2 (1.21) can both be shown to be third order in Δt using Taylor expansions.
As a general rule of thumb if the local error of a scheme is order p, then the global error is order $p-1$, given that adequate bounds on $\mathcal{L}$ and $\mathcal{N}(\cdot)$ exist, which depend on

assumptions on the operators that make up the scheme or the underlying PDE, such as boundedness of the matrix L or F being Lipschitz continuous. Heuristically, if the final time $T = N\Delta t$ is reached in N timesteps, then the local error is accumulated N times, leading to a term in the global error of the form $NO(\Delta t^p) = \frac{T}{\Delta t}O(\Delta t^p) = O(\Delta t^{p-1})$. In the two examples above, this describes the term in the global error inequalities which involves the local error $\hat{E}$ and takes the place of b in Lemma 1.5.1. The form of Lemma 1.5.1 shows us that b controls the bound on the global error. Ideally sharp estimates should be found for both the local and global errors for every scheme. In Chapters 3 and 4 we introduce new schemes and study their local errors. We then test the properties implied by the local error (i.e., accuracy and order of convergence) in numerical experiments. With the local error of these schemes examined, derivation of sharp bounds for the global error is a subject of potential future research.

1.5.2 Estimating Error in Numerical Experiments

When testing a scheme for accuracy and efficiency it is necessary to estimate the error. If the exact solution of the system is not available, this can be done by producing a comparison solve u_{comp} which is guaranteed to be close to the true solution. One can produce u_{comp} by using a scheme known to be reliable with a much smaller timestep Δt than the range used in the experiments, for example. Then u_{comp} is used instead of the true solution to estimate the error of solutions produced by the schemes being investigated. Consider a spatial discretisation with K total cells or nodes, such that the approximation u_n and the comparison solve u_{comp} satisfy $u_n, u_{\text{comp}} \in \mathbb{R}^K$. Then the following error estimation can be used.

$$\text{Estimated Error}(u_n) = \frac{||u_{\text{comp}} - u_n||_2}{\sqrt{K}}, \tag{1.28}$$

where $||\cdot||_2$ is the discrete euclidean norm. Equation (1.28) corresponds to an estimation of the Euclidean norm of the global error in (1.23). The scaling by $\sqrt{K}$ is to remove dependence of the error on the size of the system - without the scaling the error estimate would always increase with $\sqrt{K}$, due to the definition of the

Euclidean norm. We will use (1.28) for error estimation throughout the rest of this thesis. We will refer to (1.28) as the 'scaled Euclidean Norm' of the estimated error on occasion.

1.5.3 Stiff Order

We discuss here the concept of stiff order, which is a concept related to the benefits of exponential integrators. Our analysis is for classical order conditions. Prothero et al [61] are cited as discovering the phenomenon. They explained the concept in terms of so called S-stability and S-accuracy. The more well known terminology of B-convergence was later introduced by [62]. One may also see chapter IV.15 of [63]. Mention is also made in the review paper [64, §5], for example.

Consider an ODE, possibly an ODE system,

$$\frac{du(t)}{dt} = f(u(t),t), \tag{1.29}$$

where $u(t)$ and $f(u(t),t)$ may be vectors. Consider an approximation $u_n \approx u(t_n)$ by some scheme, and two different theoretical error results, in some norm,

$$||u_n - u(t_n)|| \leq Ch^p = E, \tag{1.30}$$

$$||u_n - u(t_n)|| \leq \bar{C}h^{\bar{p}} = \bar{E}, \tag{1.31}$$

where h is the timestep used, $C \neq \bar{C}$ and, crucially $p \neq \bar{p}$, that is, the error results are of different order. Both results may be theoretically true but the smaller of E, $\bar{E}$ determines the actual error behaviour.

As an example of having two different error order results, one can imagine working through the error analysis of a classical global second order method, but not considering the Taylor series to f beyond second order. One would obtain a bound on the local error of the form $\alpha_2 M_2 h^2$, even though a bound $\alpha_3 M_3 h^3$ is possible. Here the α_i are different constants and the M_i are bounds on the derivatives of f,

$$||\frac{d^i f}{dt^i}|| \leq M_i. \tag{1.32}$$

From the two different local error bounds one would then obtain global error results of first and second order. Using the notation of (1.30), (1.31), in this case let E be the first order result with $p = 1$ and $\bar{E}$ the second order result with $\bar{p} = 2$. The argument then goes that $\bar{E} \leq E$, because for low h, $h^{\bar{p}} < h^p$, and also possibly because f is a nice function whose derivatives cause $\bar{C} < C$. In this case the higher order result is more sharp.

Returning to the general case (1.30), (1.31), we can imagine an exception where $\bar{p} > p$ but $\bar{C} >> C$, so that $E \leq \bar{E}$ for any h we might practically use. In this case the lower order result is sharper and better describes the error behaviour. This can be the case when the function f is such that its higher derivatives have extremely high bounds in (1.32), resulting in $\bar{C} >> C$. Since this depends on f and thus the ODE (1.29) to which the scheme is applied, this is one explanation for the order reduction phenomenon in [61]. Another possibility is that the function f is such that a bound (1.32) simply does not exist. In that case the higher order error estimate is not just misleading, but untrue, when applied to certain systems (i.e., the assumption of the existence of an M_i was wrong in the analysis).

We can take this further towards understanding stiff order in exponential integrators, used extensively in the papers of Hochbruck and Osterman, e.g. [29]. First we specify that we are dealing with a semilinear form of (1.29), that is,

$$f(u(t), t) = Lu(t) + g(u(t), t),$$

where L is a matrix and g is the nonlinearity term. A stiff error analysis starts from an expansion of the true solution u,

$$u(t + h) = u(t) + \sum_{j=1}^{\infty} h^j \varphi_j(hL) g^{(j)}. \tag{1.33}$$

This is used to replace the classical Taylor series at all times. Also, Taylor expanding any of the $\varphi-$functions (see §1.4.1) is not allowed. Efforts are made in designing schemes to match terms in (1.33), not the classical Taylor series. Generally this causes the coefficients in error estimates to be controlled by terms like,

$$||\varphi_j(hL)\frac{d^i g}{dt^i}|| \leq \hat{M}_i, \tag{1.34}$$

that is, C and $\bar{C}$ vary in size with $\hat{M}_i$, and not M_i, for exponential integrators. Very often, $\hat{M}_i << M_i$, i.e. the φ-functions damp high derivatives of f (or that the derivatives of g are much smaller than those of f). Effects like this can be observed in Chapter 4 on 'semi-exponential' RK schemes, which are designed to satisfy classical order conditions, but exhibit an order barrier for stiff problems.

The result is that Hochbruck-type exponential integrators owe their stiff order to ensuring that their highest order error estimates remain sharp even when the scheme is applied to systems with very large derivatives of f, by having error estimate coefficients that depend on the (damping) action of a $\varphi-$function on such derivatives.

Chapter 2

Face Based Asynchronous Schemes

2.1 Introduction

We analyse and develop new schemes for the simulation of porous media flow based on an asynchronous simulation methodology. By asynchronous we mean a scheme in which different parts of the spatial domain are allowed to exist at different times simultaneously during the course of the simulation. Numerous different categories of schemes may fall under this broad description; here we are interested in schemes based on the Discrete Event Simulation (DES) methodology. This methodology is essentially the idea of evolving a system forward in time by discrete events, local in space, with each event having its own local timestep determined by the physical activity in that region, see [65, 1, 66]. In this way more active regions of the spatial domain receive more events, in principle leading to more efficient distribution of computational effort. A full description and algorithm follows in §2.2. Initially we focus on the simulation of linear conservation law models in the absence of reaction terms,

$$\frac{dc(\mathbf{x},t)}{dt} = \nabla f(c(\mathbf{x},t)), \qquad t \in \mathbb{R}, \qquad \mathbf{x} \in \Omega \subset \mathbb{R}^d, \tag{2.1}$$

$d = 1, 2, 3$, where $c(\mathbf{x}, t)$ is a concentration and f is a given flux function. An initial condition $c(\mathbf{x}, 0) = c_0(\mathbf{x})$ is provided. For simplicity of exposition, we consider 'no flow' boundary conditions, that is, Neumann type boundary conditions with zero flux on external faces. Other types of boundary conditions could easily be added in this framework. We are primarily interested in advection diffusion systems, where the flux is of the form

$$f((c(\mathbf{x},t)) = \nabla D(\mathbf{x})(c(\mathbf{x},t)) + \mathbf{v}(\mathbf{x})c(\mathbf{x},t), \tag{2.2}$$

and D is the diffusivity and $\mathbf{v}$ is a given velocity. The combination of (2.1) and (2.2) is the PDE

$$\frac{dc(\mathbf{x},t)}{dt} = \nabla^2(D(\mathbf{x})c(\mathbf{x},t)) + \nabla(\mathbf{v}(\mathbf{x})c(\mathbf{x},t)). \tag{2.3}$$

The diffusivity D may come from, for example, porosity and permeability in a porous media flow model and therefore could be a tensor, but often we consider a simple scalar field. The velocity $\mathbf{v}$ is of a vector field with the same dimension as Ω. In §2.6 we later describe the incorporation of a reaction term for our schemes. The spatial domain Ω is discretised into cells, as in a standard finite volume approach (see §1.3), and equation (2.1) is discretised in space over the grid. To describe our new schemes we start by focusing on a simple system; a conservation law without sources (2.1) with flux given by, for example, (2.2).

The outline of this chapter is as follows. In §2.1.1 we introduce the motivation and concept behind the schemes. In §2.1.2 we re-state the ideas from the Finite Volume discretisation method as they are relevant here. We present the basic version of the new schemes in §2.2, which we then compare to other schemes based on the same general methodology in §2.3. In §2.4 we describe the representation of the new schemes with so called 'connection matrices'. Modifications to the basic schemes are introduced in §2.5, and the addition of reaction terms is discussed in §2.6.

2.1.1 Motivation and Basic Concept

Our work builds on other work on the application of the asynchronous DES methodology to continuous systems [65, 1, 66]. We refer to our simplest scheme as the Basic Asynchronous Scheme (BAS). The underlying idea is simple but unusual, therefore we present the scheme now in a simplified way to elucidate the general concept. In this section we will present a motivation and explain the algorithm for BAS in a simplified way before providing a full description in §2.2

Consider a domain with varying flow rates in different spatial regions, such as the case with a strongly varying velocity or diffusivity field (see for example Figure 3.1 and the numerical experiments in Chapter 3). Assuming the domain is gridded for a finite volume discretisation, this will result in some faces in the grid having much greater flux than others. In Figure 2.1 we have a simplified example of such a grid. Figure 2.1 a) shows a section of a grid, with a value of mass in each cell. We consider the four cells shown and their four internal faces.

Figure 2.1 b) shows the fluxes on each of the internal faces. The face with the greatest flux is highlighted in red. The core of BAS is to identify the face with the greatest rate of activity, i.e. flux, and to perform an event where mass is transferred across *only this face*. Thus the two cells adjacent to the face (highlighted in plots b) and c) experience a change in mass, and we can think of the face evolving forward in time separately from the rest of the domain. The algorithm proceeds to evolve the domain forwards to a final time T by a sequence of these discrete events, in which parts of the domain evolve asynchronously, thus the terminology. Indeed, each face has its own individual time value which may be different from that of its neighbours.

Figure 2.1 c) represents a single event, where a specified amount of mass ΔM, which we refer to as the global mass unit, is allowed to pass over the face between the two cells according to the direction of the flux. The event is taken to represent activity over some interval of time Δt. This is calculated from ΔM and the flux on the face by the heuristic,

$$\frac{dm}{dt} = \text{flux}, \ \frac{dm}{dt} \approx \frac{\Delta M}{\Delta t} \implies \Delta t \approx \frac{\Delta M}{\text{flux}}. \tag{2.4}$$

A timestep Δt is calculated for every individual event and is thus specific to the face and the event. When a face undergoes an event, the Δt for the event is calculated and added to the face's individual time value. Figure 2.1 d) shows the grid after the event. Only the two cells adjacent to the active face have had their mass values changed.

The idea is to allow different parts of the domain to evolve at rates appropriate to their rates of activity (measured by flux). The arguments which motivate the method are:

1. A face with a greater flux should be allowed to evolve with more steps with smaller Δt values than a face with smaller flux, to maintain accuracy. The underlying assumption here is that the appropriate timescale is related to the flux. The greater the flux the faster the timescale; and thus a finer temporal resolution is necessary to correctly capture the local behaviour of the system. One way in which the local rate of activity effects temporal resolution is the CFL condition; this is discussed in relation to Asynchronous schemes in [67]. Another way to look at the argument is that in BAS the 'resolution' of the solve is set by ΔM - no single discrete transfer of mass in the system is allowed to exceed ΔM in magnitude, any by the heuristic (2.4), Δt is consequently inversely proportional to the flux.

2. If the faces are being processed one at a time, then faces with greater flux should have greater priority since the effects of evolving the system on that face is likely to be more significant than the effects of evolving the system on a face with smaller flux; and the greater flux face is going to need more events anyway.

This is the reasoning behind identifying the highest flux face, processing it first, and using Δt inversely proportional to the flux, in our example. More necessary details of the scheme, such as how to manage the fact that evolving one face will change the flux on nearby faces, how to correctly schedule events, and how to synchronise to a final time T, are all discussed in §2.2.

This example demonstrates the basic idea of the Asynchronous scheme. We have used a varying velocity field to demonstrate how the flux on the faces may vary, but this is not necessary; the algorithm works by looking at fluxes. Algorithm 1 describes the basic algorithm as outlined above; a more rigorous version is given in Algorithm 2. In Algorithm 1 we introduce the idea of faces having individual times.

```
while Any face has $t_k < T$ do
    Identify face $k$ with highest event priority (a function of the face's flux and individual time) ;
    Calculate $\Delta t$ for the event from (2.4) ;
    Transfer $\Delta M$ across face $k$, changing $m$ in the two adjacent cells ;
    Set face $k$'s individual time to $t_k + \Delta t$
    Update flux and priority of all faces which were affected by the changes in $m$ (including face $k$) ;
end
```

Algorithm 1: BAS - The basic idea of the scheme as outlined in §2.1.1. Some considerations such as synchronisation to a final time are ignored here, but are presented in Algorithm 2 and §2.2.

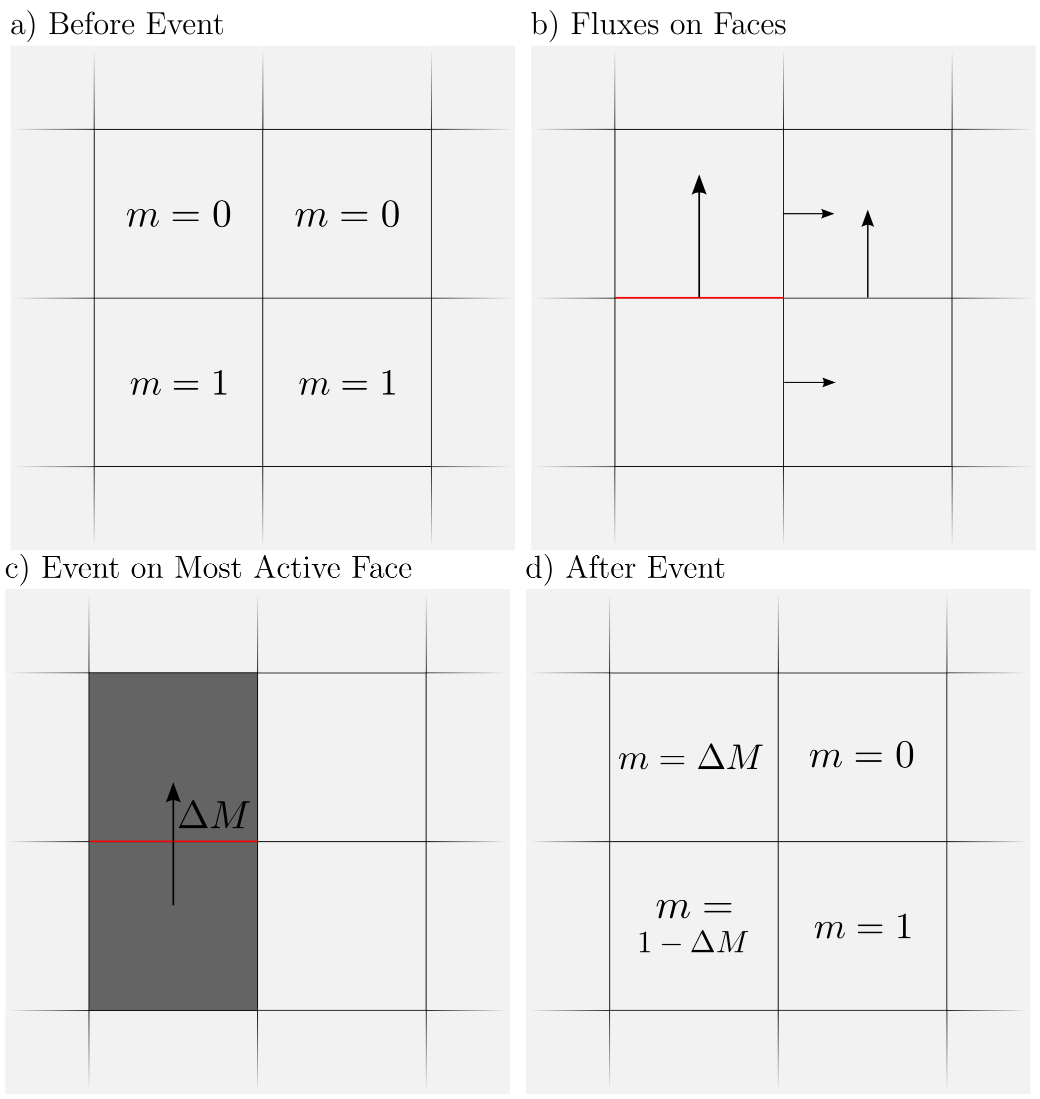


Figure 2.1: Demonstrating the idea of BAS, considering four cells and their internal faces. a) Distribution of mass in each cell. b) Fluxes across each face; the fluxes are not equal. The face with highest flux is highlighted in red. c) An asynchronous event moves the simulation forward. A pre-determined amount of mass ΔM is transferred between the two cells adjacent to the face with highest flux. d) Counterpart to a) after the event. The mass distribution in the system has changed but only in the two cells adjacent to the face which triggered the event.

2.1.2 Finite Volume Discretisation

Our schemes make use of the flux on each face of the grid approximated by the finite volume (FV) method. For clarity we restate the basics of the FV approach as they are relevant here; see also for example [12]. The spatial domain is divided to a grid of cells. Each cell has a unique index $j \in \{1, 2, \ldots, J\} = \mathcal{C}$, where we have named the set of all cell indexes $\mathcal{C}$ for convenience. Similarly every face also has a unique index $k \in \{1, 2, \ldots K\} = \mathcal{F}$. Some subsets of $\mathcal{C}$ and $\mathcal{F}$ are useful to define. For a cell with index $j \in \mathcal{C}$, define the set $\mathcal{F}_j$ of faces belonging to the cell, where $\mathcal{F}_j \subset \mathcal{F}$. Also, define the set of associated faces $\tilde{\mathcal{F}}_k$ of a face k as follows. If face k is adjacent to cells $j_1, j_2 \in \mathcal{C}$, then

$$\tilde{\mathcal{F}}_k = \mathcal{F}_{j_1} \cup \mathcal{F}_{j_2},$$

i.e., the associated faces is the set of all the faces of the two cells which face k is adjacent to. The finite volume discretisation of (2.1) is based on the approximation of the flux across faces in the grid. See Figure 2.2 for illustration of variable placement on the grid. Let f_k be the approximation of the flux on a face $k \in \mathcal{F}$, which depends upon the concentration values c_{j_1}, c_{j_2} in the two cells with indexes $j_1, j_2 \in \mathcal{C}$ adjacent to face k. The concentration c_j of a cell j is assumed constant throughout the cell, and is derived from the mass in the cell m_j and its volume V_j as $c_j = \frac{m_j}{V_j}$. The flux f_k on a face is assumed constant and defines the flow of mass across the face between its two adjacent cells, i.e., the flow of mass from cell j_1 due to face k will be $-f_k A_k$; and into cell j_2 will be be $f_k A_k$, where A_k is area of the face k. The direction of mass flow depends on the sign on f_k. To be explicit, the equations for mass flow *across a single face* k, are

$$\frac{dm_{j_1}}{dt} = f_k A_k, \; \frac{dm_{j_2}}{dt} = -f_k A_k. \tag{2.5}$$

The flux may be approximated by finite differences such as, for a diffusion only system,

$$f_k = \frac{\bar{D}_k(c_{j_2} - c_{j_1})}{\Delta x_k} = \frac{\bar{D}_k\left(\frac{m_{j_2}}{V_{j_2}} - \frac{m_{j_1}}{V_{j_1}}\right)}{\Delta x_k}, \tag{2.6}$$

where $\bar{D}_k$ is an approximation of the diffusivity at the face based on the diffusivity in the two cells, typically the harmonic mean of D_{j_1} and D_{j_2} (see §1.3), and Δx_k is the distance between the two cell centroids, for example, Figure 2.2 b) shows two cells; their centroids are marked with dots, and Δx_k is then the length of the straight line between them. For an advection-diffusion system, one of the two cells will be the upwind cell; without loss of generality let this be cell j_1. Then (2.6) will be replaced by

$$f_k = \frac{\bar{D}_k \left(\frac{m_{j_2}}{V_{j_2}} - \frac{m_{j_1}}{V_{j_1}} \right)}{\Delta x_k} - \frac{m_{j_1}}{V_{j_1}} v, \tag{2.7}$$

where v is the scalar product of the velocity at the centre of face k with the unit vector in the direction of the line from the centre of cell j_1 to cell j_2. We now take a moment to discuss the signs in (2.7) and their meanings in (2.5). First, since j_1 is the upwind cell, it is clear that $\frac{m_{j_1}}{V_{j_1}} v$ should have a negative sign since mass will leave cell j_1 and arrive in cell j_2. The diffusion term in (2.7) derives from Fick's law of diffusion, according to which the rate of change of the amount of mass at the centre of cell j_1, along the line connecting the two cells, is

$$\frac{dm_{j_1}}{dt} = -D \frac{dc_{j_1}}{dx}, \tag{2.8}$$

and respectively for cell j_2. Here D is the diffusivity, and the spatial derivative is along the line connecting the two cells. While the D replaced with an average $\bar{D}$ as stated earlier, the spatial derivative is approximated by a finite difference, $\frac{dc_{j_1}}{dx} \approx \frac{c_{j_1} - c_{j_2}}{\Delta x_k}$. The negative sign in (2.8) flips the order of the difference, which is the reason for the order of the difference in the diffusion term in (2.7). We see that the same applies for cell j_2.

The total rate of change of mass, and thus concentration in a cell j is the sum of (2.5) for each $k \in \mathcal{F}_j$. This can be expressed as a matrix, L which gives the finite volume semidiscretisation of (2.1) as a system of ODEs,

$$\frac{d\mathbf{c}}{dt} = L\mathbf{c}, \qquad L \in \mathbb{R}^{J \times J} \tag{2.9}$$

where $\mathbf{c} = (c_1, c_2, \ldots, c_J)^T$ is the vector of concentrations in cells. In a standard finite volume based implementation (2.9) is then discretised in time, resulting in the fully discrete approximation. Face based asynchronous schemes are based on events involving the transfer of mass across a single face but do not form the global system (2.9); instead they can be defined in terms of much smaller local matrices which we call "connection matrices" and introduce in §2.4. They proceed in discrete events approximating the effect of (2.5). We now present the basic asynchronous scheme in detail.

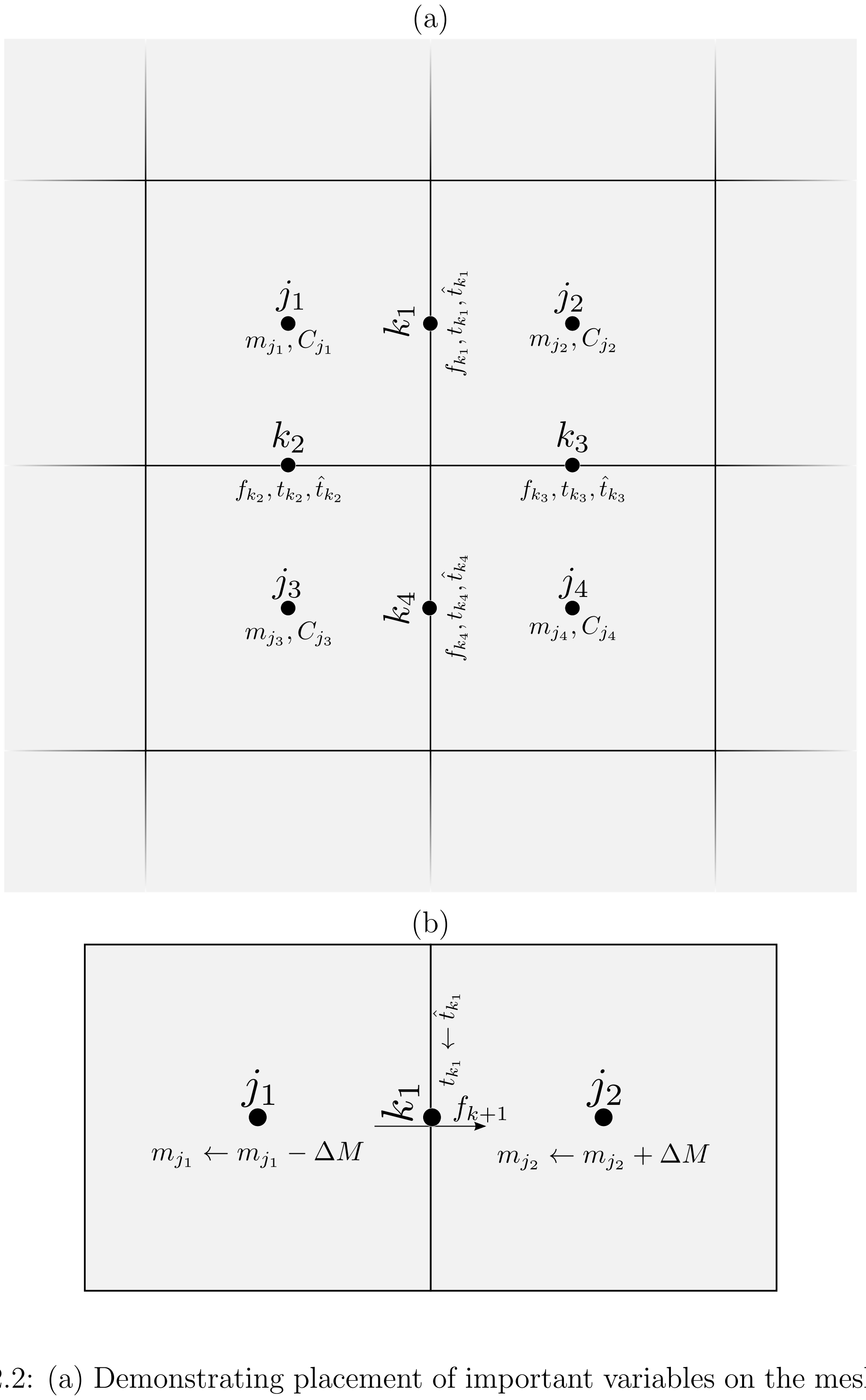

(a)
j_1
m_{j_1}, C_{j_1}
k_1
$f_{k_1}, t_{k_1}, \hat{t}_{k_1}$
j_2
m_{j_2}, C_{j_2}
k_2
$f_{k_2}, t_{k_2}, \hat{t}_{k_2}$
k_3
$f_{k_3}, t_{k_3}, \hat{t}_{k_3}$
j_3
m_{j_3}, C_{j_3}
k_4
$f_{k_4}, t_{k_4}, \hat{t}_{k_4}$
j_4
m_{j_4}, C_{j_4}
(b)
j_1
$m_{j_1} \leftarrow m_{j_1} - \Delta M$
k_1
$t_{k_1} \leftarrow \hat{t}_{k_1}$
f_{k+1}
j_2
$m_{j_2} \leftarrow m_{j_2} + \Delta M$

Figure 2.2: (a) Demonstrating placement of important variables on the mesh. Each cell (labelled j_1, j_2 etc.) has a mass, m, and thus a concentration C value. Each face (labelled k_1, k_2 etc.) has a flux, f, time, t and update time $\hat{t}$ value. (b) During an event on face k_1, only the cells adjacent to the face (here j_1 and j_2) are updated. An amount of mass ΔM is transferred between the two cells, and the time on the face t_{k_1} is set to be the update time $\hat{t}_{k_1}$. The update time and ΔM are related to the flux f_{k_1} by (2.10)

2.2 The Basic Face Based Asynchronous Scheme (BAS)

We now describe in depth the simplest form of our face based asynchronous schemes (BAS). We introduce the important parameters and variables used in the method. Consider the spatial domain Ω discretised into cells as in the standard finite volume approach. We use a subscript to denote association of a variable with a face or cell. For example, $\hat{t}_k$ is the update time for face k. Figure 2.2 (a) illustrates the placement of the variables. The variables belonging to each a cell j are:

Variables belonging to a cell j

Symbol	Description
m_j	Mass in cell j.
V_j	Volume in cell j.
c_j	Concentration; $c_j = \frac{m_j}{V_j}$.
D_j	Effective diffusivity in cell j.
$\mathbf{v}_j$	Velocity vector in cell j.

The variables associated with each face with index k are:

Variables belonging to a face k

Symbol	Description
A_k	Area of the face k.
f_k	Flux across the face k; see (2.6).
t_k	Time of the face's most recent update.
$\hat{t}_k$	Projected update time of the face.
Δt_k	The timestep with which the face will asynchronously advance during its next event.

Global values, not placed on specific parts of the discretisation of Ω are:

Global values

Symbol	Description
ΔM	Global mass unit. This is the amount of mass to be transferred between two adjacent cells in an event.
t	Global time; the lowest time on any face, i.e. $t = \min_k t_k$.
T	Final time. The algorithm will reduce the update times of faces towards the end of the solve in order to synchronise all faces to having time $t_k = T$.

We note that choosing an appropriate value of the global mass unit ΔM to balance between accuracy and efficiency is of great importance in using this method. For a face k, the projected update time $\hat{t}_k$ is calculated so that in the interval $\Delta t_k \equiv \hat{t}_k - t_k$, at most ΔM units of mass pass through the face. We discuss the relationship between $\hat{t}_k$ and Δt_k in more detail now.

The BAS method calculates the update time $\hat{t}_k$ as,

$$\hat{t}_k = \begin{cases} t_k + \frac{\Delta M}{|f_k| A_k} \text{ if this} \leq T \\ T \text{ otherwise.} \end{cases} \tag{2.10}$$

This is derived as follows. Consider a simple Euler-type approximation of the flow of flux through the face, ignoring the effect of the other faces in the cell. We want a face to have passed an amount of mass ΔM in the time interval $\hat{t}_k - t_k$. This leads to an approximation of the derivative in (2.5),

$$\text{Mass flow through single face } k \approx \frac{\Delta M}{\hat{t}_k - t_k} = |f_k| A_k, \tag{2.11}$$

from which (2.10) follows. The absolute value of the flux is used to ensure that the calculated time values are positive. The aim is to approximate the time taken for an amount of mass ΔM to cross the face k. The direction is irrelevant, thus the only magnitude of the flux is important.

When (2.10) calculates $\hat{t} > T$, the value of T is used instead. In this way the simulation finishes with every face at the desired final time T; it is an Euler-type approximation using the imposed timestep $T - t_k$. The mass transferred during this final synchronisation step is not ΔM. Let δm be the mass transferred in an event for face k. Then,

$$\delta m = \begin{cases} \Delta M \text{ if } t^k + \frac{\Delta M}{|f_k| A_k} \leq T \\ \\ |f_k|(T - t_k)A_k \text{ otherwise.} \end{cases} \tag{2.12}$$

```
   Data: Grid structure, Initial concentration values, $\Delta M$, $T$
1  Initialise: $t = 0$ ; Calculate $f_l$ from (2.6) and $\hat{t}_l$ from (2.10) $\forall$ faces $k$ ;
2  while $t \leq T$ do
3      Find face $k$ s.t. $\hat{t}_k = \min_{l \in \mathcal{F}} \hat{t}_l$ ;
4      Get cells $j_1$ and $j_2$ adjacent to $k$;
5      Calculate $\delta m$ from (2.12) ;
6      $m_{j_1} \leftarrow m_{j_1} - \text{sign}(f_k)\delta m$ ;
7      $m_{j_2} \leftarrow m_{j_2} + \text{sign}(f_k)\delta m$ ;
8      $t = t_k \leftarrow \hat{t}_k$ ;
9      for $l \in \tilde{\mathcal{F}}_k$ do
10         Recalculate $f_l$ from (2.6) ;
11         Recalculate $\hat{t}_l$ from (2.10) ;
12     end
13 end
```

Algorithm 2: Pseudo code for the basic face based scheme.

Algorithm 2 describes the BAS method. After initialising the required values on all faces, the update loop is run until every face is synchronised to the desired final time of T. Each iteration of the loop is a single event and proceeds as follows. First the face with the lowest projected update time $\hat{t}$ is found (line 3) Then the two cells adjacent to this face are located from the grid structure (line 4). The amount of mass to transfer between these cells is calculated (line 5). This equation simply returns the global mass unit ΔM in most cases, except when the face is being forced to use an update time T; see equation (2.10). Mass is transferred between the cells in the correct direction (lines 6-7). A loop (lines 8-12) updates the faces of cells j_1 and j_2; recalculating their fluxes and update times based on the new mass values. The loop then continues by finding the next face with the lowest uptime (back to line 3). Figure 2.2 (b) illustrates an event occurring on a face k.

This is the simplest face based Asynchronous scheme we can conceive. We have observed, for every experiment we have attempted, that, as the mass unit ΔM

decreases to zero, the approximation produced by this scheme converges to the exact solution of the linear ODE system produced by applying the corresponding finite volume discretisation to the corresponding PDE (2.1). Thus, even this simple Asynchronous scheme works and will agree with established classical schemes for sufficiently small ΔM. We illustrate the scheme with a simple example here; further numerical experiments can be found in Chapter 3.

In Figure 2.3 we show the results of BAS applied to (2.3). The domain is $\Omega = 10 \times 10 \times 1$ metres. There is no advection, so $\mathbf{v}(\mathbf{x}) = (0,0,0)^T \ \forall \mathbf{x} \in \Omega$; there is a constant diffusivity field of value $D(\mathbf{x}) = 1.0 \ \forall \mathbf{x} \in \Omega$, there are no-flow boundary conditions, and the initial condition is concentration zero except for a single cell of concentration 1.0 in the centre of the domain, that is $c(\mathbf{x}, 0) = 1.0$ if $\mathbf{x} = (4.95, 5.05, 0.5)^T$ and $c(\mathbf{x}, 0) = 0.0$ otherwise. The final time is $T = 1.0s$, the domain Ω divided into $100 \times 100 \times 1$ cells, making the system effectively two dimensional.

Figure 2.3 a) shows the accurate result produced by BAS with $\Delta M = 10^{-10}$. Figure 2.3 b) shows how the error (i.e., the difference between the BAS solve result and the comparison solve), decreases with ΔM, in the scaled Euclidean norm given by (1.28). Figure 2.3 c) shows the result from a BAS solve with ΔM too large, which will be discussed shortly. Finally, Figure 2.3 d) shows the comparison solve, produced by an exponential integrator, to which BAS agrees excellently when ΔM is sufficiently small - compare a) and d). For more details about how our numerical experiments were performed, see §3.1.

It is strongly indicated by Figure 2.3 a) that the mass unit ΔM acts analogously to the tolerance for adaptive timestepping schemes or the timestep for fixed timestepping schemes. Decreasing ΔM improves accuracy of the scheme at the cost of greater computational expense (i.e., more events). Also, analogously to how a fixed timestepping scheme may exhibit instability as the timestep becomes too large, the basic scheme presented here exhibits certain phenomena when ΔM is too large. These can be seen in Figure 2.3 c), and we discuss them now.

First, as can be inferred from the algorithm, there is no guarantee that that the mass taken from a cell during one event will always be less than the mass currently in the cell - thus a cell can be overdrawn, i.e., that the mass and thus concentration in a cell can become negative. We observe that this does not happen in practice for ΔM sufficiently small, but does for ΔM unsuitably large. Indeed, we can observe from the scale in Figure 2.3 c) that some cells have negative concentration. This is the *overdrawing* phenomenon (after the term used for a similar effect in [68]) at large ΔM. One of the modifications to BAS described in §2.5 eliminates overdrawing.

The second phenomenon is that the scheme fails to preserve the smoothness of a solution, when ΔM is too large. Observe the 'checkerboard' pattern of Figure 2.3 c), where adjacent cells have alternating shades or colours, caused by small scale un-physical oscillations caused by the scheme. As before, this does not occur for ΔM sufficiently small. We refer to this as the '*checkerboarding*' phenomenon.

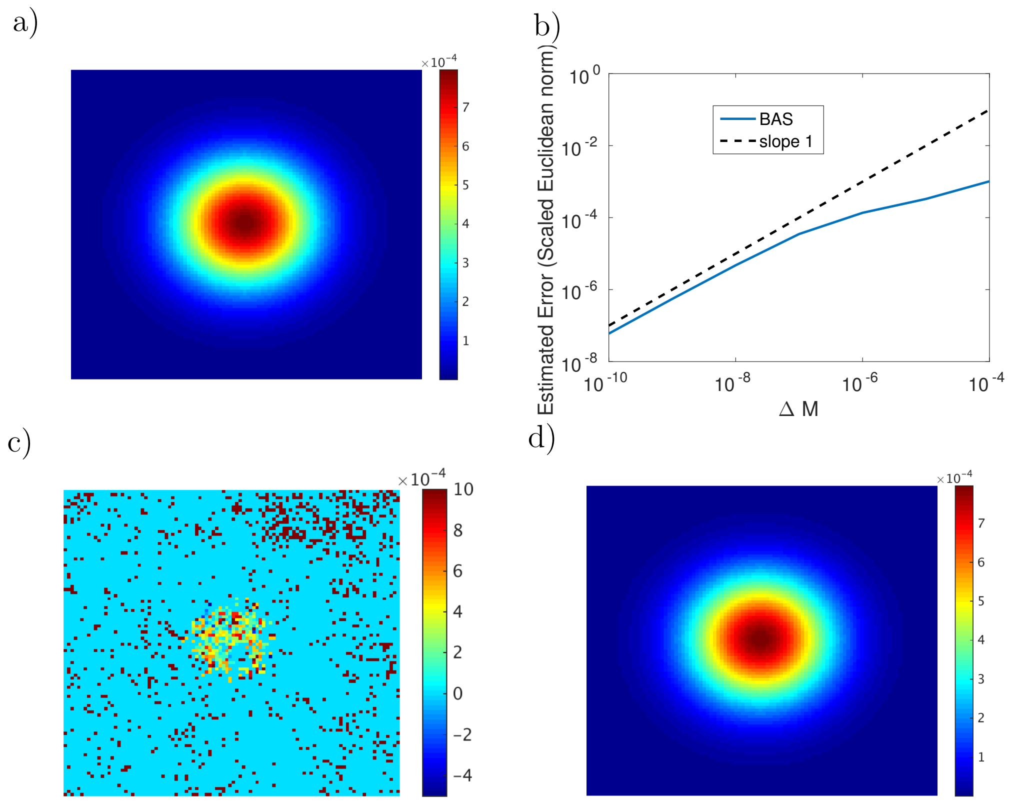


Figure 2.3: Simple example of BAS agreeing with classical solution. a) BAS solution with $\Delta M = 10^{-10}$. b) Error vs. ΔM for BAS, showing convergence. c) BAS solution with ΔM too large (10^{-5}). d) Comparison solve produced by exponential integrator.

2.3 Background and DES Schemes in General

With the concept explained through the in depth presentation of our BAS, we present some background. Traditionally Discrete Event Simulation (DES) schemes were developed for naturally discrete systems, not continuous physical systems such as fluid or solute flow models. The use of DES concepts applied to continuous physical systems was introduced by [65] for plasma simulation; extending this work has been the main pursuit of these same authors, including a parallel implementation in [69], and further development in [70]. In [65], the underlying physics was simulated directly - an event was the motion of an ion particle between two cells. The same authors then presented in [1] an asynchronous method for conservation law PDES with sources, in one dimension, based on evolving the PDE model at different rates in different cells. Our methods are all, by contrast, face based, in that an event is always the transfer of mass between cells, not evolution within a cell.

Indeed, the choice of which elements of the spatial domain that are allowed to exist at different times in an Asynchronous scheme leads to different types of scheme. In our schemes every face k of the finite volume discretisation has a time t_k. In [1] every cell has a time; in [65] individual particles have individual times. In [66], inter-cell fluxes and source terms experience asynchronous events, while cells themselves posses individual times.

Now we describe in more detail the general idea of DES based asynchronous schemes. The simulation is moved forward by discrete events on a single part of the domain, one at a time. Every event has a timestep Δt associated with it. We take an event to be the transfer of some amount of mass across a face k between the two adjacent cells (at a rate determined by the flux in (2.1), for example (2.2)). In [1] an event is the update of the local PDE in a cell; in [65], an event is a particle's position being updated according to its current trajectory, and in [66] an event models the effect of either a flux term on a single face, updating the two cells adjacent to the face, or the effect of a source term in a single cell, updating only that cell.

The faces/cells/particles/etc are systematically given a priority. Each object with a time t_k also has an update time $\hat{t_k}$. At any instant in the simulation the object with the lowest update time will have the next event. After an event the update time for the object, and typically some of its neighbours, must be updated. The update time for object k takes the form

$$\hat{t_k} = t_k + \Delta t_k,$$

where Δt_k is the timestep estimated for the next event object k has. Two factors affect Δt_k. First, the local rate of activity in the system at object k. For a face, this is the flux f_k; for a cell it may be the rate of the local PDE (i.e. the combined effect of all the fluxes of the cell), for a particle, the velocity. The update timestep Δt_k should decrease as the instantaneous rate of activity increases. The combined effect of the above is that a domain part (face, cell, particle or so on) will experience events with shorter timesteps and therefore more events when it is more active. The second factor is some user controlled measure of how much change should be allowed to take place in an event. In our work we take this to be a discrete unit of mass ΔM (the 'mass unit') which is expected to travel through any face in an event. The timestep Δt_k should decrease as ΔM decreases, leading to convergence.
The assumption implicit here is that, in order to obtain some global level of accuracy, a timestep must be inversely proportional to the local rate of activity. Asynchronous schemes self-adaptively use smaller timesteps and more events where and when local rates of activity are greater. In this way they are intended to efficiently balance computational effort (many small Δt events being more expensive than few large Δt ones).
The face-based Asynchronous scheme presented here can be applied in one, two or three spatial dimensions, and can be thought of as either simulating PDEs like (2.1), using the form of the flux term such as (2.2), or conversely by starting with the physical flux (2.2) at cell interfaces, resulting in a system with behaviour described by a PDE (2.2). The mass unit ΔM is here a global value, i.e., it is taken to be the same across the whole domain Ω. This parameter can be used to control the accuracy or computational expense of the scheme, analogous to the fixed timestep length in fixed time step scheme or the tolerance in an adaptive timestepping scheme. In our

analysis, §4.1, and experiments, §3.1, we are particularly interested in showing that the error of our asynchronous schemes converges to zero as $\Delta M \to 0$ (when compared to the exact solution of the corresponding finite volume discretisation of the same system). There is discussion in [1] on the possibility of locally varying ΔM (or the analogue to it used there). In that case analogies with adaptive timestepping methods become more apparent.

Let us discuss sources of error. Asynchronous schemes have an error due to having neighbouring cells and faces with different local times. For example, in the standard FV discretisation a flux f_k on face k may be an approximation from a finite difference formula such as (2.6). This has an error which decreases as the grid becomes finer, i.e., as the dimensions of the cell $\Delta x \to 0$. In addition, in an asynchronous scheme the concentrations in cells j_1 and j_2 may be at different times, adding another source of error. A reasonable assumption is that this error goes to zero as $\Delta M \to 0$.

While traditional PDE solvers rely on efficient linear algebra solvers, and exponential integrators rely on efficient approximation of the matrix exponential, asynchronous schemes rely on an efficient way of ordering the pending events. The list of pending events is typically stored in a binary tree or custom priority queue, adding some additional complexity to the implementation. A custom type of priority queue is described in [1], we use our own implementation of this description here (in detail The Appendix §9.1.). The same priority queue design can be found in [71], where it is a component of the DES type simulation of a chemical reaction system there.

2.3.1 Comparison of BAS With Other Schemes

We can compare this scheme with existing schemes of the same class. The scheme of [1] is also designed for conservation law systems and is cell based, with a governing PDE divided onto individual cells, and each cell experiencing asynchronous updates. This scheme was only tested for one dimensional systems, and is equipped with some innovative (though complicating) features which aim to improve its efficiency (see §2.5.3). In contrast, our scheme has faces as the event experiencing elements, and is based on interface fluxes instead of the governing PDE. We have attempted to design here the *simplest* scheme of this class we can conceive of, for simplicity of presentation of the DES concept applied to continuous systems, and to investigate the efficacy and properties of such a scheme.

The scheme presented in [66] can be called a face based scheme using our terminology. Again designed to be applied to conservation law systems, this scheme has flux terms (i.e., faces) *and* source terms as the elements which experience asynchronous events, though only cells possess times values - an event on a face will update the two adjacent cells with different timesteps according to their individual times and the update time. Interestingly, this scheme does not make use of a ΔM or equivalent - update times are always calculated according to a Courant–Friedrichs–Lewy (CFL) criterion. This precludes what is perhaps the most interesting observation of this current work - that even the most basic face based Asynchronous scheme, and modifications thereof, seems to converge with first order to the exact solution as the control parameter, ΔM or equivalent, decreases to zero, in dimensions one two or three.

The goal of asynchronous schemes may be compared with, for example, adaptive time stepping [72] schemes, where the local error after each timestep is estimated, and the step repeated with a smaller Δt if this is found too large, or local timestepping [73, 68, 74] schemes (LTS), where the spatial grid is refined in space in order to better capture more active regions, and a corresponding local timestep is used to ensure a local CFL condition. Local timestepping schemes also exist where the grid is not refined spatially and the local timesteps are varied to better capture activity according to local rates. See for example [75], where a binary tree is used to schedule the order in which cells will update (similar to the methodology presented here, as will be seen), but full asynchronicity is avoided (unlike here) by implementing a standard LTS interpolation procedure between adjacent cells at different times, when approximating spatial derivatives.

There is also some similarity here with the modelling philosophy involved with the reaction diffusion master equation (RDME) [76]. With the RDME, space is divided into a mesh, and diffusion is modelled as a random walk of molecules or particles along this grid. Reactions occur, also stochastically, between molecules within the same cell according to a given rate. The RDME itself is a differential equation expressing the rate of change of a conditional probability, the probability of there being a certain number of molecules of a certain species present at given time. The typical simulation method associated with this modelling philosophy is the Gillespie method [77] and derivatives, see [78, 71] for example. According to the terminology we are using, the Gillespie method can fairly be described as a DES method. The heart of this method is the (stochastic) calculation of the next time when a chemical reaction or particle motion will occur, followed by an update of the system at that time. A different expected time is generated for each reaction and the reaction with the *lowest* expected time is executed. An analogy to this can be drawn with the schemes to be presented here, especially with focus on the expected update time $\hat{t}_k$ (see §2.2). We have already mentioned [71] as using a similar priority queue structure; there a *dependency graph* is used to determine if the expected time for a reaction needs to be updated after a system update. This is similar to the associated face concept and the set $\tilde{\mathcal{F}}_k$ used here, which determines which faces need to have their fluxes and thus update times re-calculated after an event. In general Gillespie-type algorithms are based on discrete events for simulating discrete systems (systems of particles). The schemes here differ from the Gillespie method and its variants in the following ways. Firstly the scale of interest is different, between the scale of counting individual molecules and the scale at which Fick's law of diffusion or bulk moment due to advection is expected to be appropriate. Secondly the schemes here are deterministic. Stochastic effects can be incorporated by either using, for example, random diffusivity fields (§3.1.4), or potentially by adding a stochastic component to the computation of update times (this would be a further research topic). (Also, we note that a stochastic component would have a different meaning than it does in a particle simulation. The random motion of particles is supposed to be subsumed into the deterministic Fickian diffusion equation. Adding a stochastic component to this would model something like inhomogeneities in the diffusivity field at small scale, or represent some correction to the upscaling theory that Fickian diffusion derives from). Thirdly the schemes here are unambiguously asynchronous, with adjacent cells being allowed to exist at different times simultaneously, and the asynchronous data from these cells being combined in spatial derivative approximations (and this turns out to not be pathological). Gillespie-type schemes are based on discrete events, but they model discrete systems.

An interesting comparison with the Gillespie method is as follows. The Gillespie method does not approximate the RDME (as is often intractable) directly, but is instead based on underlying concepts (reaction propensity from which expected reaction times are derived, and so on), and simulates the solution to an RDME without making use of it in the implementation [77, §1]. The face-based Asynchronous schemes presented here analogously to this, and to [65], simulate the solution to the (finite volume discretisation of) (2.1) without referencing the PDE itself in implementation.

By advancing different regions at different rates, Asynchronous schemes like BAS can be viewed as mixing the temporal and spatial components of the solution. Other methodologies exist which do this much more explicitly. For example, Space-Time Discontinuous Galerkin methods (see e.g. [79]) mesh and discretise the space-time domain, as opposed to just the space domain, in a way similar to the finite volume discretisation method. For example, if the spatial domain is two dimensional, then the mesh will be generated over the three dimensional space-time domain, and the governing equation discretised over this mesh. Moreover, the dependency of space-time elements on each other can be analysed in terms of their dihedral angles. See for example, [80]. It is thus possible to identify space-time elements as independent and solve them in parallel. Space-Time DG schemes do not naturally attempt to prioritise high activity areas of the domain for computational attention like BAS or related schemes. The theoretical understanding of spatial-temporal dependence that can be utilised in space-time DG schemes would clearly be a huge advantage for the development of more sophisticated schemes in the same class as BAS. Potential research attempting to apply the discrete event simulation methodology to a space time DG mesh may be very promising.

BAS and related schemes order local update events according to activity. This can be compared with the cascading method of Appleyard and Cheshire [81], and methods related to and derived from this, for example [82]. Note that we use the term ‘cascading’ to describe a type of scheme below; an explicit connection with [81] is not implied. The cascading method of Appleyard and Cheshire starts with the spatial domain discretised in a standard way, and a standard implicit temporal discretisation, resulting in a nonlinear algebraic system of equations to be solved for each timestep update. The standard approach is to use the Newton iteration or some modification. In [81] the Newton algorithm is augmented with a cascading sweep though the cells of the spatial grid, producing an approximation to the solution variable cell by cell, treating each cell as an independent system. This is similar to BAS treating each face and adjacent pair of cells as an independent system during an event. The algorithm of [81] is not asynchronous and does not incur an asynchronicity error as a result of its independent cell solves, because the solves are ordered according to the velocity field, such that the cells are solved in order of dependency (this is not always possible, in which case the process must iterate). The cascading scheme of [81] does not attempt to focus computational work in regions and times of significant activity like BAS and related schemes.

We may also compare BAS and related schemes with streamline methods, see [83] and, for example, [84, 85]. In a streamline method the velocity field is calculated on a standard spatial mesh, after which the streamlines for the velocity field are found. The solution is updated by solving along the streamlines. Solving along a streamline is a simpler problem than solving over the whole grid - there is thus a similarity to BAS solving a succession of simple one-face systems or schemes like those in [1] solving one-systems. Consideration of the underlying physics is used in streamline methods to allow a simpler approximation technique, in BAS it is used to identify a concept that can be used to order events (i.e., the flux). Streamline methods have no inherent asynchronicity by design.

2.4 Connection Matrices

We introduce here the concept of connection matrices, which we can then use for designing improvements in §2.5.1 and in our analysis §4.1. One event in the scheme of Algorithm 2 is the transfer of mass across the active face k, between the two cells j_1 and j_2 adjacent to k. In effect, during the event, the face k and the two cells are being considered as an independent system from the rest of the domain, i.e., like in Figure 2.2 (b). The only free variables are the masses m_{j_1} and m_{j_2} in the two cells. Thus, with the finite volume discretisation of the flux in place, the local flow of mass across the face may be considered as a 2×2 ODE system. Consider (2.9) for only two cells, after multiplying out each cell equation by the volume V_j, we have

$$\frac{d}{dt}\begin{pmatrix} m_{j_1} \\ m_{j_2} \end{pmatrix} = \begin{pmatrix} -a_k & b_k \\ a_k & -b_k \end{pmatrix}\begin{pmatrix} m_{j_1} \\ m_{j_2} \end{pmatrix} = \begin{pmatrix} A_k f_k \\ -A_k f_k \end{pmatrix}. \tag{2.13}$$

The non-negative scalars a_k, b_k are functions of the diffusivity D_j and velocity v_j of the two cells, the distance between their centres, and the area of the face k. Recalling equations (2.5), (2.6) and (2.7), we can see that, if j_1 is the upwind cell, then a and b are,

$$a_k = \bar{D}_k \frac{1}{V_{j_1}\Delta x_k} + v,\ b_k = \bar{D}_k \frac{1}{V_{j_2}\Delta x_k},$$

or, if j_2 is the upwind cell,

$$a_k = \bar{D}_k \frac{1}{V_{j_1}\Delta x_k},\ b_k = \bar{D}_k \frac{1}{V_{j_2}\Delta x_k} + v,$$

where v is the scalar product of the velocity at the centre of the face, with the unit vector in the direction of the line connecting the centres of the two cells, pointing from the upwind into the downwind cell. Thus we see that a_k and b_k are indeed non-negative, since $\bar{D}_k$ and v are both non-negative.

The matrix in (2.13) is an example of what we henceforth refer to as a local *connection matrix* $\tilde{L}_k$. The corresponding global connection matrix L_k is the sparse matrix with nonzero elements at (j_1, j_1), (j_1, j_2), (j_2, j_1) and (j_2, j_2);

$$L_k \equiv \begin{pmatrix} & & \\ & -a_k & b_k & \\ & & & \\ & a_k & -b_k & \\ & & & \end{pmatrix} \in \mathbb{R}^{J \times J}. \tag{2.14}$$

The structure of the connection matrix reflects the conservation of mass between the two adjacent cells (since the column sum is zero). The connection matrix L_k associated with face k describes the relationship between the two cells j_1 and j_2 adjacent to face k in the discretisation (2.9), and thus has nonzero entries only in columns and rows j_1 and j_2.

Let $\mathbf{m}$ be the vector of all mass values in the system and $\mathbf{c}$ the vector of all concentration values in the system, related by $\mathbf{c} = \mathbf{m}\mathbf{V}$, where $\mathbf{V}$ is the diagonal matrix with entries $\frac{1}{V_j}$, i.e., the inverse of the volume in each cell. The global ODE system for $\mathbf{m}$ can be accumulated from the connection global matrices on each face, that is,

$$\frac{d\mathbf{m}}{dt} = \sum_{k \in \mathcal{F}} L_k \mathbf{m}.$$

Right multiplying by $\mathbf{V}$ gives

$$\frac{d\mathbf{c}}{dt} = \sum_{k \in \mathcal{F}} L_k \mathbf{c},$$

and we see that the system discretisation matrix L in (2.9) is accumulated from the connection global matrices on every face, that is,

$$L = \sum_{k \in \mathcal{F}} L_k. \tag{2.15}$$

Connection matrices are useful for re-expressing our schemes. Consider the local description of an event across face k with adjacent cells j_1, j_2. Lines $5 - 7$ in Algorithm 2 describe an update that is equivalent to an Euler type step for solving

(2.13), i.e.,

$$\begin{pmatrix} m_{j_1} \\ m_{j_2} \end{pmatrix} \leftarrow (I + \Delta t_k \tilde{L}_k) \begin{pmatrix} m_{j_1} \\ m_{j_2} \end{pmatrix}, \tag{2.16}$$

where I is the identity matrix. Alternatively, using the $J \times J$ connection matrix L_k, then we can express event updates in terms of the entire system. The full system version of (2.16) is

$$\mathbf{m} \leftarrow (I + \Delta t_k L_k)\mathbf{m}. \tag{2.17}$$

Due to the sparsity of L_k, clearly only the cells j_1, j_2 are affected by (2.17) even though the equation describes the entire system. Writing an event like this will be used later in our analysis - see §4.5.

We now describe properties of global connection matrices. A connection matrix acting on any vector produces a vector pointing in only one direction in the solution space. That is, the action of a connection matrix L_k on any vector $\mathbf{x}$ is a scalar multiple of a vector $\hat{\mathbf{z}}_k$, determined by L_k. Consider a connection matrix L_k with non-empty columns and rows j_1, j_2, then

$$L_k\mathbf{x} = (b_k x_{j_2} - a_k x_{j_1})\hat{\mathbf{z}}_k, \tag{2.18}$$

where $\hat{\mathbf{z}}_k = (0, \dots, 0, 1, 0, \dots, 0, -1, 0, \dots, 0)^T$, where the non-zero entries are at j_1 and j_2. It follows that $\hat{\mathbf{z}}_k$ is an eigenvector of L_k and the corresponding eigenvalue can be found,

$$L_k\hat{\mathbf{z}}_k = \lambda_k\hat{\mathbf{z}}_k \qquad \lambda_k = -(a_k + b_k), \tag{2.19}$$

thus the eigenvalue λ_k is negative. From this we can prove the following Lemma.

Lemma 2.4.1. *Let L_k be a connection matrix corresponding to face k, which has adjacent cells j_1, j_2. Let $\hat{\mathbf{z}}_k$ be the eigenvector of L_k be with eigenvalue λ_k, according to* (2.19)*. Then, for any scalar s and vector $\mathbf{x}$,*

$$(e^{sL_k} - I)\mathbf{x} = s(b_k x_{j_2} - a_k x_{j_1})\varphi_1(-s(a_k + b_k))\hat{\mathbf{z}}_k, \tag{2.20}$$

where the $\varphi-$function $\varphi_1(\cdot)$ is as given by (1.14) *in §1.4.1.*

Proof. We have that

$$(e^{sL_k} - I)\mathbf{x} = \sum_{i=1}^{\infty} \frac{(sL_k)^i}{i!}\mathbf{x} = s(b_k x_{j_2} - a_k x_{j_1}) \sum_{i=1}^{\infty} \frac{(sL_k)^{i-1}}{i!}\hat{\mathbf{z}}_k.$$

Since $\hat{\mathbf{z}}_k$ is an eigenvalue of L_k, the sum becomes a scalar sum of powers of the eigenvalue λ_k, and we can shift the index to get

$$(e^{sL_k} - I)\mathbf{x} = s(b_k x_{j_2} - a_k x_{j_1}) \sum_{i=0}^{\infty} \frac{(s\lambda_k)^i}{(i+1)!}\hat{\mathbf{z}}_k.$$

The series is $\varphi_1(s\lambda_k)$, by (1.14), and using (2.19) we have (2.20). □

As a consequence we can write

$$e^{sL_k}\mathbf{x} = \mathbf{x} + \Delta M_e(s)\hat{\mathbf{z}}_k, \tag{2.21}$$

where the parameter $\Delta M_e(s)$ is given by

$$\Delta M_e(s) = s(b_k x_{j_2} - a_k x_{j_1})\varphi_1(-s(a_k + b_k)), \tag{2.22}$$

which is the key to a new scheme introduced in §2.5.1. A crucial property of that scheme is a consequence of the following observation. We can show that if the j_1, j_2 entries in $\mathbf{x}$ (i.e. x_{j_1} and x_{j_2}) are both non-negative, then the j_1, j_2 entries in $e^{sL_k}\mathbf{x}$ are also non-negative.

Corollary 2.4.2. *With the same assumptions as in Lemma 2.4.1, the j_1, j_2 entries in $e^{sL_k}\mathbf{x}$ are given by $x_{j_1} + \Delta M_e(s)$ and $x_{j_2} - \Delta M_e(s)$ respectively. Both of these are non-negative if x_{j_1} and x_{j_2} are non-negative.*

Proof. The first claim follows simply from (2.21) and the form of $\hat{\mathbf{z}}_k$. For the second claim, consider $\Delta M_e(s)$ re-written as

$$\Delta M_e(s) = (1 - e^{-s(a_k+b_k)})\frac{(b_k x_{j_2} - a_k x_{j_1})}{a_k + b_k}.$$

The $(1 - e^{-s(a_k+b_k)})$ part is a monotonically increasing function, from 0 when $s = 0$ to 1 as $s \to \infty$, so it is in $[0, 1)$. The other part of $\Delta M_e(s)$ is $\frac{(b_k x_{j_2} - a_k x_{j_1})}{a_k + b_k}$ and can

be either positive or negative. We can consider each case separately.
If $\frac{(b_k x_{j_2} - a_k x_{j_1})}{a+b} \geq 0$, then

$$0 \leq \Delta M_e(s) < \frac{(b_k x_{j_2} - a_k x_{j_1})}{a_k + b_k},$$

and so,

$$x_{j_1} \leq x_{j_1} + \Delta M_e(s) < x_{j_1} + \frac{(b_k x_{j_2} - a_k x_{j_1})}{a_k + b_k},$$

and

$$x_{j_2} \geq x_{j_2} - \Delta M_e(s) > x_{j_2} - \frac{(b_k x_{j_2} - a_k x_{j_1})}{a_k + b_k} = \frac{a_k(x_{j_1} + x_{j_2})}{a_k + b_k},$$

so that both $x_{j_1} + \Delta M_e(s)$ and $x_{j_2} - \Delta M_e(s)$ are non-negative.
If $\frac{(b_k x_{j_2} - a_k x_{j_1})}{a_k+b_k} \leq 0$ then,

$$0 \geq \Delta M_e(s) > \frac{(b_k x_{j_2} - a_k x_{j_1})}{a_k + b_k},$$

leading to

$$x_{j_1} \geq x_{j_1} + \Delta M_e(s) > x_{j_1} + \frac{(b_k x_{j_2} - a_k x_{j_1})}{a_k + b_k} = \frac{b_k(x_{j_1} + x_{j_2})}{a_k + b_k},$$

and

$$x_{j_2} \leq x_{j_2} - \Delta M_e(s) < x_{j_2} - \frac{(b_k x_{j_2} - a_k x_{j_1})}{a_k + b_k}.$$

Thus we have that $x_{j_1}+\Delta M_e(s)$ and $x_{j_2}-\Delta M_e(s)$ are non-negative in either case. $\square$

Another useful property of connection matrices is to re-express the timestep implicitly defined by (2.10) (i.e., $\Delta t_k = \frac{\Delta M}{|f_k| A_k}$). We use the properties of the connection matrix L_k (see (2.13)) to express the flux from (2.6) as follows,

$$|f_k| A_k = \frac{||L_k \mathbf{m}||}{\sqrt{2}},$$

where $||L_k\mathbf{m}||$ is the Euclidean norm of $L_k\mathbf{m}$, and $\mathbf{m}$ is the current mass vector of the system, like in the expression (2.17). The result follows from the fact that the only two nonzero entries of $||L_k\mathbf{m}||$ are $+f_kA_k$ and $-f_kA_k$, from (2.13) and (2.18).

Then,

$$\Delta t_k = \frac{\sqrt{2}\Delta M}{||L_k \mathbf{m}||}. \tag{2.23}$$

We make use of (2.23) in our analysis in §4.1.

2.5 Modifications to BAS

It is possible to design modifications to the basic face based asynchronous scheme described in §2.2. We describe here two potential improvements: the use of a mass-passed value, and exact mass transfer across the face. The mass-passed value is an attempt to improve the synchronisation error between related faces, by tracking the mass that would have passed through faces near the event face. The exact mass transfer eliminates the error from the Euler discretisation of the mass transfer across a face. We use acronyms to refer to the new schemes. As the Basic face-based Asynchronous Scheme is BAS, the modification which tracks the mass-passed value is BAST. The Exact mass transfer Asynchronous Scheme is then EAS. A fourth scheme incorporating both modifications is EAST.

2.5.1 Exact Mass Transfer Across a Face - the EAS Scheme

Here is a practical consequence of the concept of connection matrices introduced in §2.4. The exponential mass passing idea is based on the exact solution to (2.13) over the timestep Δt_k, instead of the Euler-type approximation (2.16) which is used in Algorithm 2 lines $5-7$. The step is,

$$\begin{pmatrix} m_{j_1} \\ m_{j_2} \end{pmatrix} \leftarrow e^{\Delta t_k \tilde{L}_k} \begin{pmatrix} m_{j_1} \\ m_{j_2} \end{pmatrix}. \tag{2.24}$$

In this way the mass transferred during in an event is not the mass unit ΔM, however ΔM is still used to calculate the timestep Δt_k (equation (2.10)). This is done in order to allow us to use ΔM as a control parameter as in the other schemes.

We may write a global analogue to (2.24) in the same way as (2.17) and (2.16),

$$\mathbf{m} \leftarrow e^{\Delta t_k L_k}\mathbf{m} \tag{2.25}$$

The special structure of the connection matrix allows the matrix exponential to be replaced with a scalar computation. This is a result of Lemma 2.4.1. Using equation (2.21), we can write (2.25) as

$$\mathbf{m} \leftarrow e^{sL_k}\mathbf{m} = \mathbf{m} + \Delta M_e(\Delta t)\hat{\mathbf{z}}_k,$$

and thus only need to calculate $\Delta M_e(\Delta t)$ according to (2.22), and add or subtract it to each of the relevant two cells, during an event. The parameters a_k, b_k, λ_k for a face k can be pre-computed to make this more efficient. Line 5 of Algorithm 2 is replaced with the computation of $\Delta M_e(\Delta t)$ then lines $6 - 7$ are the same with δM replaced by $\Delta M_e(\Delta t)$.

The aim of exponential mass transfer is to take advantage of the fact that only tiny subsystems are updated during an event, and cheaply provide exact solutions on those subsystems. It is not expected to help with asynchronisation error (i.e., the fact that adjacent cells may be at different times, or the error in decoupling the faces for an event). An additional property of the exact mass transfer scheme is that from Corollary 2.4.2, an event will never overdraw a cell, thus preserving the positivity of the solution.

The approach described here has some similarities with the approximate Riemann Solvers of Roe, see [86] and, for example, [87]. In a Roe-type solver, the spatial discretisation is viewed as producing a series of Riemann problems (i.e., a conservation equation with discontinuous initial data), one at each face in the grid. Each Riemann problem can be approximately solved by introducing a matrix approximation at the face with certain properties. It may be an interesting approach in future work to attempt to apply a discrete event methodology to true Roe-type solver.

2.5.2 Using A Mass-Passed Tracking Value - the BAST Scheme

When an event occurs in the basic scheme, only one face is updated, while the associated faces are not. Consider adding an extra parameter to each face k, which is intended to track the mass that the face should have passed during an event of an associated face. Let this parameter be called the mass passed value, $\Delta M_{p,k}$.
We describe the implementation of $\Delta M_{p,k}$ to illustrate its intended function. First, for every face it is initialised to zero, and reset to zero when the face has an event. In Algorithm 2, during the loop of (lines 9-12) over each face l in the set of associated faces $\tilde{\mathcal{F}}_k$ of the active face k, *except k itself*, the mass passed value is updated as

$$\Delta M_{p,l} = \Delta M_{p,l} + (\hat{t}_k - t_l) A_l f_l. \tag{2.26}$$

Compare this to the second equation in (2.12). In (2.26), the mass-passed tracking value $\Delta M_{p,l}$ is incremented by the amount of mass that would have passed through face l during a timestep of length $\hat{t}_k - t_l$. Also, in the modified scheme every face of the cells j_1, j_2 has its time updated at this point,

$$t_l = \hat{t}_k,$$

as though these faces have also had events, although no transfer has occurred for these faces. The mass passed value effectively tracks the mass the faces would have passed in the time $[t_l, \hat{t}_k]$.
The mass passed value then adjusts the next event for a face k, depending on the size of $\Delta M_{p,k}$, as follows. The timestep approximation (2.11) is replaced with

$$\frac{\Delta M - \Delta M_{p,k}}{\hat{t}_k - t_k} \approx \frac{dm}{dt} = |f_k| A_k,$$

leading to the modified version of (2.10), the equation for $\hat{t}_k$,

$$\hat{t}_k = \begin{cases} t_k + \frac{\Delta M - \Delta M_{p,k}}{|f_k| A_k} \text{ if this} \leq T \\ T \text{ otherwise.} \end{cases} \tag{2.27}$$

Thus, faces will have increased priority for events if they have greater mass passed values. We can implement both the mass tracking and the exaxt mass transfer modifications to generate a fourth scheme, which we will call EAST. Unlike EAS, this scheme will be capable of overdrawing cells, since Corollary 2.4.2 does not apply to the ΔM_p value added to ΔM_e during an event.

2.5.3 The Cascading or ‘Flux Capacitor’ Concept of [1]

A crucial innovation in [1] is allowing cells to trigger their own events if they have been subject to too much activity without an event - each cell has a ‘flux capacitor’ value assigned, which is incremented each time a neighbouring cell has an event, and reset to zero when the cell itself has an event. The concept inspired our ‘mass passed’ value, §2.5.2. However, instead of affecting the update time of faces (or cells), the job of the flux capacitor value is, if and when it exceeds a certain threshold, to trigger a new event its cell, *independent of its update time and the priority queue.*

In a situation such as an advancing front or simply a region of high activity, this can lead to cells (or faces) constantly triggering their neighbours, following the path of high activity and ignoring the costly update time and priority queue calculations temporarily. This further emphasises the objective of DES methods to focus attention on the most active parts of the domain.

We have implemented this concept in our face-based Asynchronous schemes, using the similarity with the mass-tracking concept. These are mass-tracking schemes with the following modifications. First, the dependence of update time on $\Delta M_{p,k}$ is removed (i.e., instead of (2.27), we use the basic (2.10)). Second, when some face j has its $\Delta M_{p,j}$ incremented as part of an event on an associated face k, then an event is automatically triggered on j if $\Delta M_{p,j} > \Delta M$. We note there is potential to use other threshold values than the mass unit ΔM, but it seems to work well; see §3.1.

Because we expect these schemes to produce cascades of ‘cheap’ events along fronts or regions of very high activity, we refer to them as cascading schemes, however the underlying concept is of course nothing but the 'flux capacitor' of [1]. We have the basic scheme, BAS, augmented with the concept, which we will abbreviate as BAS-casc, and is analogous to BAST. We also have the exact mass type scheme, EAS, augmented with the concept, abbreviated as EAS-casc and analogous to EAST.

2.6 Adding a Reaction Term

We have experimented with including a reaction term, so that we may simulate conservation equations of the form

$$\frac{dc(\mathbf{x},t)}{dt} = \nabla f(c(\mathbf{x},t)) + r(c(\mathbf{x})), \qquad t \in \mathbb{R}, \qquad \mathbf{x} \in \Omega, \tag{2.28}$$

(i.e., (2.1) with a reaction term r. It is in principle possible to also apply our method to non-autonomous reaction terms $r = r(c\mathbf{x}, t)$. We have found that a simple leapfrog implementation of the reaction during events is effective. We describe the additions to the BAS method now. The first modification is that each cell now has its own independent time t_j. Now consider lines 5-7 of Algorithm 2, in which mass is transferred, replaced by the following method.

1. Calculate timestep values for each of the two cells, as $\Delta t_{j_1} = \hat{t}_k - t_{j_1}$ and $\Delta t_{j_2} = \hat{t}_k - t_{j_2}$.

2. Update the mass in both the cells according to the reaction term, using an Euler type step. For each cell use half the timestep for the cell. That is, perform the update,

$$m_{j_1} \leftarrow m_{j_1} + V_{j_1}\frac{\Delta t_{j_1}}{2} r\left(\frac{m_{j_1}}{V_{j_1}}\right),$$

$$m_{j_2} \leftarrow m_{j_2} + V_{j_2}\frac{\Delta t_{j_2}}{2} r\left(\frac{m_{j_2}}{V_{j_2}}\right),$$

3. The mass transfer across the face proceeds exactly as in the original scheme; lines 5-7 of Algorithm 2.

4. Repeat step 2.

Also, at line 8 of the algorithm, we set the cell times to be $\hat{t}_k$ alongside the face time. That is, $t_{j_1} \leftarrow \hat{t}_k$, $t_{j_2} \leftarrow \hat{t}_k$.

We now clarify this process. First consider step 2). The first thing to note is that it can be expressed in terms of concentration instead of mass simply as,

$$c_{j_1} \leftarrow c_{j_1} + \frac{\Delta t_{j_1}}{2} r\left(c_{j_1}\right),$$

$$c_{j_2} \leftarrow c_{j_2} + \frac{\Delta t_{j_2}}{2} r\left(c_{j_2}\right).$$

Ignoring for the moment the half timesteps, this is a single step of the Euler method (though with different timesteps for each cell) for the system consisting only of cells j_1 and j_2, and governed by the reaction-term-only PDE,

$$\frac{dc(\mathbf{x},t)}{dt} = r(c(\mathbf{x})), \qquad t \in \mathbb{R}, \qquad \mathbf{x} \in \Omega. \tag{2.29}$$

This is analogous to how in BAS we consider the system consisting only of the two cells j_1 and j_2 (and the internal face k), governed by the flux PDE (2.5). Step 4) corresponds to step 2) by 'completing' the halved Euler step. Steps 2) through 4) are thus simply an operator splitting method, applied to the tiny two cell subsystem considered by each event. Specifically it a leapfrog method, the simplest form of operator splitting.

This is a simple extension of the concept of our face based schemes to systems with reaction or source terms; the only technicality is the introduction of time values assigned to cells as well as faces, which is required to define timesteps for the reaction steps in a sensible way. This method retains the interesting property of needing to be explicitly based on a PDE. Indeed, the scheme can be implemented based on (2.5) for the faces and (2.29) for the cells, without any use or reference to (2.28). The modifications described here to BAS can be applied to any of our schemes; BAS, BAST, EAS and EAST.

We can imagine this method having difficulty with situations or parts of a domain where there is no flux between cells but still reactions within the cells changing the concentration values there - in this case the reaction activity could be 'missed' by the lack of events.

Chapter 3

Face Based Asynchronous Schemes - Numerical Experiments

3.1 Numerical Results for Systems Corresponding to Linear PDEs

Here we present some numerical experiments with the new schemes which demonstrate convergence and the relationship between scheme parameters such as N the total number of events, ΔM the mass unit and the average time step Δt. Here the test systems are linear PDEs, which produce ODE systems of the form (2.9) after a finite volume discretisation. The systems we investigate are advection-diffusion systems of the form (2.3), which we restate here for convenience,

$$\frac{dc(\mathbf{x},t)}{dt} = \nabla^2 D(\mathbf{x})(c(\mathbf{x},t)) + \nabla \mathbf{v}(\mathbf{x})c(\mathbf{x},t), \tag{3.1}$$

with different diffusivity $D(\mathbf{x})$ and velocity $\mathbf{v}(\mathbf{x})$ fields.

For our error estimations, see §1.5.2, we proceed as follows. For our comparison solve we use the exact solution of (2.9), i.e.,

$$u_{\text{comp}} = \mathbf{c}(T) = e^{TL}\mathbf{c}(0),$$

where the matrix exponential e^{TL} is approximated to very high accuracy by a Krylov subspace projection method, namely the code phipm [55]. T denotes the final simulation time. If the Finite Volume discretisation of the system has K cells, then the concentration vector $\mathbf{c}(t)$ is $\mathbf{c}(t) \in \mathbb{R}^K$. In our plots we also include solutions of (2.9) produced by the classical backwards Euler method, for comparison.

We used the package MRST [88] to produce our grids in Matlab for these examples. These grids were then used to create corresponding finite volume matrices for the phipm and classical solvers, and used to create corresponding grid objects in our Asynchronous schemes, which are implemented in C++. The grids we use are all regular Cartesian grids. See §4.6 for a discussion of how our methods might be extended to nonuniform grids.

With our results we include plots showing the concentration of events in the spatial domain, for example Figure 3.8 b). These 'heat maps' show the logarithm of the number events each cell has had during the solve (i.e., every time a face has an event, the two cells it is adjacent to are each considered to have an event). These heat maps of events show clearly how the asynchronous schemes vary computational effort across the spatial domain.

We will often compare the new schemes against a classical semi-implicit solver code. Applied to a semi-linear ODE system, the semi-implicit solver advances with an implicit time discretisation for the linear part and explicit time discretisation for the nonlinear part. Since the systems in this section are linear, we are effectively comparing against a classical implicit solver. We identify this scheme as 'Implicit-Solver' on our plots.

3.1.1 Fracture with High Péclet Number

In this example a single layer of cells is used, making the problem effectively two dimensional. The domain is $\Omega = 10 \times 10 \times 10$ metres, divided into 100×100 cells of equal size. Thus each cell has volume $0.1m^3$ and each internal face has area $1m^2$. The PDE to be solved is (3.1), and the diffusivity and velocity fields were prepared as follows.

A fracture in the domain is represented by having a line of cells which we will give certain properties. These cells were chosen by a weighted random walk through the grid (weighted to favour moving in the positive y-direction so that the fracture would bisect the domain). This process started on an initial cell which was marked as being in the fracture, then randomly chose a neighbour of the cell and repeated the process. This was done once to prepare the grid before the main tests. In this example, the fracture cells differs from the rest of the domain in that they have different permeability values. This has an effect on the pressure and thus velocity fields in the domain, as given by (1.6) and (1.1).

We use a permeability matrix of the form

$$\mathbf{K} = \begin{pmatrix} k_x & 0 \\ 0 & k_y \end{pmatrix}.$$

We set $k_x = 1$ in all cells except on the $x = 10$ boundary where it was set to zero, and $k_y = 2000$ on the fracture cells and $k_y = 1$ in all other cells, except on the $y = 10$ boundary where it was set to zero. See Figure 3.1 a). This intended to cause a large velocity in the y-direction inside the fracture, and a small velocity elsewhere. We use (1.6) to find a pressure field for this system. Dirchlet boundary conditions were imposed; with $p(x, 0) = 1$, $p(x, 10) = 0$, and no-flow conditions on the other edges. That is, high pressure along the $y = 0$ edge of the domain and low pressure at the $y = 10$ edge of the domain, creating a gradient. We then approximate the solution to (1.6) (a finite volume discretisation was used to produce a linear system which was then solved). The resulting pressure field can be seen in Figure 3.1 b). We then used a finite difference approximation of (1.1) (with ϕ, μ set to one and $\mathbf{g}$ set to zero) to find the resulting velocity field. The x- and y- velocities can be seen in Figure 3.1 c) and d), on a logarithmic scale. The y-velocity is extremely high in the fracture compared to elsewhere in the domain; this is expected to produce highly localised activity. Some streamlines are shown in Figure 3.1 e); here the flow is in the y-direction, i.e. bottom to top in the plot.

In every cell we set the diffusivity $D = 0.01$. A measure of the relative importance of advection compared to diffusion in a cell k is the Péclet number, Pe_k;

$$Pe_k = \frac{||\mathbf{v}_k||_\infty}{|D_k|},$$

where $\mathbf{v}_k$ and D_k are the velocity and diffusivity for the cell k, respectively. The infinity norm $||\cdot||_\infty$ for a vector is defined by $||(x_1, \dots x_n)^T||_\infty = \max\{|x_1|, \dots |x_n|\}$. We show the logarithm of the Péclet number in Figure 3.1 f). It varies by five orders of magnitude in the domain.

With the velocity and diffusivity fields thus prepared, we approximate the ADR (1.4) on this domain. Zero Neumann boundary conditions ('no flow') were applied on every boundary, and the initial condition was $c(\mathbf{x}) = 0$ everywhere except for $\mathbf{x} = (5.95, 0.05)^T$, where $c(\mathbf{x}) = 1$. That is, there is a single cell with concentration 1 on the bottom edge of the domain, close to the fracture. The final time was $T = 17$. The final solution approximated with schemes using $\Delta M = 10^{-8}$ is shown in Figure 3.2 a), c), e), and g). In Figure 3.2 b), d), f) and h) we show heat maps of the logarithm of the number of events experienced by a cell during the solve. We consider a cell to have had an event if one of its faces has an event. These plots shows how the activity is localised, as the number of events varies in seven orders of magnitude between a large part of the domain and the fracture. We note that these high accuracy solves are very similar for the schemes, as they all seem to converge to the correct solution for small ΔM (as shown in the following). The heat maps of events are also very similar, except for BAS which has a significant amount of activity throughout the domain.

The error was estimated as described at the start of this section. Figure 3.3 a) shows the convergence of the schemes with respect to ΔM; which appears to be first order. BAS is notably less accurate than the three modified schemes. Figure 3.3 b) shows efficiency by plotting error against cputime. For comparison we include two classical schemes. These are the semi-implicit scheme (actually fully implicit in this case since there is no nonlinearity), and forward Euler, the simplest scheme. The results for the two classical schemes are marked with triangles to distinguish them. We see that the improved asynchronous schemes are more efficient than the implicit solver for this problem. Surprisingly, the Euler scheme is more efficient than all other schemes for this problem, owing largely to its speed of execution (although stability is an issue). We have only plotted the results for Euler at the timestep values where it is stable. In most situations we expect instability to make Euler impractical; here it is able to out-compete the implicit solver and the implicit schemes because the relatively coarse mesh allows it to have a significant regime of stability.

In Figure 3.4 we show various parameter relations for the schemes. Plot a) shows error against average timestep Δt. Here the error of the schemes seems to be first order with respect to average Δt. In plot b) we show the total number of events N in a solve against the mass unit ΔM. Another strong relation can be observed; it seems that N is of order -1 with respect to ΔM. Plot c) shows average Δt against ΔM, and seems to imply another correlation of order one. Finally, plot d) shows average Δt against N, and implies that average Δt is order -1 with respect to N.

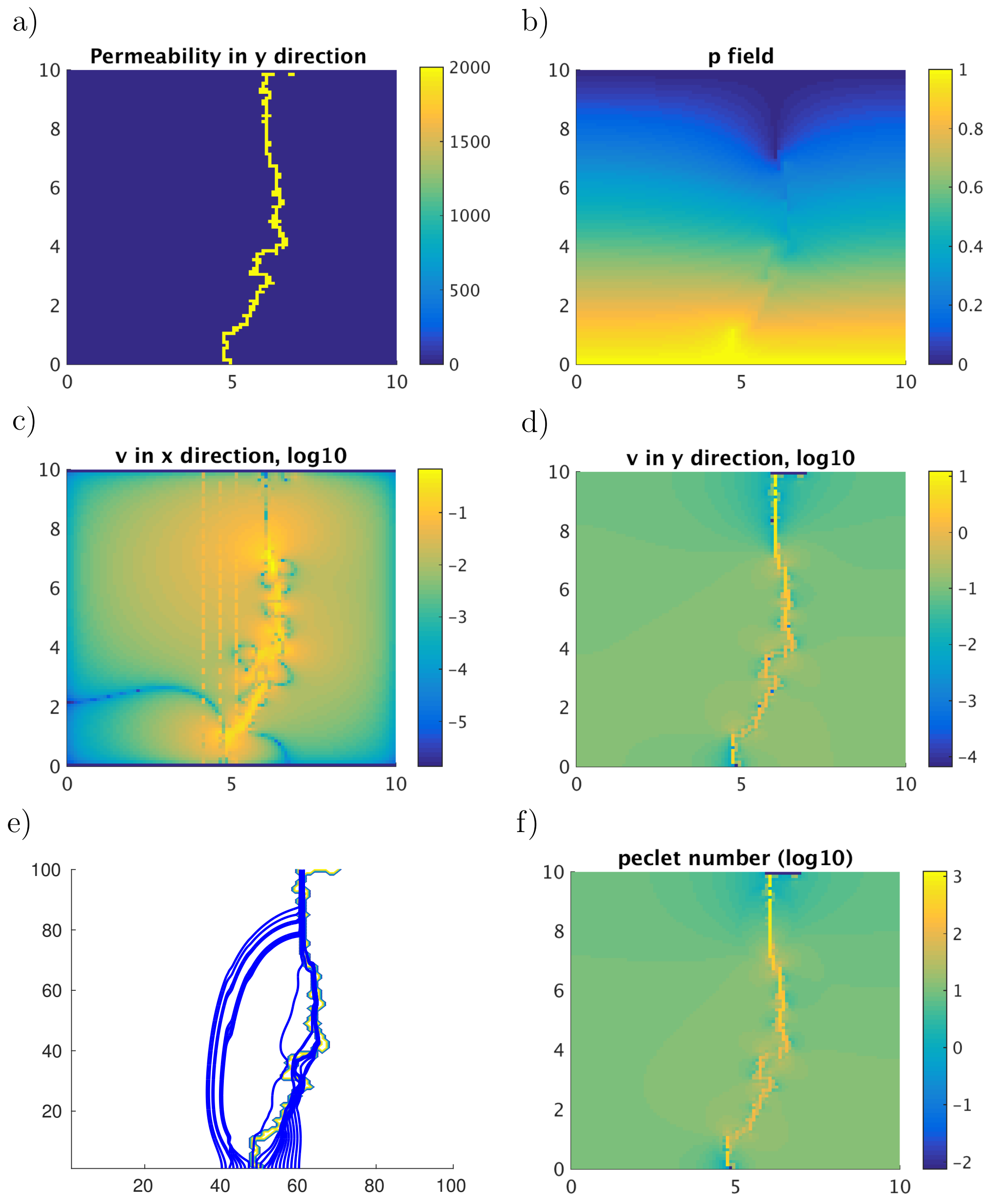
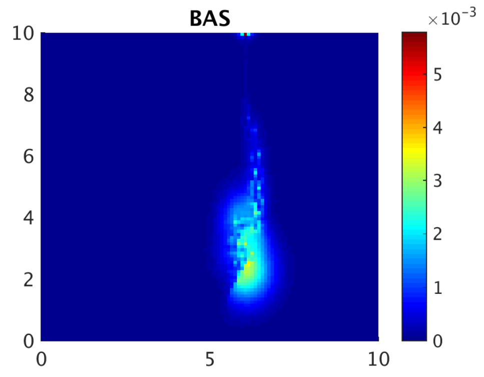
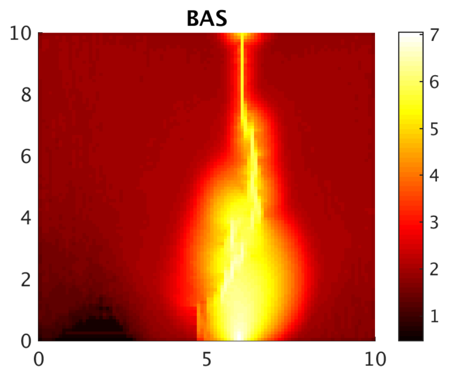


Figure 3.1: Permeability, pressure and velocity fields for the fracture example with varying velocity, §3.1.1. a) k_y, the permeability in the y-direction. b) The calculated pressure field. c) The calculated velocity field in the x-direction, logarithmic scale. d) The calculated velocity field in the y-direction, logarithmic scale. e) Some streamlines of the calculated velocity field; flow is from bottom to top. f) Péclet number, logarithmic scale.

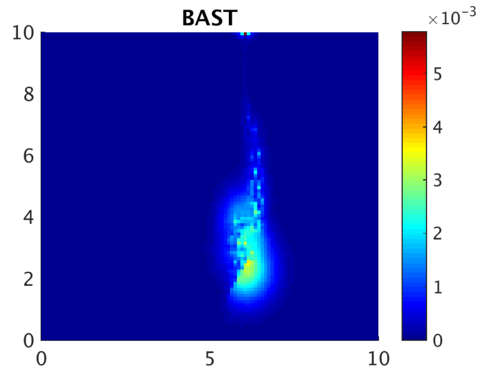
a)



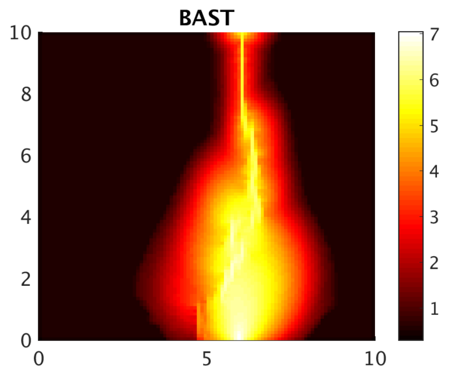
b)



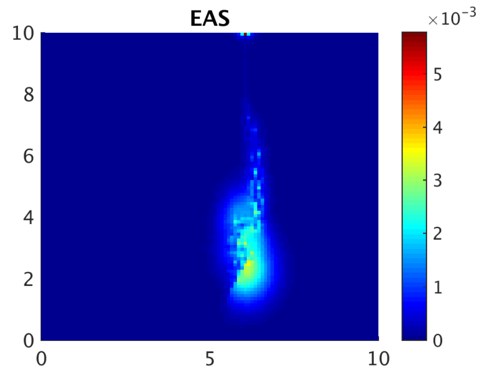
c)



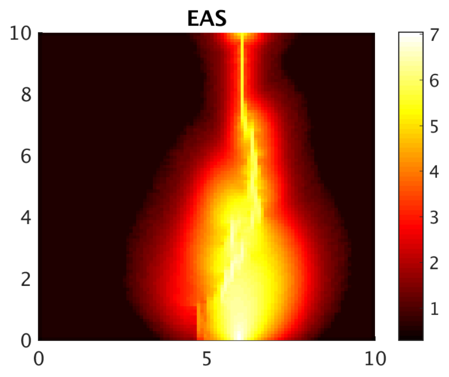
d)



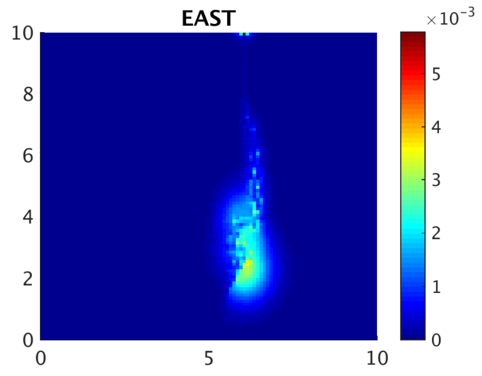
e)



f)



g)



h)

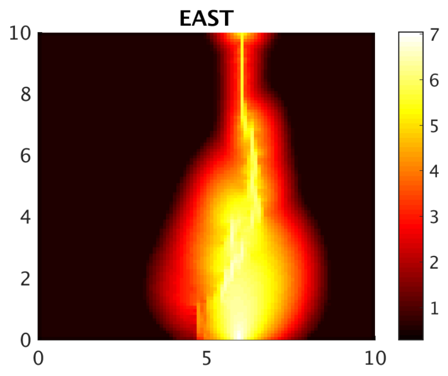


Figure 3.2: Final state of the first fracture example, approximated with each of the new schemes, with $\Delta M = 10^{-8}$. a), c), e), g) Concentration field. b), d), f), h) Events per cell, logarithmic scale.

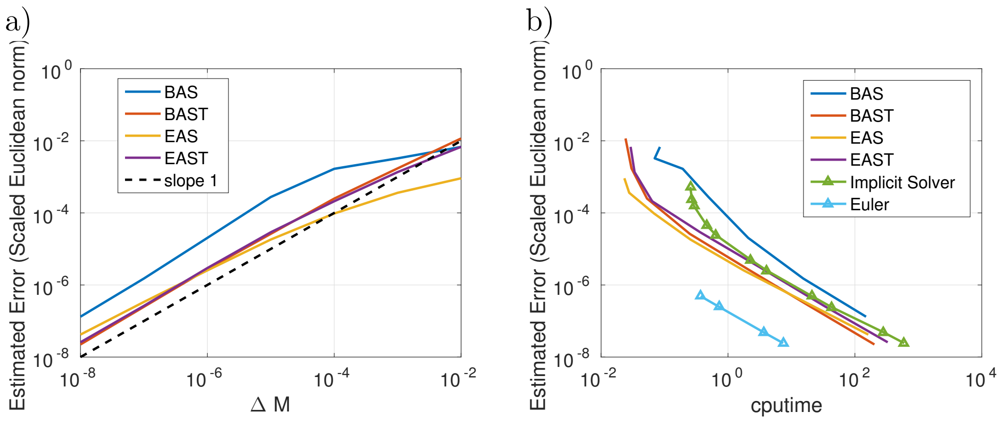


Figure 3.3: Convergence in ΔM and efficiency for the first fracture example. See §3.1.1.

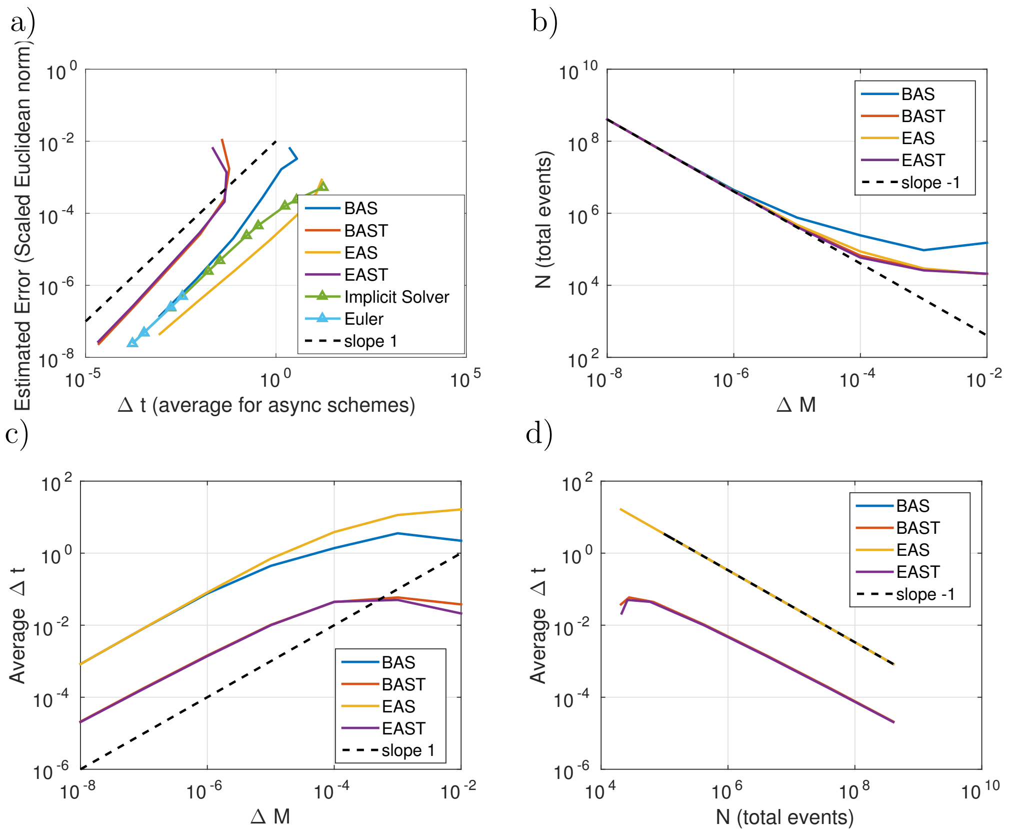


Figure 3.4: Further Results for the first fracture example as ΔM is reduced. See §3.1.1.

With Greater T

We repeat the experiment in the previous section with $T = 170$. We see that the same relationships between parameters hold for the asynchronous schemes for this longer timescale; Figures 3.5, 3.6. We have do not show the results with the classical Euler solver in these plots as it is only stable for the smallest two timesteps used (the timestep sizes are larger than in the previous example). The timestep range was from 0.017 to 170. We note that unlike for the T=17 experiment, here the classical implicit solver is more efficient than the asynchronous schemes; see Figure 3.5 b). A heat map of the logarithm of the number of events for each cell is given in Figure 3.7 for the BAST solver with $\Delta M = 10^{-8}$, showing the increased activity over time; compare to Figure 3.2 b).

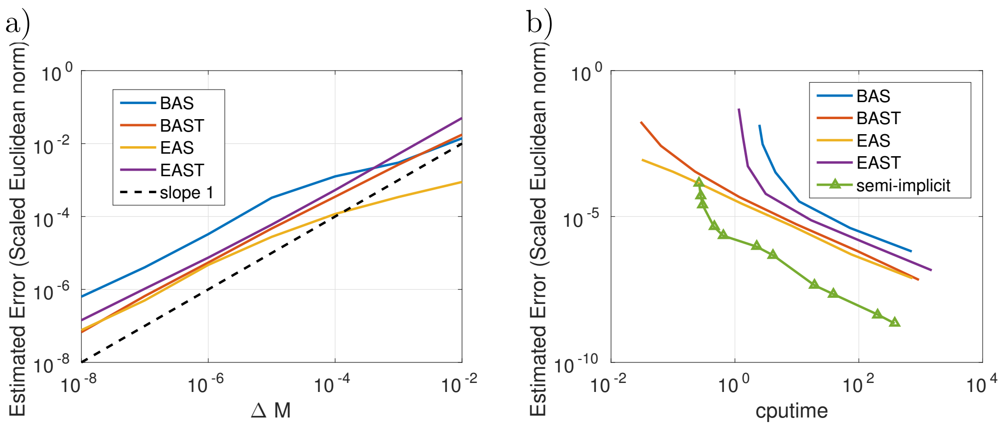


Figure 3.5: Convergence in ΔM and efficiency for the first fracture example with $T = 170$. See §3.1.1.

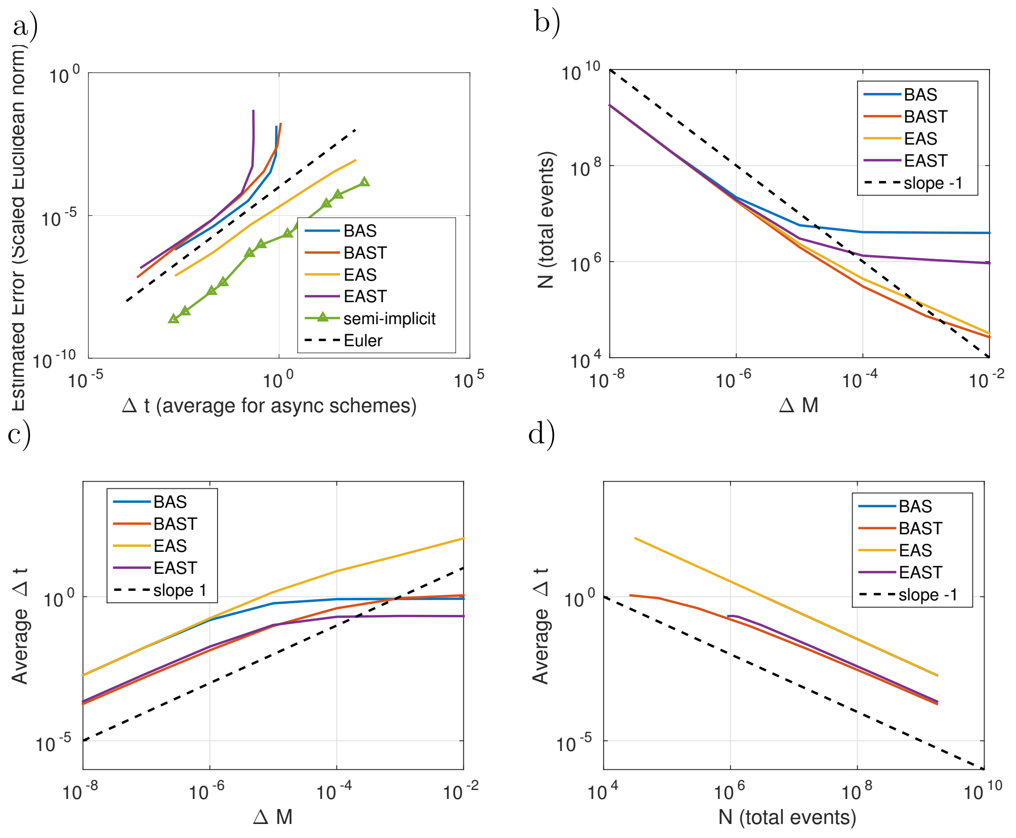


Figure 3.6: Further Results for the first fracture example with $T = 170$. See §3.1.1.

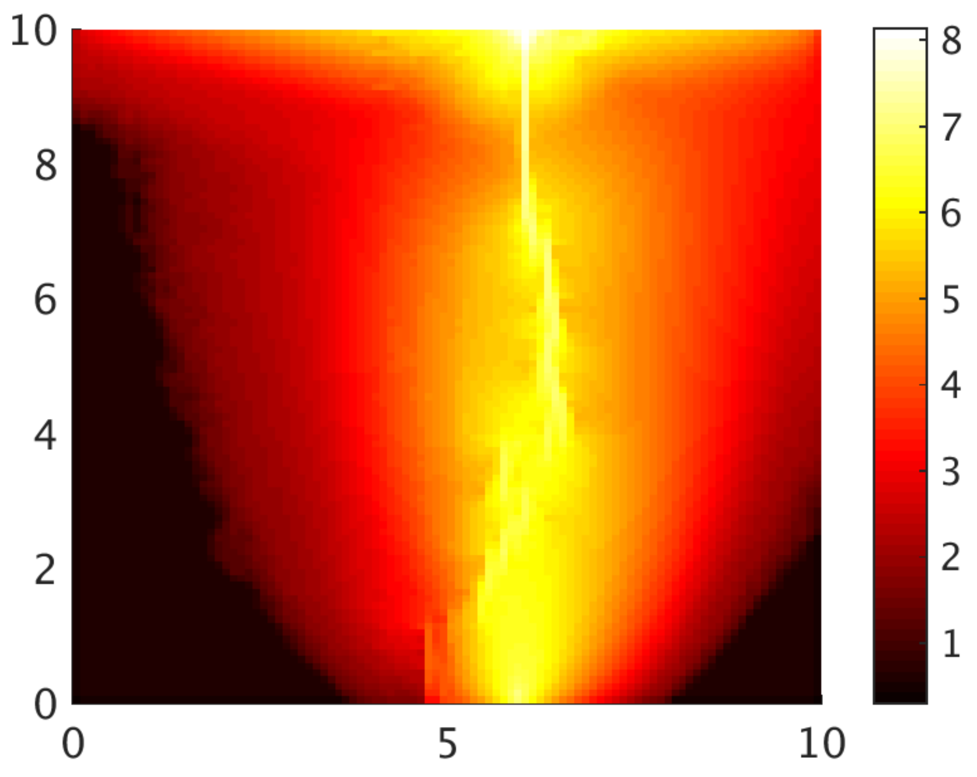


Figure 3.7: Heat map of logarithm of number of events per cell for the first fracture example with $T = 170$. See §3.1.1.

3.1.2 Fracture System with Varying Diffusivity

In this example the grid is the same as in §3.1.1. The same fracture structure is used, only now we set the diffusivity to be $D = 100$ on the fracture and $D = 0.1$ elsewhere. Figure 3.8 a) shows the solution at $t = 0.8$, and Figure 3.8 c) shows the solution at $t = 2.4$, as calculated by BAST with $\Delta M = 10^{-9}$. Figure 3.8 b) and d) are logarithmic heat maps of events for the solves corresponding to Figure 3.8 a) and c). It is interesting to note that the fracture cells can easily be identified in Figure 3.8 b), as the jagged line with extremely high activity.

In contrast to the previous example, we specify a simple velocity field. The velocity field was set to be uniformly one in the x-direction and zero in the other directions in the domain, i.e., $\mathbf{v}(\mathbf{x}) = (1, 0, 0)^T$, to the right in Figure 3.8. The initial condition was $c(\mathbf{x}) = 0$ everywhere except at $\mathbf{x} = (4.95, 9.95)^T$ where $c(\mathbf{x}) = 1$.

In Figure 3.9 are plots showing the convergence and efficiency of the schemes. In plot a) the estimated error is plotted against the mass unit ΔM, and we clearly observe that the error for all our schemes is $O(\Delta M)$ for sufficiently small ΔM. In plot b) the error is plotted against cputime to demonstrate efficiency.

In Figure 3.10 we show relationships between parameters such as ΔM, the total number of events N, and the average timestep. In Figure 3.10 a) the estimated error is plotted against timestep, Δt. For the four asynchronous schemes this is the average timestep across all events; for the backwards Euler solver it is the fixed timestep that was used. The implicit scheme has its classical $O(\Delta t)$ error; interestingly the error of the asynchronous schemes seems to also be first order with respect to the average timestep. Plot b) shows the total number of events N against ΔM. In plot Figure 3.10 c) the average timestep is plotted against the mass unit, and again there is a strong relation between the parameters for small enough mass unit. We appear to have $\Delta t = O(\Delta M)$. It is interesting that for schemes BAS and EAS the relations are the same in the limit despite the lack of obvious similarity between the schemes. For BAST and EAST too relationship is also the same in the limit, and these two schemes have smaller average timestep that BAS and EAS for all mass unit values. Plot d) shows average timestep against total events, and we have a clear $\Delta t = O(N)$ for BAS and and EAS (again having the same relation), and a very similar relation for BAST and EAST.

In Figure 3.9 a) and b) we see that the mass tracking and exact mass passing modifications improve the accuracy and thus the efficiency of the basic asynchronous face based scheme. For small ΔM EAS and BAST are the most accurate schemes, with BAST performing slightly better. We can observe at large ΔM that EAS is more accurate than BAST; this would be expected since BAST would severely overdraw cells of mass (i.e., producing a negative concentration) for large ΔM, while EAS is incapable of overdrawing at all. It is interesting that EAST, combining both the mass tracking and exact mass passing, is worse in terms of accuracy than both BAST and EAS, suggesting that the modifications are mutually incompatible. We can see from Figure 3.9 b) that the classical backwards Euler solver out-competes our schemes in terms of efficiency. We will discuss the efficiency of these particular implementation in the conclusion to this chapter.

For Figure 3.10 b), it is interesting how for all schemes the relationship between N and ΔM is the same for sufficiently small ΔM, as we see clearly $N = O(\Delta M^{-1})$. For larger mass unit values we observe that N is unchanging with respect to ΔM for BAS and EAST, although from plot Figure 3.9 a) we can see that the error is still decreasing for that range of mass unit values.

Plots Figure 3.10 c) and d), and to some extent b) (for large mass unit values), distinguish the schemes which use mass passed tracking (BAST and EAST) from those that don't (BAS and EAS). For large ΔM BAS uses the most events, while the two tracking schemes use less and EAS uses least of all (plot b). EAS uses by far larger timestep values on average (corresponding with its small number of events), and the tracking methods have the smallest timestep values on average (plot c). In plot d) we see that for a given number of events N, the tracking schemes will have a smaller average timestep for each event. The differences can be explained by the construction of the tracking methods. Recall that when a face k has an event in a mass tracking scheme, each associated face in $\tilde{\mathcal{F}}_k$ has a pseudo event in that the mass passed value for each face in $\tilde{\mathcal{F}}_k$ is incremented. The mass passed value then hastens future events, leading to shorter timesteps on average. See §2.5.2. Effectively the timesteps for the tracking schemes in Figure 3.10 c) and d) are systematically smaller because we do not record the 'effective' timestep length related to the mass passed value. We note that plot Figure 3.9 a) indicates that BAST having much smaller error than BAS, indicates that the attempt to increase coupling between faces represented by the mass tracking scheme, if quite successful. It is also interesting that at very large ΔM, Figure 3.10 plot b) shows us that EAS has the smallest number of events during a solve. This may be due to the overdrawing of cells. EAS is the only scheme guaranteed not to overdraw at all, and it is possible that in a large ΔM regime the other three schemes will produce 'wasted events', where mass is passed back and forth between cells to some extent.

The strong relations between parameters in Figure 3.10 plots b), c) and d) are highly interesting. The relation $\Delta t = O(\Delta M)$ implied by plot c) may be naively inferred from (2.23), however we cannot take this for granted since after any number of events the mass vector $\mathbf{m}$ can be expected to be different if a different value of ΔM is used for the solve. Thus we cannot rule out a priori that the denominator $||L_k\mathbf{m}||$ in (2.23) has some dependence on ΔM. The strong linear relationship (for sufficiently small ΔM) indicated in plot c) is promising, however it does provide information only on the average timestep over all events in a solve, and ideally further analysis is required to establish the relationship between Δt_k and ΔM. Plot b) indicates that for sufficiently small ΔM, the total number of events over the solve, for a given ΔM, is the same or almost the same, for each of our four different types of solver. One thing this could possibly indicate is the existence of some 'preferred path' of events, that is, an ordering of faces on which events occur, which in the limit $\Delta M \to 0$ all our schemes follow. This is merely a conjecture, but warrants further investigation.

For this experiment we also tested the cascading schemes described in §2.5.3. We compare these results with the results for the mass tracking schemes in Figure 3.11 and Figure 3.12; these have the same format as Figure 3.9 and Figure 3.10 respectively. We differentiate the cascading schemes by plotting them with star points. It is interesting that, although we might assume their behaviour to be quite different due to the possibility of automatically triggered events, the cascading schemes seem to exhibit the same parameter relations described in plots a) through d) in Figure 3.10 described above.

We can also see from Figure 3.11 plot a) that BAS-casc has comparable accuracy to BAST. By contrast, EAS-casc has worse - it is in fact worse than EAS. It appears that, similar to the situation with EAST, the exact mass transfer modification does not interact well with the cascading modification.

The most efficient Asynchronous scheme is BAS-casc, which is comparable to the classical scheme. It was shown in Figure 3.11 a) that BAS-casc has similar accuracy to BAST at small ΔM; in plot b) it is revealed that BAS-casc has a faster runtime and thus a better efficiency. This is presumably due to much more of the events in the BAS-casc run being the cheaper automatically triggered events.

In Figure 3.13 we demonstrate the effects of using an unsuitably large ΔM value for the solve; in this case $\Delta M = 10^{-5}$ is clearly large enough to be problematic. The effects we call 'checkerboarding', and the overdrawing of mass from cells are in evidence (we alluded to these effects in §2.2). We see how EAS (plot c) uniquely does not allow overdrawing and seems to have the smoothest, most physical solution. The most primitive scheme BAS (plot a) seems to have severe problems with artificial advection at this value of ΔM. Comparing Figure 3.13 to Figure 3.8 c), the most qualitatively correct scheme (with respect to general plume shape) appears to be EAST (plot d), albeit with a highly nonsmooth solution and plenty of overdrawn cells. Referring to Figure 3.9 a), we see that at this value of ΔM, EAST indeed has the smallest error norm, but the error norm for EAS is only slightly larger, possibly owing to the increased smoothness of the solution. From Figure 3.8 a) and c), and Figure 3.9 a), we have evidence that our class of face based asynchronous schemes can produce high accuracy results in sufficiently small ΔM regimes. From Figure 3.13 we also have the unsurprising result that, in regimes of unsuitably large ΔM, these schemes produce results that lack physical correctness. Most clearly they can produce schemes lacking smoothness and positivity (a,b,d).

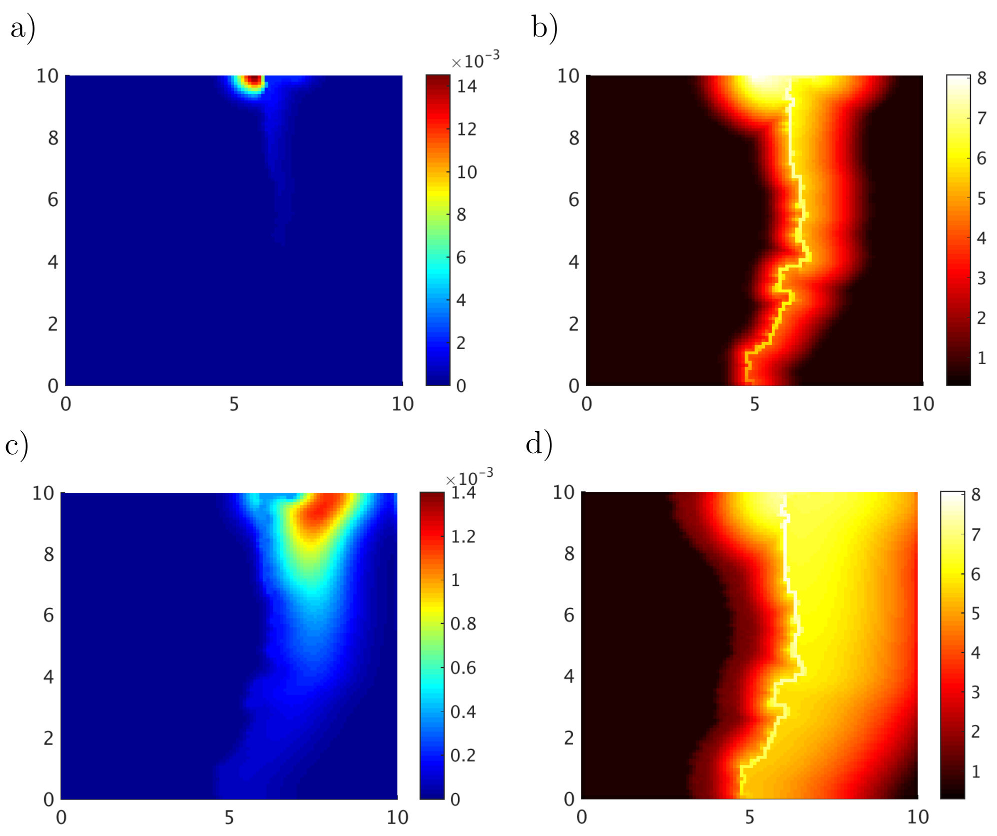


Figure 3.8: A 100 × 100 test advection diffusion system with a 'fracture' (line of high diffusivity). a) The system evolved up to $t = 0.8$. b) Heat map of logarithm of events per cell corresponding to a). c) The system evolved from $t = 0.8$ to $t = 2.4$. d) Heat map of logarithm of events per cell corresponding to c). Note that the colour scales in a) and c) differ by roughly an order of magnitude. The Maps of events make it clear where the fracture is, and show that the scheme is focusing effort there, where the system is evolving the fastest. These results were produced by the BAST scheme with $\Delta M = 10^{-9}$.

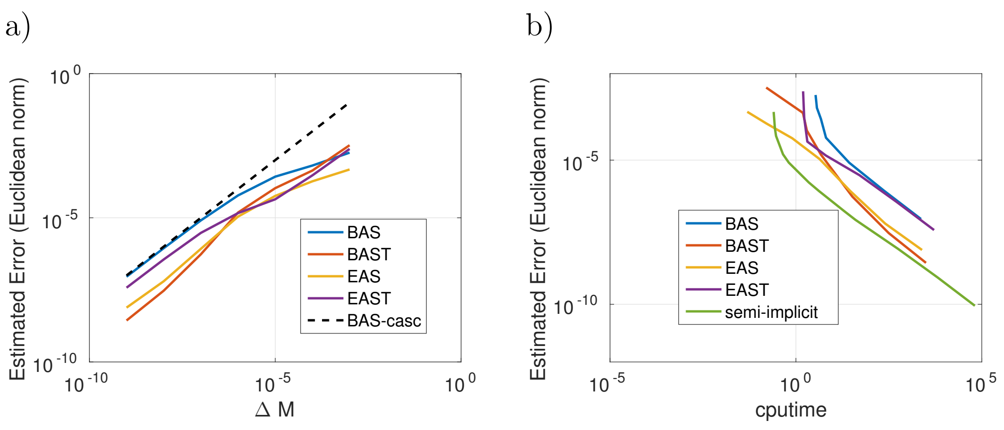


Figure 3.9: Results for the fracture system in Figure 3.8 concerning error convergence in ΔM and efficiency. For discussion see text; §3.1.2.

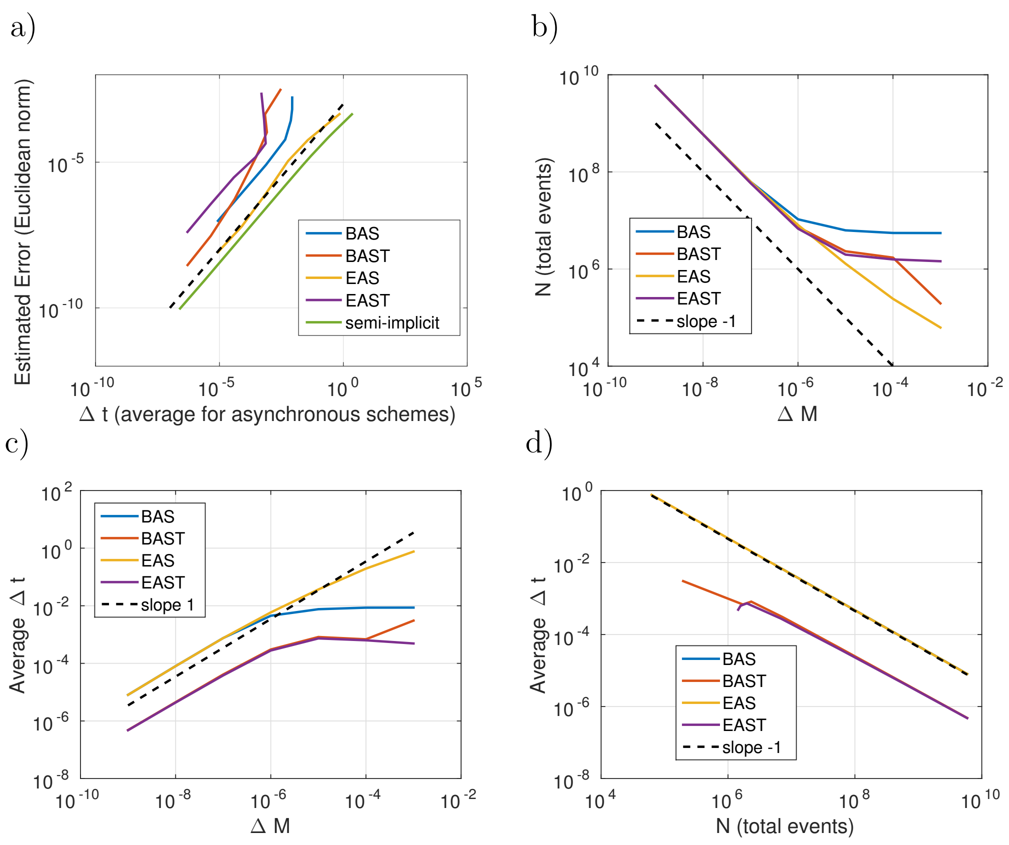


Figure 3.10: Results for the fracture system in Figure 3.8 concerning relationships between parameters. For discussion see text; §3.1.2.

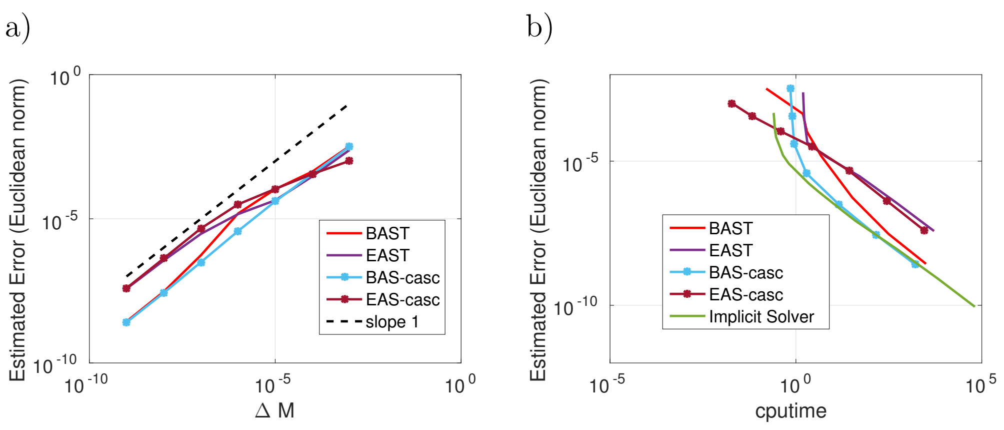


Figure 3.11: Results for the fracture system in Figure 3.8 concerning error convergence in ΔM and efficiency, comparing the mass-tracking with the cascading schemes. For discussion see text; §3.1.2.

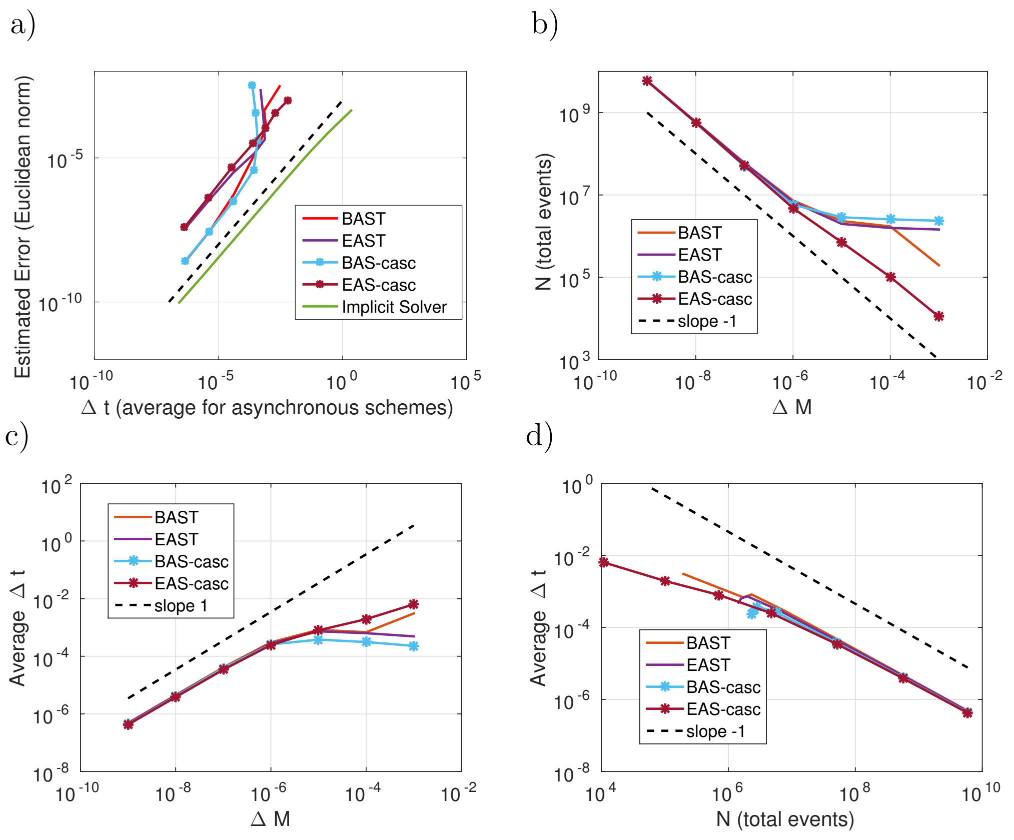


Figure 3.12: Results for the fracture system in Figure 3.8 concerning relationships between parameters, comparing the mass-tracking with the cascading schemes. For discussion see text; §3.1.2.

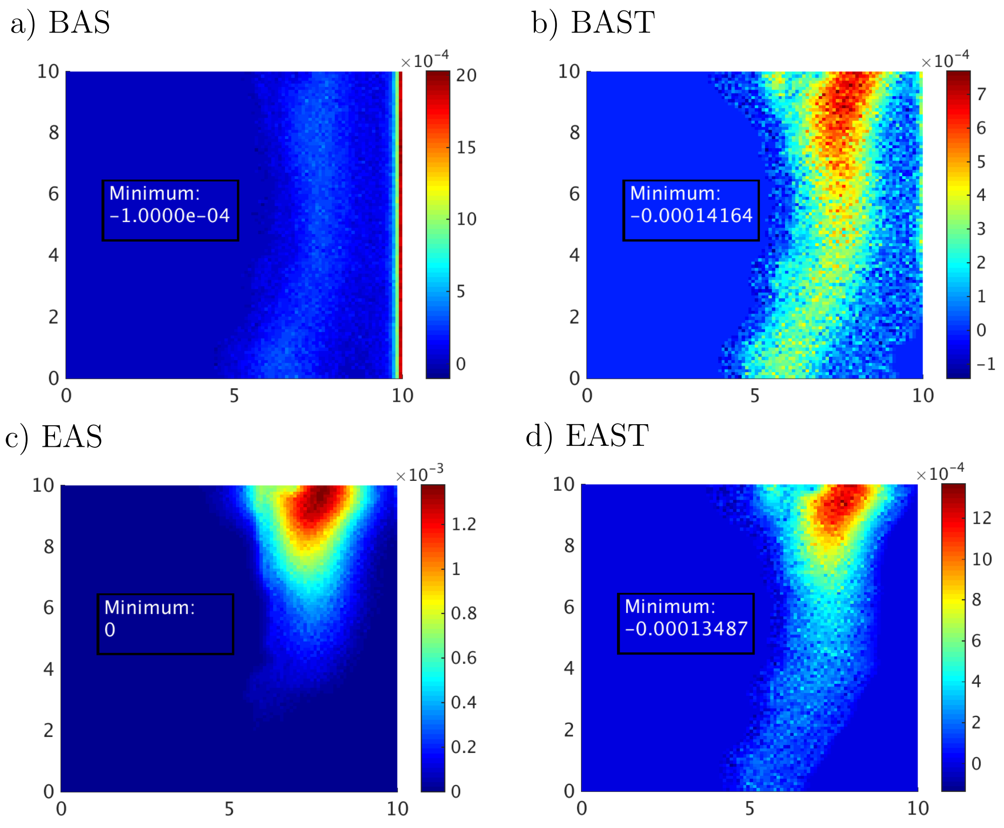


Figure 3.13: Demonstrating the effects of using unsuitably large ΔM (10^{-5}) values for the asynchronous schemes, for the fracture example. Here we can observe ‘checkerboarding’ (where concentration does not vary smoothly in space, causing adjacent cells to seem to alternate in colour like a checker board), and the overdrawing of cells (where more mass is removed than is present in a cell, causing the mass and thus concentration to become negative). These plots are to be compared to Figure 3.8 c). On each plot we have included the minimum concentration value on the domain, to demonstrate the overdrawing effect. EAS does not cause overdrawing by design, and also seems to suffer least from checkerboarding (plot c). Also interesting is that BAS seems to have caused artificially increased advection (plot a).

3.1.3 A Three Dimensional Example

In this example the domain is $\Omega = 10 \times 10 \times 10$ metres again, discretised into $40 \times 40 \times 32$ cells, for a total of 51200 cells in the system. We solve (3.1) with a diffusivity field that is uniformly $D(\mathbf{x}) = 2$ and a constant velocity field $\mathbf{v}(\mathbf{x}) = (0.1, 1.1, 0)^T$. The initial condition is sinusoidal, varying between 0 and 1, on the line of cells where $y = z = 0$, and zero elsewhere. The final time was $T = 2.4$.

We show the state of the system at the final time T in Figure 3.14 a), as produced by BAST with $\Delta M = 1.9532 \times 10^{-10}$. This solution is very accurate to our comparison solve produced using the exponential integrator; see Figure 3.15 a). Plot b) in Figure 3.14 shows the logarithm of the number of events experienced by each cell during the same BAST solve. We see again how the scheme automatically focuses more computational effort, in the form of transfer events, at different areas of the domain according to local rate of activity. There is about a difference of five orders of magnitude in number of events between the least and most active cells in Figure 3.14 b).

In Figure 3.15 we present comparisons of various parameters for the schemes; the format is the same as Figure 3.9 and Figure 3.10, and many of the conclusions similar. In Figure 3.15 a) first order convergence of the schemes with ΔM, for sufficiently small ΔM, is again observed. Plot b) compares the efficiency by plotting the estimated error against cputime. From plots a) and b) we can see that EAS is the most accurate and efficient scheme for large ΔM, while BAST is slightly more accurate and increasingly the most efficient as ΔM decreases. These are the same conclusions as for the previous example. We do not include comparison with a classical solver for this example.

Plots a) through d) in Figure 3.16 indicate relationships between the parameters ΔM, N (total number of events), average Δt, and error, of the schemes. The plots imply the same relationships as in the previous example. The relationships are as follows. From plot a), Error $= O(\Delta t(\text{average}))$ for sufficiently small ΔM (although BAST and BAS appear to have steeper slopes at certain parts of the plot). From plot b), for sufficiently small ΔM, $N = O(\Delta M)$, and there is also, just as observed in Figure 3.10 apparently a single line on the $\log(N)$ vs $\log(\Delta M)$ plot that the plots for all schemes seem to converge to. See §3.1.2 for discussion. From plot c), we have that $\Delta t(\text{average}) = O(\Delta M)$ for ΔM sufficiently small, and finally from plot f), $\Delta t(\text{average}) = O(N)$. The last relation for plot d) may not hold for BAS and BAST for ΔM too large - observe the change in slope for the smallest N value.

In Figure 3.17 we have displayed the effects of using inappropriately large ΔM. We can draw similar conclusions as from Figure 3.13. All schemes have suffered loss of continuity ('checkerboarding') to some extent, EAS the least. All schemes except EAS have overdrawn cells (i.e., negative concentration values).From plot a) we observe that BAS has some problem with increased advection, comparable to Figure 3.13 a). EAS appears to preferable if a large ΔM solve is necessary, as already mentioned in discussion of Figure 3.15 a) and b). However it is clear from Figure 3.15 and Figure 3.17 that a small ΔM is preferable. In general for a sufficiently small ΔM all the schemes appear to enter a regime of first order convergence, and a high level of correlation between parameters.

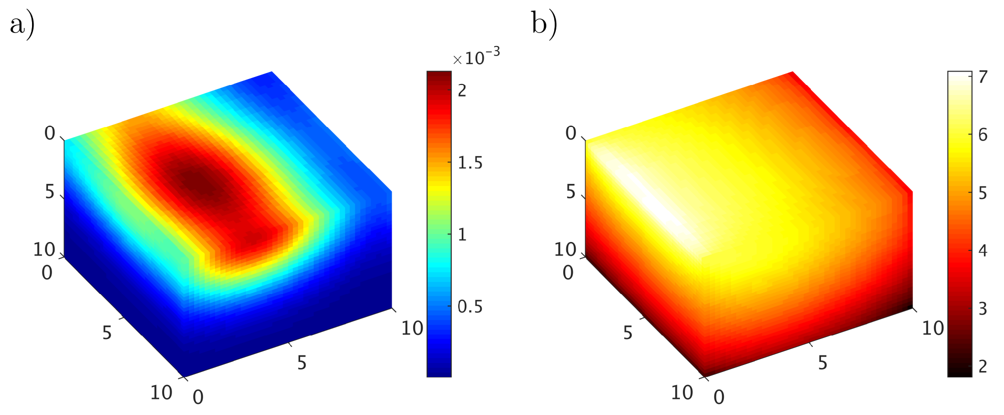


Figure 3.14: Three dimensional advection-diffusion system, see §3.1.3. a) result using BAST and $\Delta M = 1.9532 \times 10^{-10}$. b) heat map of logarithm of number of events per cell for the same solve.

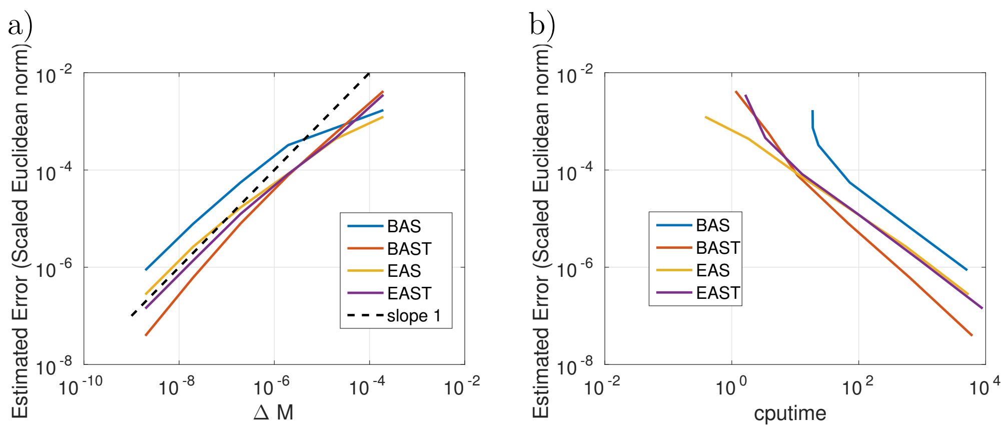


Figure 3.15: Results for the 3D system in Figure 3.14 concerning error convergence in ΔM and efficiency. For discussion see text; §3.1.3.

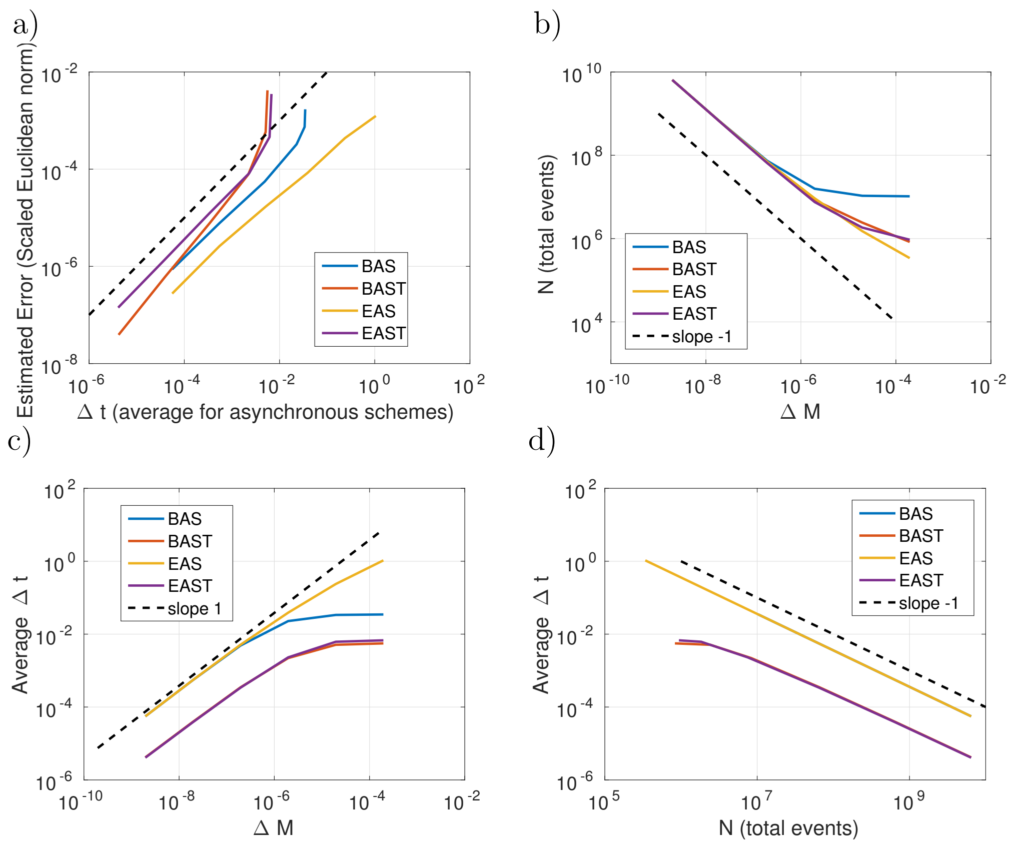


Figure 3.16: Results for the 3D system in Figure 3.14 concerning relationships between parameters. For discussion see text; §3.1.3.

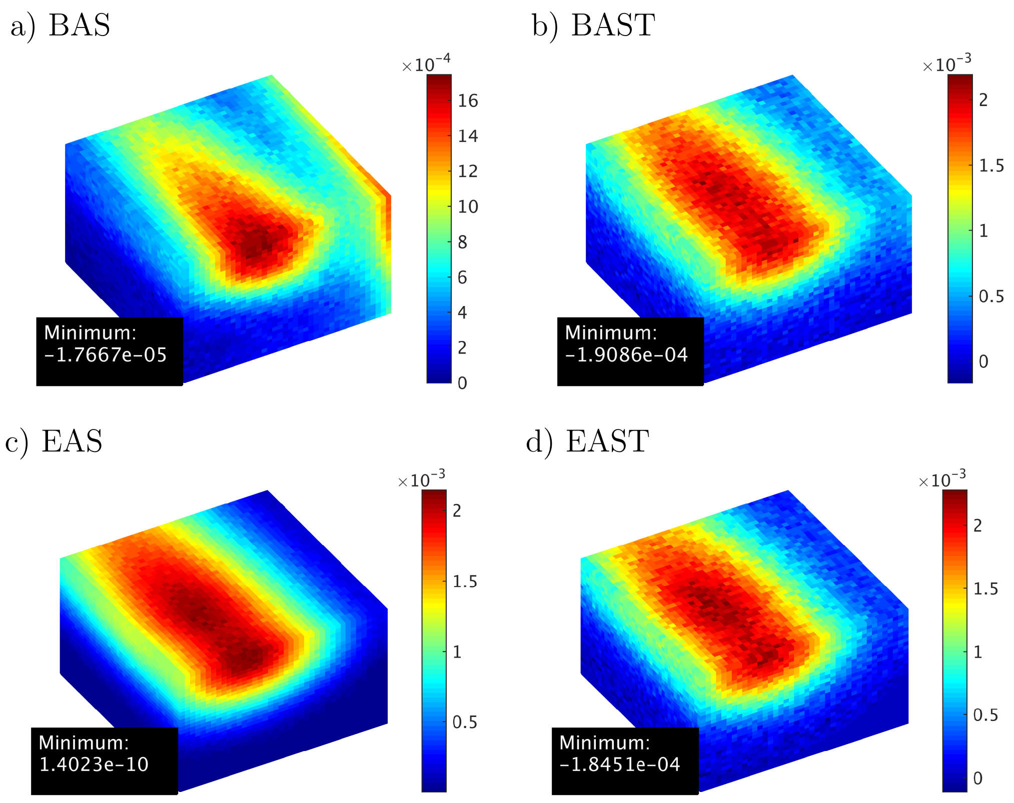


Figure 3.17: Effect of using unsuitably large ΔM (1.9532×10^{-6}) values with each of the schemes with the 3D example of §3.1.3. Compare to Figure 3.14 a). The minimum value of u in any cell in the domain is displayed to demonstrate the overdrawing effect.

3.1.4 Diffusion With a Random Diffusivity Field

The third test is in two dimensions, with a random diffusivity field. The PDE is again (3.1). The domain is again $\Omega = 10 \times 10 \times 10$ metres and discretised into $100 \times 100 \times 1$ cells. The diffusivity field is as follows. We prepared a Normal, mean-zero, random field $\psi(\mathbf{x})$ over the cells with correlation function

$$C(X,Y) = e^{\frac{||X-Y||}{l}},$$

with the correlation length between the x and y directions being $l = 9$. The diffusivity field used was then $D(\mathbf{x}) = 10.0^{\psi(\mathbf{x})}$. The field was generated using the standard Cholesky technique (see for example [59]); there was no need for approximation due to the relatively small number of cells. The velocity field was uniformly zero.

For this test the concentration was $c(\mathbf{x}) = 0$ for all $\mathbf{x}$ except at $\mathbf{x} = (4.95, 5.05)^T$ where $c(\mathbf{x}) = 0$. The boundary conditions were no-flow on all boundaries.

In Figure 3.18 is displayed the comparison solve (produced by the exponential integrator), the diffusivity field, the solve produced by BAST with $\Delta M = 10^{-9}$, and the logarithm of the number of events for the same solve. There is a difference of about six orders of magnitude between the most and least active parts of the domain.

In Figure 3.19 and Figure 3.20 are convergence and efficiency results and parameter relations for this system. Broadly, most of the conclusions are the same as from from previous experiments. See the discussions in §3.1.2 and §3.1.3. There are some differences for this system, however.

In Figure 3.19 a) there is clearly a decrease in the error as ΔM decreases - all the schemes appear to converge. However the plots for every scheme are not as straight and clearly first order for sufficiently small ΔM as with the previous examples. Indeed, here it seems that BAS, BAST, and EAST are all quite neatly first order for *large* ΔM, before the plots become slightly erratic for smaller ΔM. EAS seems to be converging of quite low order at large ΔM.

Generally, the implications of of Figure 3.20 a) through d) are the same as for Figure 3.11 and Figure 3.16 a) through d), discussed in §3.1.2 and §3.1.3. Note in particular that again the lines for every scheme seem to converge together in plot b), as in the other examples. The implication from plot a) that Error $= O(\Delta t(\text{average}))$, for sufficiently small ΔM, may be called into question for this example. It is certainly true for EAS, but BAST again seems to have a steeper relation for the smallest Δt(average). We have talked about a small ΔM regime in which the parameters of the schemes have strong correlation, most noticeable in plots a), b) and c) of Figures 3.11, 3.16 and 3.20. Comparing the three figures, and particularly Figure 3.20 with Figure 3.10 (the two systems have the same size domain and discretisation), we can see that this small ΔM regime starts at a comparatively smaller value for this random diffusivity field example. That is,in the plots a), b) and c) of Figure 3.20, the less of the regime where there is clear first order correlation between parameters is visible. This is likely due to the increased complexity of the system due to the random diffusivity field.

Finally we have Figure 3.21, demonstrating the effect of using too large ΔM values. The discussion for this figure differs little from corresponding discussion in for the previous examples. It is interesting that BAS still fares by far the worst, but this time the main problem cannot be identified as related to advection (there is none). The loss of continuity for BAST and EAST (plots b) and d) seems possibly worse for this example. The extent to which EAS does not suffer continuity problems compared to the other schemes is very marked here.

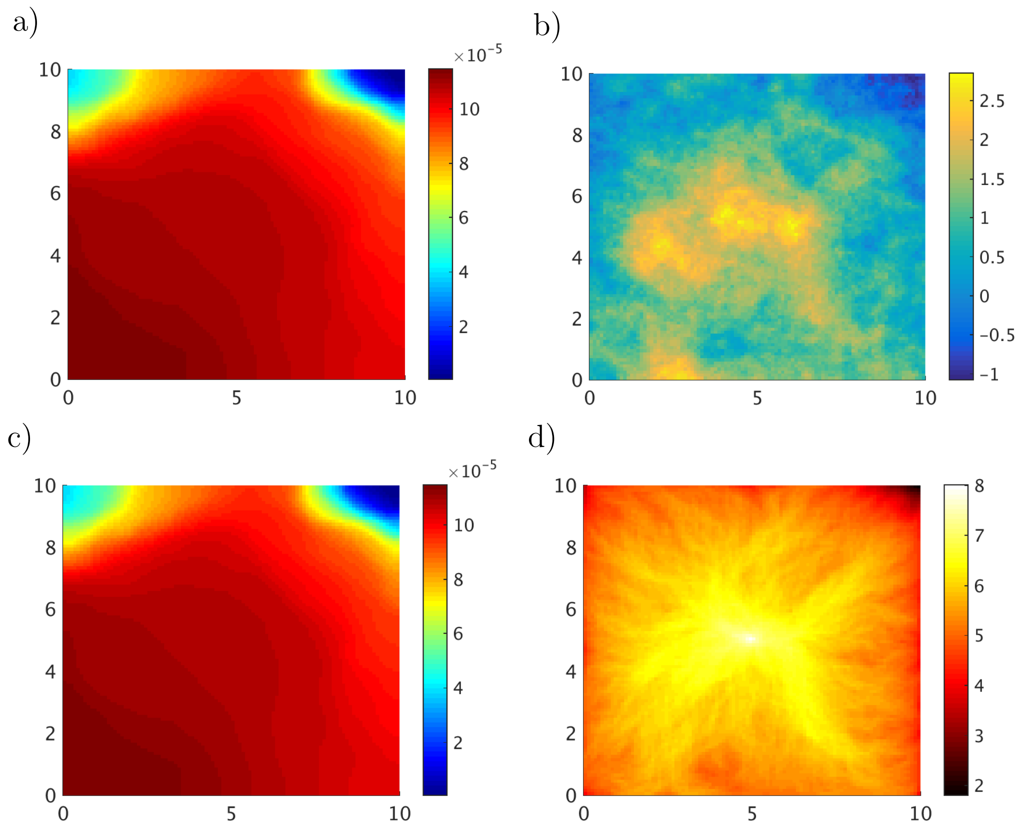


Figure 3.18: Two dimensional diffusion example with random diffusivity field. Plot a) shows the comparison solve using the exponential integrator. Plot b) shows the logarithm of the diffusivity field. Plot c) shows the solution using BAST and $\Delta M = 10^{-9}$. Plot d) shows the logarithm of events for each cell as a heat map.

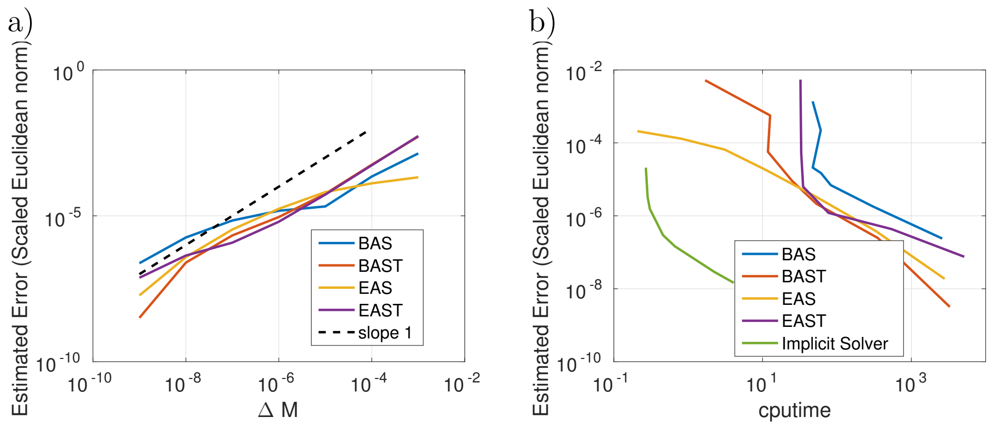


Figure 3.19: Results for the 2D diffusion example with random diffusivity field from Figure 3.18, concerning error convergence in ΔM and efficiency. For discussion see §3.1.4.

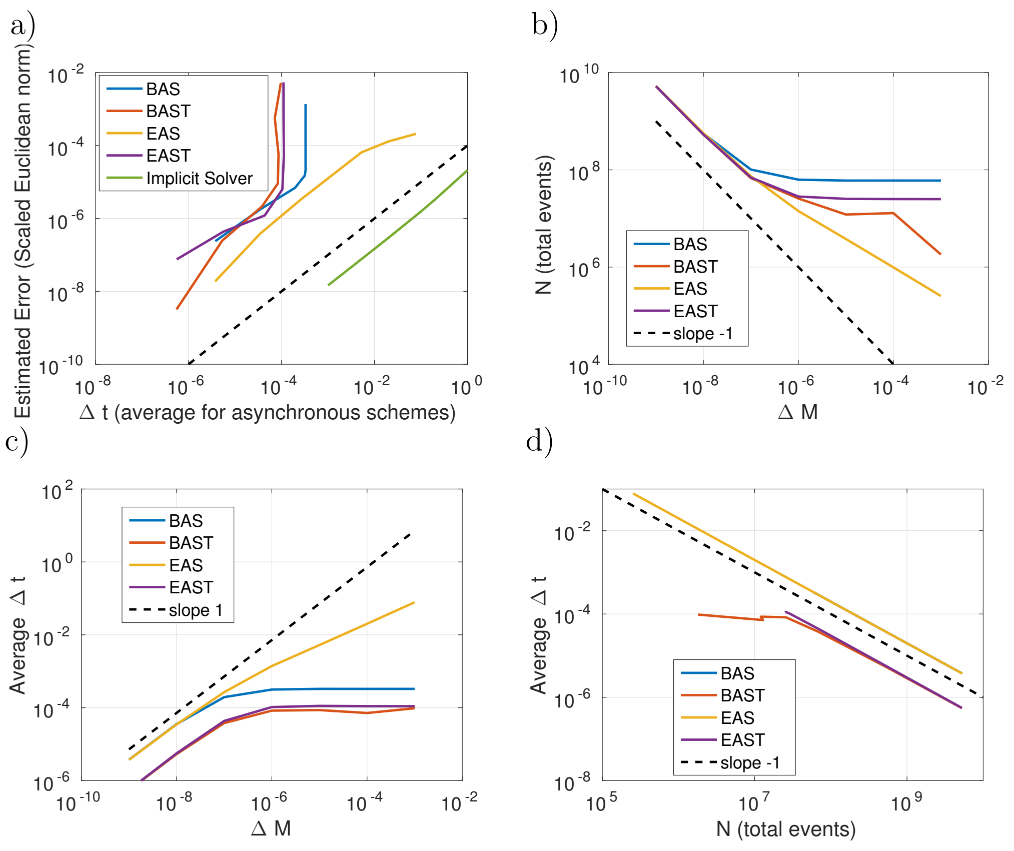


Figure 3.20: Results for the 2D diffusion example with random diffusivity field from Figure 3.18, concerning relationships between parameters. For discussion see §3.1.4.

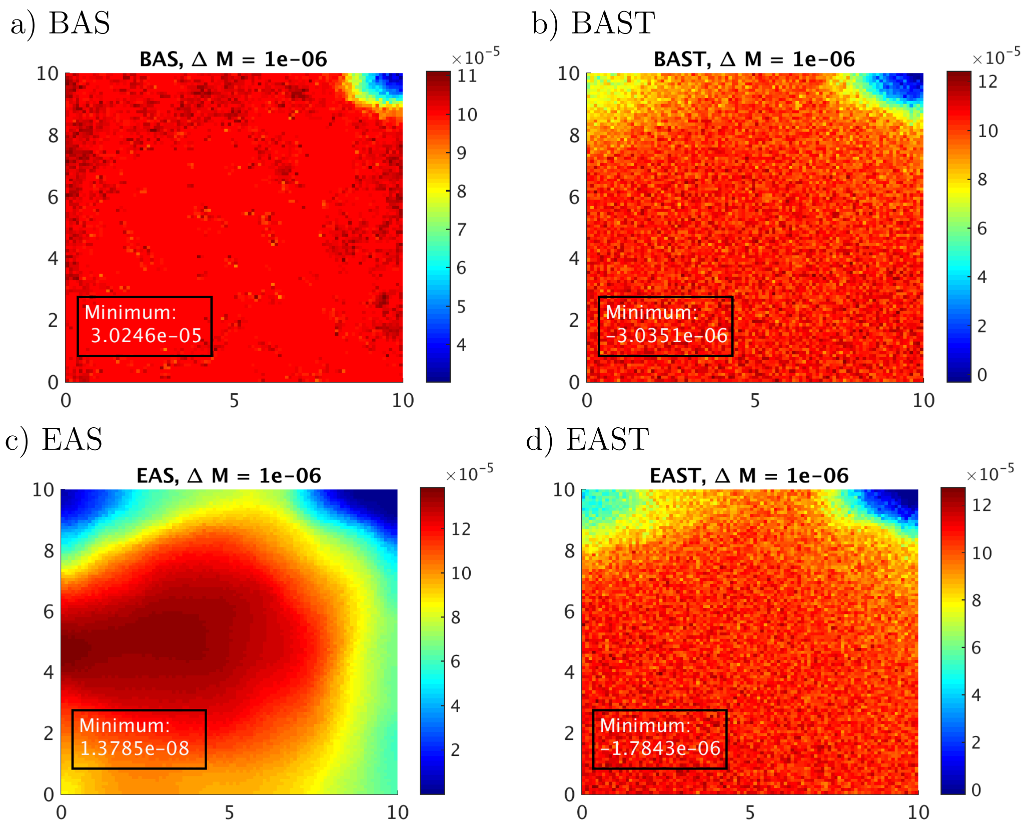


Figure 3.21: Demonstrating the effects of using unsuitably large ΔM for the random diffusivity field example of §3.1.4. Compare to Figure 3.18 a) and c). The minimum value of u is included to demonstrate the overdrawing effect.

3.2 Tests for the Reaction Term Scheme

These tests proceed in a similar way to the previous section, except now a full semilinear PDE is used, and the reaction term modification to the Asynchronous schemes is used. We will compare against the same semi-implicit solver as in the previous section, which is now acting truly as a semi-linear scheme as the PDE is semilinear.

3.2.1 Reaction-diffusion test system

This experiment is a simple small reaction-diffusion system in two dimensions with a Cartesian grid, intended to test the leapfrog type reaction term implementation described above. The velocity field was uniformly zero. The reaction term used is

$$r(c) = -\frac{c}{1+c},$$

which is a Langumiur-type reaction term, which can be used to model, for example, mass in the system adsorbing to the walls of the porous medium and thus being lost (see for example, [89], [40]). In our example a high concentration in the centre of the domain diffuses outwards (the diffusivity field is uniform) while reacting according to this term. The final time is $T = 1$ and the domain, discretisation, boundary conditions and initial condition are the same as in §3.1.4.

The results figures follow the same pattern as in §3.1. In Figure 3.22 we show the best solution, produced again by BAST. In Figure 3.23 we show the convergence and parameter relations of the schemes. In addition to comparing against the semi-implicit classical scheme, we compare against the exponential integrator ETD1 in plots Figure 3.23 b) and Figure 3.24 a). Interestingly, the parameter relations revealed in Figure 3.24 plots a) through d) are the same as those from the experiments in §3.1. Again, Figure 3.23 plot a) shows that error of the schemes converge to zero as ΔM decreases to zero. The schemes are still first order with the complication of a reaction term. In this example EAS is more accurate and efficient than is expected at this point - in plot b) we see that it has efficiency comparable to BAST, and in plot a), is only slightly less accurate.

We show in Figure 3.25 the results of using an unsuitably large ΔM. The observations are in line with the experiments for linear systems. Every scheme overdraws except EAS, which also produces a much smoother solution. The solution produced by BAS is the most problematic, in that it seems to not reproduce the correct circular shape of the plume; instead it is a rounded square shape.

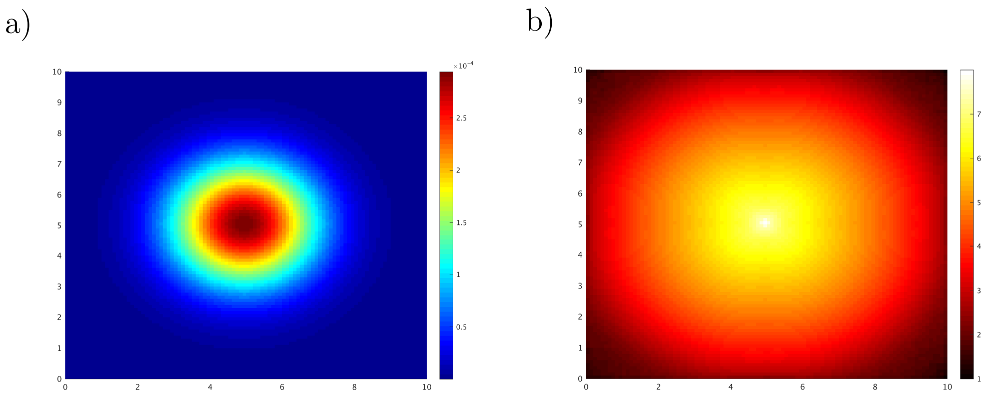


Figure 3.22: Two dimensional reaction-diffusion system. Left: solution with BAST with $\Delta M = 10^{-9}$. Right: heat map of logarithm of events for each cell.

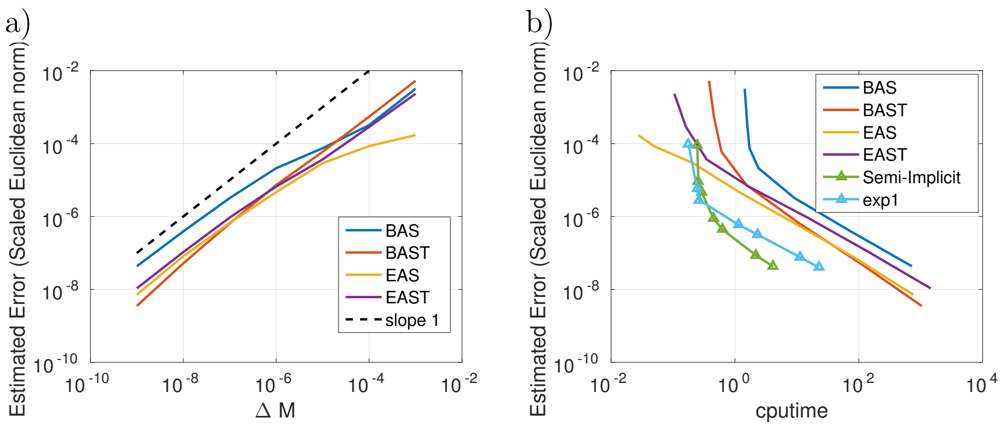


Figure 3.23: Results for the reaction-diffusion system in Figure 3.22, concerning error convergence in ΔM and efficiency. For discussion see text; §3.2.1.

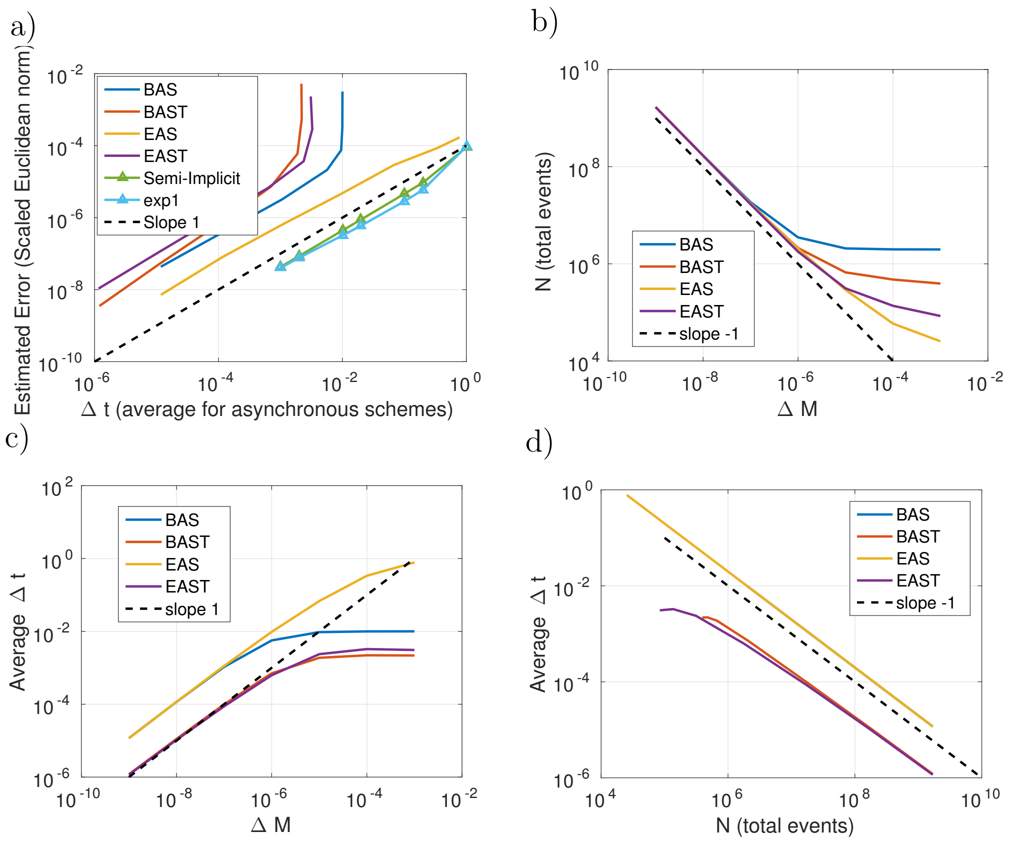


Figure 3.24: Results for the reaction-diffusion system in Figure 3.22, concerning relationships between parameters. For discussion see text; §3.2.1.

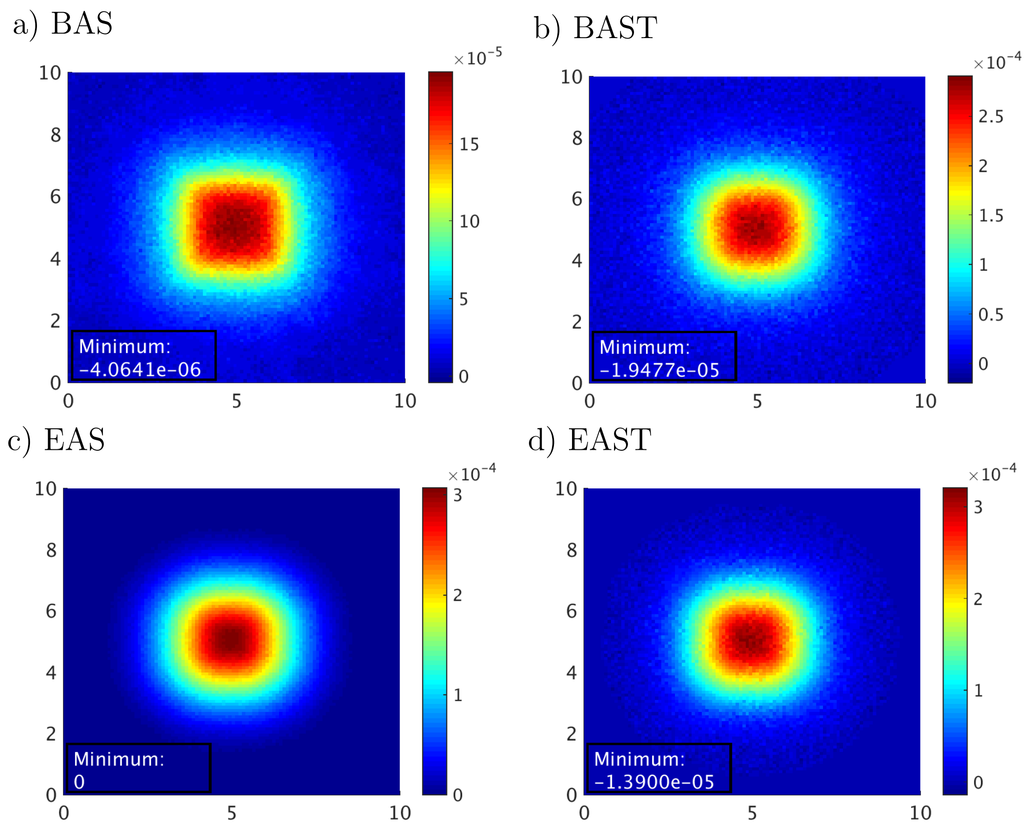


Figure 3.25: Effect of using unsuitably large ΔM (10^{-7}) values with each of the schemes with the reaction-diffusion example of §3.2.1. Compare to Figure 3.22 a). The minimum value of u in any cell in the domain is displayed to demonstrate the overdrawing effect.

3.2.2 Reaction Advection Diffusion Example

In this example we use the same grid, velocity field, diffusivity field, initial condition, and boundary conditions as in §3.1.2. We again add a Langmuir type reaction term,

$$r(c, \mathbf{x}) = -\epsilon(\mathbf{x})\frac{c}{1+c},$$

where we have introduced a spatial dependence by the factor ϵ;

$$\epsilon(\mathbf{x}) = \frac{0.02}{D(\mathbf{x})^2}.$$

In this way the reaction occurs much slower in the fracture than the rest of the domain. Physically this represents the solute species being much less likely to adsorb to the walls of the porous medium, and be lost, within the fracture. The final time is $T = 2.4$.

In Figures 3.26, 3.27, 3.28 we show the final state of the system, the convergence and efficiency results, and the parameter relations for the schemes. The layout is the same as with all the previous examples. The conclusions are largely the same.

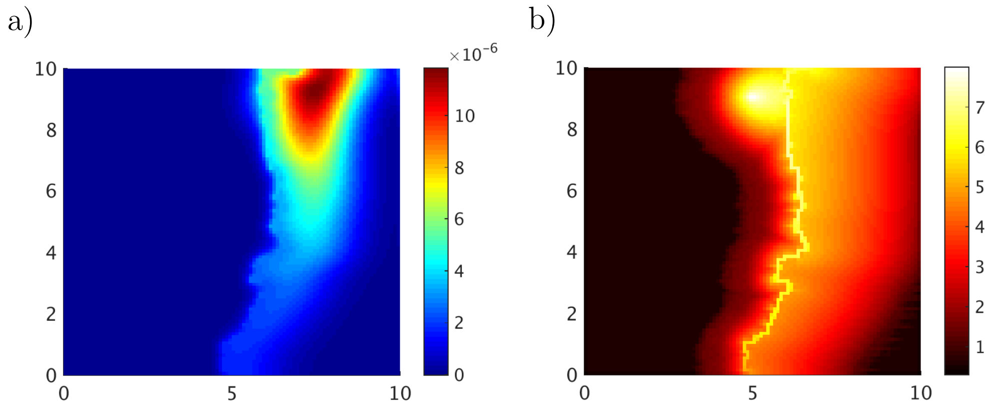


Figure 3.26: Two dimensional reaction-diffusion-advection system. Left: solution with BAST with $\Delta M = 10^{-9}$. Right: heat map of logarithm of events for each cell.

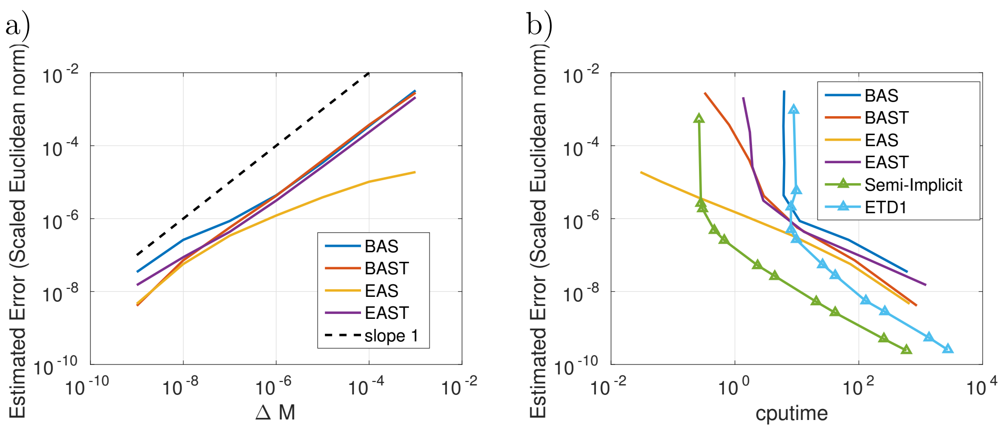


Figure 3.27: Results for the reaction-diffusion-advection system in Figure 3.26, concerning error convergence in ΔM and efficiency. For discussion see text; §3.2.2.

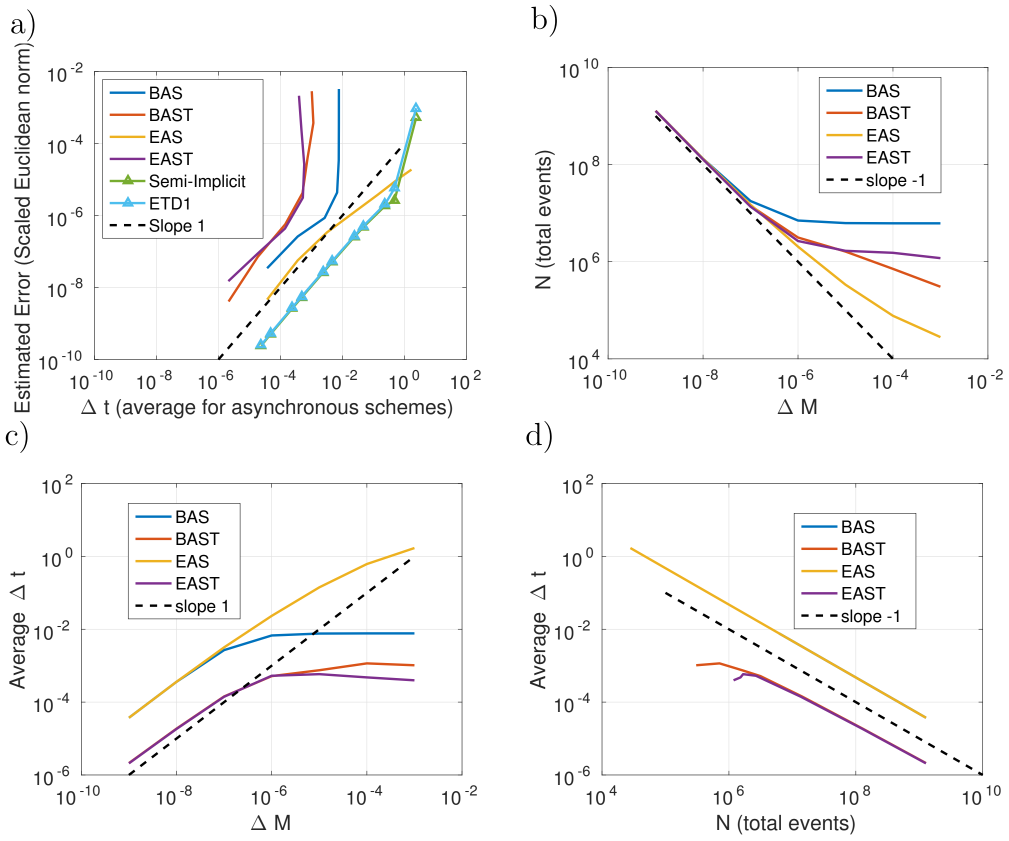


Figure 3.28: Results for the reaction-diffusion-advection system in Figure 3.22, concerning relationships between parameters. For discussion see text; §3.2.2.

3.3 Conclusions from Numerical Tests

From these tests we see that the new asynchronous schemes converge in error, for fixed grids, as $\Delta M \to 0$. The order of convergence is indicated to be roughly $O(\Delta M)$ according to the results. There also seems to be a regime of sufficiently low ΔM in which parameter relationships emerge. These relationships are, Error = $O(\Delta t \text{ (average)})$, $N = O(\Delta M^{-1})$, Δt (average) $= O(\Delta M)$, and Δt (average) $= O(N^{-1})$. These relationships suggest further theoretical investigation, which is started in the next chapter. The convergence results also indicate the basic viability of the face based asynchronous schemes, and the fact that the same conclusions can be drawn for BAS and three different modified schemes, implies the existence of a large space of possible viable schemes of this class, some of which may exhibit interesting or useful properties.

We note that the relation Δt (average) $= O(N^{-1})$ can be explained a priori, following from the fact that every face will have timesteps summing to T, and that N is the sum of the number of events on each face. The way that Δt (average) is calculated for the non-tracking schemes BAS and EAS is then equivalent to Δt (average) $= \frac{TK}{N}$, where K is the number of faces. This must be modified for the tracking schemes but a similar a priori relation can certainly be found. We note further that then the relationship Δt (average) $= O(\Delta M)$ is equivalent to $O(N^{-1}) = O(\Delta M)$, so that these two observed relations are equivalent. Also, the relations Δt (average) $= O(\Delta M)$ and Error $= O(\Delta M)$ together imply Error $= O(\Delta t \text{ (average)})$.

This leaves the observed relations Error $= O(\Delta M)$ and $N = O(\Delta M^{-1})$ as independent and needing explanation, and highly interesting. The relation $N = O(\Delta M^{-1})$ may seem to follow naturally from the construction of the schemes, but showing this rigorously while taking account of the asynchronous nature of the schemes is highly difficult.

Our current implementations of these four face based asynchronous schemes perform poorly, with respect to efficiency, compared to classical schemes. This may be because the face based asynchronous approach is inherently inefficient, or perhaps an alternative implementation may provide greater efficiency. While care was taken in optimizing our codes, they remain essentially demonstration pieces. For obvious reasons the implementation of DES based schemes for continuous systems such as these is not as well understood as for classical schemes. Our schemes are also essentially quite primitive - BAS is the simplest scheme of this type we could develop based on mass conserving exchanges based on flux; the other schemes add complexity to this for improved performance. For this current work, it is interesting that even these primitive asynchronous schemes appear to converge with a definite order, and on domains with dimension greater than one.

Chapter 4

Face Based Asynchronous Schemes - Analysis

4.1 Analysis of the New Schemes

We present some analytical results for the new schemes, which are first steps towards convergence results explaining the behaviour observed in Chapter 3. First some useful notation.

Let the set of all face connection matrices be

$$\mathcal{L} = \{L_1, \dots, L_K\}. \tag{4.1}$$

Further, let the events be indexed with $i = 1, 2, \dots, n$ where n is the total number of events so far. We will use N to denote the total number of events for a finished solve, that is, one where every face time t_k has value T. In each event i, there will be a corresponding face with index $k_i \in \mathcal{F}$ which is the face that has the event. We allow that k_i and N change as the mass unit ΔM changes and produces a different path of events. Let the event timestep be δt_i, which is the same as the face timestep Δt_{k_i} defined by (2.23),

$$\delta t_i = \Delta t_{k_i} = \frac{\sqrt{2}\Delta M}{||L_{k_i}\mathbf{m}_{i-1}||},$$

where k_i is such that

$$t_{k_i} + \Delta t_{k_i} = \min_{k \in \mathcal{F}} \{t_k + \Delta t_k\}.$$

4.2 Convergence of BAS with One Internal Face

We prove convergence of our scheme for the simplified case of two cells and one internal face, see Figure 2.2 (b). Here there are only two mass (or equivalently concentration) values. Since there is only one face, there is only one connection matrix, which we call L (in this simple system the finite volume matrix from (2.9), and the general and local connection matrices $\tilde{L}_1$, L_1, are all the same). Let the direction vector associated with the connection matrix be $\hat{\mathbf{z}}$, and let the normalised version of this be $\mathbf{z} = \frac{1}{\sqrt{2}}\hat{\mathbf{z}}$. Let the vector of the mass values be $\mathbf{m}$, and let $\mathbf{m}_n$ be this vector after the nth event. Let the initial condition of the system be $\mathbf{m}_0$. First we consider the nature of the true solution. The true solution is $\mathbf{m}(t) = e^{tL}\mathbf{m}_0$ which is the solution of the ODE

$$\frac{d\mathbf{m}(t)}{dt} = L\mathbf{m}(t).$$

The matrix exponential solution may be expressed in terms of $\mathbf{z}$ using the following arguments. The true solution is,

$$\mathbf{m}(t) = e^{tL}\mathbf{m}_0 = \mathbf{m}_0 + (e^{tL} - I)\mathbf{m}_0.$$

A line of argument similar to Lemma 2.4.1 follows. We have,

$$(e^{tL} - I)\mathbf{m}_0 = \sum_{i=1}^{\infty} \frac{(tL)^i}{i!}\mathbf{m}_0.$$

Using the properties of the connection matrix, we have that $L\mathbf{m}_0 = ||L\mathbf{m}_0||\mathbf{z}$. Thus,

$$(e^{tL} - I)\mathbf{m}_0 = t||L\mathbf{m}_0|| \sum_{i=1}^{\infty} \frac{(tL)^{i-1}}{i!}\mathbf{z}.$$

From this it follows that

$$(e^{tL} - I)\mathbf{m}_0 = t||L\mathbf{m}_0||\varphi_1(t\lambda)\mathbf{z}, \tag{4.2}$$

where λ is the eigenvalue of L satisfying $L\mathbf{z} = \lambda\mathbf{z}$, from (2.19). Then, letting,

$$\alpha(t) \equiv t||L\mathbf{m}_0||\varphi_1(t\lambda), \tag{4.3}$$

we have,

$$\mathbf{m}(t) = \mathbf{m}_0 + \alpha(t)\mathbf{z}.$$

We see that the true solution is confined to a single direction, $\mathbf{z}$, which is a result of mass conservation.

Now we consider the asynchronous scheme. From (2.17), and using the notation of the event timestep, the mass of the system after n events may be expressed as

$$\mathbf{m}_n = \prod_{i=1}^{n}(I + \delta t_i L)\mathbf{m}_0.$$

The mass is updated in an event as

$$\mathbf{m}_{n+1} = \mathbf{m}_n + \frac{\sqrt{2}\Delta M}{||L\mathbf{m}_n||}L\mathbf{m}_n = \mathbf{m}_n + \sqrt{2}\Delta M\mathbf{z}, \tag{4.4}$$

where we have used (2.23). We have assumed that the direction of mass transfer is consistent across the whole solve, i.e., that there is never a $-\sqrt{2}\Delta M\mathbf{z}$ event. This is justified since in this simple two cell system, we expect equilibrium to be reached before the net flow changes direction. This may not be the case in the context of discrete events, though it should be true as $\Delta M \to 0$. From (4.4), the state after n events of the scheme is,

$$\mathbf{m}_n = \mathbf{m}_0 + \sqrt{2}\Delta M n\mathbf{z}. \tag{4.5}$$

We may use (4.5) to re-express (2.23) as,

$$\Delta t_{1,i} = \frac{\sqrt{2}\Delta M}{||L\mathbf{m}_0|| + \sqrt{2}\Delta M i\lambda}, \tag{4.6}$$

having multiplied through by L and using the fact that the norm of $\mathbf{z}$ is unity. We note that we know λ which we know to be negative from §2.4.

We see from (4.5) that the asynchronous scheme only progresses in a fixed direction, $\mathbf{z}$, like the true solution.

Next we need to find the stopping condition for the scheme. Consider some event number n, with corresponding system time t_n. The time is given by summing (4.6),

$$t_{1,n} = \sum_{i=0}^{n-1} \frac{\sqrt{2}\Delta M}{||L\mathbf{m}_0|| + i\sqrt{2}\Delta M\lambda}. \tag{4.7}$$

We can interpret (4.7) as a simple rectangular quadrature approximation to an integral of the function $f(x) = \frac{\sqrt{2}}{||L\mathbf{m}_0||+x\sqrt{2}\lambda}$, with quadrature points ΔM_i, $i = 1, 2, \ldots, n-1$ (see for example Chapter 4 of [90]). Since this function is concave the quadrature is an overestimation. The true integral is well known, and we have,

$$t_{1,n} = \frac{1}{\lambda} \ln \left(\frac{||L\mathbf{m}_0|| + n\Delta M\sqrt{2}\lambda}{||L\mathbf{m}_0||} \right) + E_Q(\Delta M, n).$$

Note that both λ and the logarithm are negative. The quadrature error $E_Q(\Delta M, n)$ is positive and assumed to be $O(\Delta M)$. From (4.7) we have,

$$t_{1,n} \geq \frac{1}{\lambda} \ln \left(\frac{||L\mathbf{m}_0|| + n\Delta M\sqrt{2}\lambda}{||L\mathbf{m}_0||} \right),$$

which re-arranges to,

$$||L\mathbf{m}_0|| \frac{\left(e^{t_{1,n}\lambda} - 1\right)}{\sqrt{2}\lambda\Delta M} \geq n,$$

and from (4.3) we have

$$\frac{\alpha(t_{1,n})}{\sqrt{2}\Delta M} \geq n,$$

so that

$$\alpha(t_n) - n\sqrt{2}\Delta M \geq 0. \tag{4.8}$$

We also have that $t_{1,n+1} \geq t_{1,n}$, i.e.,

$$\frac{1}{\lambda} \ln \left(\frac{||L\mathbf{m}_0|| + (n+1)\Delta M\sqrt{2}\lambda}{||L\mathbf{m}_0||} \right) + E_Q(\Delta M, n+1) \geq t_{1,n},$$

which by re-arrangements similar to those which lead to (4.8), gives

$$n + 1 \geq ||L\mathbf{m}_0|| \frac{(e^{\lambda(t_{1,n} - E_Q(\Delta M, n+1))} - 1)}{\sqrt{2}\Delta M \lambda}. \tag{4.9}$$

In order to relate (4.9) to $\alpha(t_{1,n})$ in (4.3) we compare the exponential terms in each. Since $E_Q(\Delta M, N+1)$ is positive and λ negative, we have $\lambda(t_{1,n} - E_Q(\Delta M, n+1)) > \lambda t_{1,n}$ and $e^{\lambda(t_{1,n} - E_Q(\Delta M, N+1))} > e^{\lambda t_{1,n}}$, so that (4.9) gives us

$$n \geq \frac{\alpha(t_{1,n})}{\sqrt{2}\Delta M},$$

and combining this with (4.8) gives

$$\sqrt{2}\Delta M \geq \alpha(t_{1,n}) - n\sqrt{2}\Delta M \geq 0,$$

thus

$$|\alpha(t_{1,n}) - n\sqrt{2}\Delta M| \leq \sqrt{2}\Delta M.$$

Thus we have proven that in the one face scenario that the scheme at time $t_{1,n}$ stays within a bounded distance of the true solution at time $t_{1,n}$, and that that distance decreases with ΔM. It would be interesting in future work to investigate if this is true in general.

Now, using $n = N$, the number of events at which the scheme reaches the target time T, we can express the error of the scheme as

$$E = ||\mathbf{m}(t) - \mathbf{m}_N|| = ||\mathbf{m}_0 + \alpha(T)\mathbf{z} - \mathbf{m}_0 - N\sqrt{2}\Delta M\mathbf{z} - M'\mathbf{z}||, \tag{4.10}$$

where M' is the amount of mass transferred in the final synchronisation step, satisfying $M' \leq \Delta M$. Equation (4.10) simplifies to

$$E = |\alpha(T) - N\sqrt{2}\Delta M - M'| \leq |M'| + |\alpha(T) - N\sqrt{2}\Delta M| \leq (1 + \sqrt{2})\Delta M,$$

which is first order convergence in ΔM.

4.3 Towards a General Convergence Result for BAS

Here we present a framework for extending the analysis of BAS (§4.2) to an arbitrary number of faces. We use connection matrices again; see §2.4. In particular we use the fact that the action of a connection matrix L_i on any vector $\mathbf{y}$ produces a scalar, determined by $\mathbf{y}$, multiplying a direction vector $\hat{z}_i$. For two connection matrices L_i and L_j, with corresponding direction vectors $\hat{z}_i$, $\hat{z}_j$, define $c_{i,j}$ to be such that

$$L_i \hat{z}_j = c_{i,j} \hat{z}_i,$$

and vice versa for $c_{j,i}$. The eigenvalue of L_i from (2.19) is then $\lambda_i = c_{i,i}$. Define the matrix C as having the entries $(C)_{i,j} = c_{i,j}$. Let L be the sum of some connection matrices, $L = \sum_k^K L_k$. Consider the action of L on some vector $\mathbf{m_0}$,

$$L\mathbf{m}_0 = \sum_k^K f_k \hat{z}_k,$$

where we have defined f_k by $L_k \mathbf{m}_0 = f_k \hat{z}_k$, using (2.18). Let $\hat{Z}$ be the matrix whose kth column is $\hat{z}_k$, and let $\mathbf{f}_0$ be the vector whose kth entry is f_k, then

$$L\mathbf{m}_0 = \hat{Z}\mathbf{f}_0.$$

Now consider,

$$L_i L\mathbf{m}_0 = \sum_k^K f_k c_{i,k} \hat{z}_i = \hat{z}_i (c_{i,1}, \ldots, c_{i,K}) \mathbf{f}_0.$$

Then

$$L^2 \mathbf{m}_0 = \sum_i^K \hat{z}_i (c_{i,1}, \ldots, c_{i,K}) \mathbf{f}_0 = \hat{Z} C \mathbf{f}_0,$$

where the sum is over the action of each L_i on the $L\mathbf{u}_0$. Indeed, for any arbitrary $\mathbf{y}$,

$$L\hat{Z}\mathbf{y} = \sum_i^K \hat{z}_i (c_{i,1}, \ldots, c_{i,K}) \mathbf{y} = \hat{Z} C \mathbf{y}.$$

From this we have

$$L^n \mathbf{m}_0 = \hat{Z} C^{n-1} \mathbf{f}_0.$$

We can use this to generalise the re-expression of $e^{tL}\mathbf{m}_0$ given in §4.2; (4.2). For,

$$e^{tL}\mathbf{m}_0 = \mathbf{m}_0 + \sum_{i=1}^{\infty} \frac{(tL)^i}{i!} \mathbf{m}_0,$$

which is

$$e^{tL}\mathbf{m}_0 = \mathbf{m}_0 + \hat{Z} \sum_{i=1}^{\infty} \frac{t^i C^{i-1}}{i!} \mathbf{f}_0,$$

and again we use the definition of φ_1,

$$e^{tL}\mathbf{m}_0 = \mathbf{m}_0 + t\hat{Z}\varphi_1(tC)\mathbf{f}_0.$$

For the scheme BAS, after some total number of events n, let n_k be the number of events experienced by face k. Let $\mathbf{n}$ be the vector whose kth entry is n_k. Then the state of BAS can be expressed as,

$$\mathbf{m}_n = \mathbf{m}_0 + \Delta M \hat{Z} \mathbf{n},$$

analogously to (4.5). Note that we are again assuming that the direction of mass transfer is consistent across the whole solve, which may not be completely justified in all cases. To prove convergence we would need to show that

$$\Delta M \hat{Z} \mathbf{n} \to t\hat{Z}\varphi_1(tC)\mathbf{f}_0 \tag{4.11}$$

as $\Delta M \to 0$, when $\mathbf{n}$ evolves according to the rules of the scheme (the face with the lowest update time being the one updated during an event, and so on). Alternatively we may attempt to show something like

$$\Delta M C \mathbf{n} + \mathbf{f}_0 = e^{tC}\mathbf{f}_0,$$

in the limit $\Delta M \to 0$, which could then rearrange to yield (4.11). It is interesting to note that the left hand side is a vector of the fluxes of each face. To see this consider the action of a L_k on $\mathbf{m}_n$,

$$L_k \mathbf{m}_n = \Delta M (c_{k,1}, \dots c_{k,K}) \mathbf{n} \hat{z}_k.$$

The vector on the right hand side has only two nonzero entries, the positive and negative of the flux across face k. Since $\hat{z}_k$ has only nonzero entries -1 and 1, the flux across face k is the coefficient of the right hand side, $\Delta M (c_{k,1}, \dots c_{k,K}) \mathbf{n}$. Thus we have,

$$\text{flux across each face} = \Delta M C \mathbf{n} + \mathbf{f}_0.$$

We now pursue a heuristic argument towards showing (4.11). First we approximate $\Delta M \mathbf{n}$ by a continuous variable, $\mathbf{x} = \Delta M \mathbf{n}$. We assume that in the limit $\Delta M \to 0$ this is justifiable, as ΔM becomes so small that integer multiples of ΔM become effectively continuous. We wish to argue that

$$\frac{d\mathbf{x}}{dt} = C\mathbf{x} + \mathbf{f}_0. \tag{4.12}$$

Given that $\mathbf{x}(0) = 0$, the solution to this is

$$\mathbf{x}(t) = t\varphi_1(tC)\mathbf{f}_0,$$

from which (4.11) would follow. Note that the right hand side of (4.12) is, again, the flux. We must interpret the t in the derivative as the system time, i.e. the time of the face which has most recently updated. Since $\mathbf{m}_n = \mathbf{m}_0 + \hat{Z}\mathbf{x}$, $\mathbf{x}$ is the vector of displacements along each direction vector $\hat{z}$, from the starting point of $\mathbf{m}_0$. Thus (4.12), if true, implies that the rate of change of the solution in the direction of a $\hat{z}$ associated with a face k, with respect to the system time, is equal to the flux across face k.

We can ask if anything in the construction of the scheme indicates the potential for this behaviour. Interestingly, we can examine (2.10), the equation for determining update time for a face. We restate it here for convenience,

$$\hat{t}_k = t_k + \frac{\Delta M}{f_k},$$

where f_k is the flux across the face k, $\hat{t}_k$ is the update time of the face, and t_k is the time of the face. If the face is chosen for an event (by having a lowest update time), then it updates with timestep $\Delta t_k = \hat{t}_k - t_k$. We may re-arrange to $\frac{\Delta M}{\Delta t} = f_k$. Heuristically, in the limit $\Delta M \to 0$ we may replace the fraction with $\frac{dx_k}{dt}$, and write

$$\frac{dx_k}{dt} = f_k.$$

A vector of these values would give (4.12). There is however the need to bridge the gap between the asynchronous nature of the algorithm and the synchronous nature of the ODE (4.12). For this we would have to assume or demonstrate that in the limit $\Delta M \to 0$, the individual face times t_k tend towards being equal or arbitrarily close to the entire system time t. This is a potential subject of further work.

4.4 Assumptions of Parameter Relations

Identifying relationships between the parameters such as ΔM, N, and individual event timesteps δt_i would be essential for a full analysis of the schemes presented here and of DES based Asynchronous schemes in general. Some relationships are heavily implied by our results in §3.1; others suggest themselves from the form of the scheme, but there are subtleties to consider. We first state three assumptions that we base on the form of the schemes, which we will use in the next section in a sketch proof of convergence for EAS, and then discuss these assumptions.

Assumption 4.4.1. *There exists a positive real d_1 such that*

$$\delta t_n = O\left(\Delta M^{d_1}\right),\ n = 1, \ldots, N.$$

That is, for an event number n, if ΔM is small enough that event n occurs (i.e., that $n < N$, see Assumption 4.4.2), the event timestep decreases as ΔM decreases, for ΔM sufficiently small. The source for this is the observed numerical results (plot c) in all of Figures 3.10, 3.16 and 3.20), and (2.23), which implies a proportionality between the timestep Δt and ΔM. However, we must observe that the denominator in the right hand side of (2.23) may have some dependence on ΔM since the ordering of events may depend on ΔM. This means that $||L_k\mathbf{m}||$ is different for different ΔM values for given n. Ideally we would like to identify a C such that $\frac{1}{||L_k\mathbf{m}||} \leq C$, or equivalently

$$||L_k\mathbf{m}|| \geq 1/C,$$

which is the same as claiming that, for any event n and ΔM value, the flux across the face k *chosen for an event* is bounded below by some constant $1/C$. A face k is chosen for an event due to a combination of low t_k and high flux; this might make the existence of such a bound seem reasonable, but this is not conclusive for all cases. Our numerical results that support the assumption indicate that the exponent has a value around $d_1 = 1$.

Assumption 4.4.2. *The number of events N increases as ΔM decreases, and there exists a positive real number d_2 such that*

$$N = O\left(\Delta M^{-d_2}\right).$$

As well as being implied strongly by our numerical results (plot b) in all of Figures 3.10, 3.16 and 3.20), this assumption should follow from the construction of the algorithms as long as the initial data is such that there is some activity in the domain (i.e., a nonzero flux). A first event must then occur, with timestep directly proportional to ΔM by (2.23), that is if k is the face with the greatest flux initially, the initial timestep will be $\Delta t = \frac{\sqrt{2}}{||L_k\mathbf{m}_0||}$. This is clearly proportional to ΔM as every other parameter on the right hand side is constant. There will be a ΔM sufficiently small that the initial timestep is less than the final time T, so that at least two events must occur on that face in order to bring its individual time up to T and end the solve. We can then consider values of ΔM small enough that the corresponding timesteps are small enough that arbitrarily many events on the first face, then its associated faces and so on, must occur. This follows from the proportionality of Δt and ΔM given by (2.23), again considering the caveat discussed after Assumption 4.4.1. Our numerical results imply that $d_2 = 1$.

Bringing together Assumptions 4.4.1 and 4.4.2 gives a third parameter relation.

Remark 4.4.3. *Given Assumptions 4.4.1 and 4.4.2, as N increases, the event timestep decreases; there exists a positive real d_3 such that*

$$\delta t_n = O\left(N^{-d_3}\right), \ n = 1, \ldots, N.$$

This is supported by our numerical results (plot d) in all of Figures 3.10, 3.16 and 3.20, for the average Δt, with $d_3 = 1$.

4.5 Analysis of EAS

We consider the linear ODE system (2.9), after dividing each row by V_k to produce an equation for the mass values, i.e., $\frac{d\mathbf{m}}{dt} = L\mathbf{m}$. The exact solution is,

$$\mathbf{m}(T) = \exp\left(T\sum_{k=1}^{K} L_k\right)\mathbf{m}(0), \tag{4.13}$$

where we have expressed $TL = T\sum_{k=1}^{K} L_k$, as a sum of the connection matrices. The approximation produced by EAS after n events is

$$\mathbf{m}_n = \prod_{i=1}^{n} \exp(\delta t_i L_{k_i})\mathbf{m}(0) = \exp(Z_n)\mathbf{m}(0), \tag{4.14}$$

where $L_{k_i} \in \mathcal{L}$ is the connection matrix chosen for the ith event, and Z_n is to be determined. The iterative formula for $\mathbf{m}_n$ is

$$\mathbf{m}_{n+1} = \exp(\delta t_{n+1} L_{k_{n+1}})\mathbf{m}_n = \exp(\delta t_{n+1} L_{k_{n+1}})\exp(Z_n)\mathbf{m}(0). \tag{4.15}$$

We can make use of the Baker-Campell-Hausdorff (BCH) formula (see for example, [91]), which states that, for operators A, B,

$$\exp(A)\exp(B) = \exp(C), \tag{4.16}$$

with

$$C = A + B + \frac{1}{2}[A,B] + \frac{1}{12}[A,[A,B]] - \frac{1}{12}[B,[B,A]] + \dots, \tag{4.17}$$

where the Lie bracket $[\cdot]$ is defined as $[A,B] = AB - BA\ A, B \in \mathbb{R}^{J\times J}$. We have an iterative formula for Z_n from (4.15), by taking $A = \delta t_{n+1} L_{k_{n+1}}$ and $B = Z_n$ in (4.17),

$$\begin{aligned} Z_{n+1} &= \delta t_{n+1} L_{k_{n+1}} + Z_n + \frac{1}{2}[\delta t_{n+1} L_{k_{n+1}}, Z_n] \\ &+ \frac{1}{12}[\delta t_{n+1} L_{k_{n+1}}, [\delta t_{n+1} L_{k_{n+1}}, Z_n]] - \frac{1}{12}[Z_n, [Z_n, \delta t_{n+1} L_{k_{n+1}}]] + \dots . \end{aligned} \tag{4.18}$$

An alternative expression of (4.17) is the Goldberg series [92], see also for example [93]. The Goldberg series is a double sum of words made of the operators A and B; a word here means a simple multiplicative term, for example A, B, ABA, $AABB$ are all examples of words made from A and B. A word of length i is made of i instances of the operators, for example A and B are length one, ABA of length three and $AABB$ length four. There are 2^i words of length i that can be made from operators A and B (3^i if there were three operators, and so on). For the purposes of writing a sum over the words, let $W(j, i, A, B)$ be the jth word of length i ($j = 1, \dots, 2^i$) made from A and B. For example, $W(j, 1, A, B)$ could be A or B; ABA would be one of the $W(j, 3, A, B)$; $AABB$ would be one of the $W(j, 4, A, B)$, and so on. Then, Goldberg's exponential series for C in (4.16) is

$$C = A + B + \sum_{i=2}^{\infty} \sum_{j=1}^{2^i} g(j, i, X, Y) W(j, iX, Y). \tag{4.19}$$

The Goldberg coefficients $g(j, i, X, Y)$ corresponding to each word are rational numbers, and listings and discussions of the calculations of these can be found in [94]. An advantage of EAS is that it the scheme can be written in exponential form (4.14), which lends itself to analysis using the BCH. In the spirit of analysis of symplectic operator splitting schemes [95], we can attempt to prove convergence of $\mathbf{m}_N$ to $\mathbf{m}(T)$, by proving convergence of Z_N to $TL = T\sum_{k=1}^{K} L_k$, where N is the number of events in the EAS solve (i.e., the number of events after which the scheme has brought the individual time on every face to T). We demonstrate here how such an argument may proceed.

It will be useful to express Z_n from (4.14) in a modified form of (4.19). We use words made from the event timesteps instead of operators. Let $\hat{w}(j, i, n)$ be the jth word of length i, made from elements of the set $\{\delta t_1, \ldots, \delta t_n\}$. Because the timesteps are scalars they commute, unlike the operator words that make up the Goldburg series. For example, $ABA \neq BAA$, but $\delta t_1 \delta t_2 \delta t_1$ and $\delta t_2 \delta t_1 \delta t_1$ are both equal to $\delta t_1^2 \delta t_2$. Because of this the number of possible words $\hat{w}(j, i, n)$ is given by the multiset coefficient $\binom{n+i-1}{i}$. We write

$$Z_n = \sum_{i=1}^{\infty} \sum_{j=1}^{\binom{n+i-1}{i}} \hat{g}(j, i, n) \hat{w}(j, i, n), \tag{4.20}$$

where the modified coefficients $\hat{g}(j, i, n)$ are not rational numbers but linear combinations of operator words made from elements of the set $\{L_{k_1}, \ldots, L_{k_n}\}$. An example is helpful. Consider advancing from $n = 1$ to $n = 2$. Clearly $Z_1 = \delta t_1 L_{k_1}$, and expanding (4.18) gives us

$$
\begin{aligned}
Z_2 = {} & \delta t_1 L_{k_1} + \delta t_2 L_{k_2} \\
& + \frac{1}{2}\delta t_1 \delta t_2 L_{k_1} L_{k_2} - \frac{1}{2}\delta t_1 \delta t_2 L_{k_2} L_{k_1} \\
& + \frac{1}{12}\delta t_1^2 \delta t_2 L_{k_1} L_{k_1} L_{k_2} - \frac{1}{6}\delta t_1^2 \delta t_2 L_{k_1} L_{k_2} L_{k_1} + \frac{1}{12}\delta t_1^2 \delta t_2 L_{k_2} L_{k_1} L_{k_1} \\
& - \frac{1}{12}\delta t_1 \delta t_2^2 L_{k_2} L_{k_2} L_{k_1} + \frac{1}{6}\delta t_1 \delta t_2^2 L_{k_2} L_{k_1} L_{k_2} - \frac{1}{12}\delta t_1 \delta t_2^2 L_{k_1} L_{k_2} L_{k_2} \\
& + \dots
\end{aligned}
$$

Collecting the timestep words this is

$$
\begin{aligned}
Z_2 = {} & \delta t_1 L_{k_1} + \delta t_2 L_{k_2} \\
& + \delta t_1 \delta t_2 \left(\frac{1}{2} L_{k_1} L_{k_2} - \frac{1}{2} L_{k_2} L_{k_1} \right) \\
& + \delta t_1^2 \delta t_2 \left(\frac{1}{12} L_{k_1} L_{k_1} L_{k_2} - \frac{1}{6} L_{k_1} L_{k_2} L_{k_1} + \frac{1}{12} L_{k_2} L_{k_1} L_{k_1} \right) \\
& + \delta t_1 \delta t_2^2 \left(-\frac{1}{12} L_{k_2} L_{k_2} L_{k_1} + \frac{1}{6} L_{k_2} L_{k_1} L_{k_2} - \frac{1}{12} L_{k_1} L_{k_2} L_{k_2} \right) \\
& + \dots
\end{aligned}
$$

In the form of (4.20) this is,

$$
Z_2 = \sum_{i=1}^{\infty} \sum_{j}^{\binom{2-i-1}{i}} \hat{w}(j,i,2)\hat{g}(j,i,2).
$$

Of the three possible length two words, only $\delta t_1 \delta t_2$ has a nonzero $\hat{g}$ coefficient, which is $\left(\frac{1}{2} L_{k_1} L_{k_2} - \frac{1}{2} L_{k_2} L_{k_1}\right)$. Of the four possible length three words, only $\delta t_1^2 \delta t_2$ and $\delta t_1 \delta t_2^2$ have nonzero $\hat{g}$; the $\hat{g}$ for $\delta t_1^2 \delta t_2$ is $\left(\frac{1}{12} L_{k_1} L_{k_1} L_{k_2} - \frac{1}{6} L_{k_1} L_{k_2} L_{k_1} + \frac{1}{12} L_{k_2} L_{k_1} L_{k_1}\right)$, and so on.

We consider the length one words in (4.20). Let the sum of all the length one words in Z_n be S_n. From (4.18) it is clear that $S_{n+1} = S_n + \delta t_{n+1} L_{k_{n+1}}$, and since $S_1 = Z_1 = \delta t_1 L_{k_1}$, each timestep word in S_n has a coefficient $L_k \in \mathcal{L}$, with $\mathcal{L}$ given by (4.1). Thus we can write

$$
S_n = \sum_{k=1}^{K} t_k L_k, \tag{4.21}
$$

where t_k is the sum of the δt for every event which has used L_k as the event operator, i.e., every event on face k. Then, t_k is nothing but the face time defined in §2.2. The algorithm guarantees that at event N, $t_k = T$ for every face. This leads to

$$Z_N = T\sum_{k=1}^{K} L_k + \sum_{i=2}^{\infty} \sum_{j=1}^{\binom{N+i-1}{i}} \hat{g}(j,i,N)\hat{w}(j,i,N). \tag{4.22}$$

We can write

$$Z_n = S_n + R_n.$$

Comparing (4.22) to (4.13) leads to

Conjecture 4.5.1.

$$R_N = \sum_{i=2}^{\infty} \sum_{j=1}^{\binom{N+i-1}{i}} \hat{g}(j,i,N)\hat{w}(j,i,N) \to 0$$

as $N \to \infty$.

To prove Conjecture 4.5.1 we might invoke Remark 4.4.3 and assume that a length i timestep word is $O(N^{-d_3 i})$. Then as $N \to \infty$,

$$R_N \to \sum_{i=2}^{\infty} \sum_{j=1}^{\binom{N+i-1}{i}} \hat{g}(j,i,N)O(N^{-d_3 i}) = \sum_{i=2}^{\infty} C(i,N)O(N^{-d_3 i}),$$

where $C(i,N)$ is some bound on the sum of the $\hat{g}(j,i,N)$ for a given i. Proving a desirable bound $C(i,N)$ would require two results. First, we must ensure that no $\hat{g}(j,i,N)$ becomes unboundedly large (in some norm). Second, we must ensure that the number of nonzero $\hat{g}(j,i,N)$ for a given i is sufficiently bounded.

Concerning the first required result, the $\hat{g}(j,i,N)$ are linear combinations of operator words and these words have the potential to become arbitrarily long as $N \to \infty$. This may not be pathological if we consider the actions of connection matrices on each other. Consider the product $L_k L_{k'}$. Unless L_k and L_k are associated faces, the product is an empty matrix. Typically the size of a set of associated faces is much smaller than the size of the set $\mathcal{L}$ of all connection matrices. Thus as the length of an operator word becomes arbitrarily large, the chance of it including a null pairing of connection matrices like this may become extremely high or certain.

For the second result, we can immediately place an upper bound on the number of $\hat{g}(j,i,N)$ as $\binom{N+i-1}{i}$, which is $O(N^i)$ as $N \to \infty$. Assuming we have the first required result, we would then have

$$R_N = \sum_{i=2}^{\infty} O(N^{(1-d_3)i}), \tag{4.23}$$

which proves Conjecture 4.5.1 if $d_3 > 1$. With Assumption 4.4.2, this becomes $O(\Delta M^{-2d_2(1-d_3)})$ and we have a very rough convergence result that does not consider the ordering of events, or the initial condition. This outlines how a convergence result for EAS may be formulated by taking advantage of the ability to write that scheme as a product of exponentials. A complete result will have to take into account the initial condition and how this affects event ordering, how the scheme handles event ordering in general, and unique properties of the connection matrices.

As remarked in §4.4, d_3 seems to be most likely 1 and thus too small to guarantee convergence by (4.23). To move towards remedying this, we can make more detailed observations on the nature of $C(i,n)$. For example, we can observe that since the Golding coefficients are zero for operator words of the form X^i, Y^i, there will consequently be no timestep words of the form $\delta t_1^i, \ldots \delta t_n^i$ in Z_n, so that $C(i,n)$ is bounded above by $\binom{n+i-1}{i} - n$. This is however still $O(n^i)$ as $N \to \infty$. Since we have observed empirically that EAS does converge as $\Delta M \to 0$, it seems likely that the true upper bound on $C(i,n)$ really is still lower. Deeper observation of the properties of the L_k are required.

We will present an illustrative argument. We consider going from n to $n+1$ events, and examine only the second order terms (i.e., timestep words of length 2). The length two words in Z_{n+1} are all of length two words in Z_n, plus the words of length two produced by the action of the term $\frac{1}{2}\left[\delta t_{n+1} L_{k_{n+1}}, Z_n\right]$ (from (4.18)). The length two words from this will be produced from the length one words in Z_n, since this term takes timestep words and adds δt_{n+1} to them.

There are exactly n length one words in Z_n, as a consequence of the construction of the scheme. Thus in moving from event n to $n+1$, at most n additional words of length two are created, or, $C(i, n+1) \leq C(i, n) + n$. This bound must actually be lower for $C(i, n)$ to be less than order n^2.

Assume that n is sufficiently large, and the path of the events sufficiently intensive, that every single connection matrix $L_k \in \mathcal{L}$ has been used in an event, and that $n > K$. That is, every $t_k > 0$ for every t_k appearing in (4.21). Now there are only a small set of $L_k \in \mathcal{L}$ which do not have the property $L_{k_{n+1}} L_k = 0$ (the set of associated faces for face k_{n+1} is less than the set of all faces), as discussed earlier. These are the only terms which are not set to zero by the term $\frac{1}{2}\left[\delta t_{n+1} L_{k_{n+1}}, Z_n\right]$ which produces new length two words. Thus, given the assumption that every connection matrix has had at least one event, then moving from n to $n+1$ events will increase the number of length two words by *less* than n, due to the properties of the connection matrices.

This is an example of how the special properties of the operators used to express the scheme can improve the convergence results. More are no doubt possible. It is also possible to extend the analysis to schemes which are not EAS. Given that EAS converges, we would have to allow that each step of the scheme does not produce the action of another exponential $e^{\delta t_{n+1} L_{k_{n+1}}}$, but rather an approximation of it, $e^{\delta t_{n+1} L_{k_{n+1}}} + E_{n+1}(\delta t_{n+1})$ for example. The extra error would then accumulate, and we would have to show that this is bounded by ΔM again.

4.6 Extensions and Conclusions

In this chapter we have introduced new DES based Asynchronous schemes for advection-diffusion and advection-diffusion-reaction systems. Numerical tests in dimensions up to three have revealed convergence of these schemes to corresponding classical solutions as a control parameter ΔM decreases to zero, as well as indicating other interesting relationships between parameters. Some framework has been laid down for progress towards rigorous analysis of these schemes.

Our schemes include the most basic possible example of the class, along with others incorporating various modifications. All of these schemes have been successful in our numerical experiments; this indicates that there exist a wide variety of potentially effective DES based Asynchronous schemes to be discovered. Finding the most effective modifications and implementations for these schemes is essential for achieving their potential. In particular we have found that the mass-tracking concept to be especially effective for improving the face based schemes here. We have also demonstrated that the flux capacitor concept of [1] can be applied to face based schemes, and has great potential for the design of fast efficient Asynchronous schemes. In general our schemes as implemented here do not outperform classical schemes. Our codes remain essentially demonstration pieces.

We should specifically note the question of extending these face based Asynchrounous schemes to nonuniform, non Cartesian grids. These methods are based on an underlying Finite Volume discretisation, and it is well known that for such a discretisation on a nonuniform grid to be consistent spatially (i.e., converge to the correct PDE as the grid size decreases to zero), the spatial derivative approximations across a face must include data from more cells than the two immediately adjacent to the face (see §1.3). For our purposes this means that the set of associated faces, $\tilde{\mathcal{F}}_k$ of some face k must be larger, in the case of a nonuniform grid; to include the faces of all cells which contribute data to the spatial derivative approximation across k. The algorithm would proceed as usual, with a different $\tilde{\mathcal{F}}_k$ for each face as well as a different equation for the flux, e.g. (2.7) may be replaced with an approximation from a MPFA scheme (see e.g. [19]). Such an implementation, and an investigation into possible ways of improving efficiency in such a case, remain subjects of potential further research.

Another possibility to expand the schemes is to attempt to add stochastic forcing. The aim would be to have a stochastic scheme which approximates a version of the PDE (2.28) with a stochastic forcing term. One possible way to do this is to add some kind of stochastic forcing to either (2.12) or (2.10).

Chapter 5

Krylov Subspace Recycling

5.1 Introduction

In this chapter we leave Asynchronous methods and consider Exponential Integrator schemes. We consider the numerical integration of a large system of semilinear ODEs of the form

$$\begin{aligned} \frac{du}{dt} &= Lu + F(t, u(t)) \\ u(0) &= u_0, \quad t \in [0, \infty) \end{aligned} \tag{5.1}$$

with $u, F(t, u(t)) \in \mathbb{R}^N$ and $L \in \mathbb{R}^{N\times N}$ a matrix. We are concerned with the approximation of (5.1) by Exponential Integrators (§1.4). We recall the scheme ETD1 can be written as

$$u^{etd}_{n+1} = u^{etd}_n + \Delta t \varphi_1(\Delta t L)\left(Lu_n + F^{etd}_n\right). \tag{5.2}$$

where $u^{etd}_n \approx u(t_n)$ at discrete times t_n, and

$$F^{etd}_n \equiv F(t_n, u^{etd}_n).$$

It is useful to introduce the additional notation

$$g^{etd}_n \equiv Lu^{etd}_n + F(t_n, u^{etd}_n).$$

This allows us to write (5.2) as

$$u_{n+1}^{etd} = u_n^{etd} + \Delta t \varphi_1(\Delta t L) g_n^{etd},$$

which will be useful later.

Recalling §1.4.1, the function φ_1 is part of a family of matrix exponential functions defined by

$$\varphi_0(z) = e^z,\ \varphi_1(z) = z^{-1}\left(e^z - I\right),$$

and in general

$$\varphi_{k+1}(z) = z^{-1}\left(\varphi_k - \frac{I}{k!}\right), \tag{5.3}$$

where I is the identity matrix, which appear in all exponential integrator schemes; see [55]. In particular we use φ_1, and for brevity we will later use the following notation,

$$p_\tau \equiv \tau \varphi_1(\tau L). \tag{5.4}$$

We can then re-write (5.2) as

$$u_{n+1}^{etd} = u_n^{etd} + p_{\Delta t}\left(L u_n + F_n\right). \tag{5.5}$$

We consider the Krylov projection method for approximating terms like $\varphi_1(\Delta t L) g_n$ in (5.2). In the Krylov method, this term is approximated on a Krylov subspace defined by the vector g_n and the matrix L. Typically the subspace is recomputed, in the form of a matrix of basis vectors V_m, every time the solution vector, u_n in (5.2), is updated (and thus also g_n). This is done using a call to the Arnoldi algorithm [53], and is often the most expensive part of each step. It is possible to 'recycle' this matrix at least once, as demonstrated in [96]. Here we investigate this possibility further.

We examine the effect of splitting the single step of (5.2) of length Δt in to S substeps of length $\delta t = \frac{\Delta t}{S}$, through which the Krylov subspace and matrices are recycled. By deriving expressions for the local error, we show that the scheme remains locally second order for any number S of substeps, and that the leading term of the local error decreases. We then examine the possibility of using the extra information from the substeps to form a corrector to increase the overall order of the scheme.

The chapter is arranged as follows. In Section 2 we describe the Krylov subspace projection method for approximating the action of $\varphi-$functions on vectors. In section 3 we describe the concept of recycling the Krylov subspace across substeps in order to increase the accuracy of the scheme, and show that the leading term of the local error of the scheme decreases as the number of substeps uses increases. We then prove a lemma to express the local error expression at arbitrary order. With this information about the local error expansion, and the extra information from the substeps taken, it is possible to construct correctors for the scheme the increase the accuracy and local order of the scheme. We demonstrate one simple such corrector in Section 4. Numerical examples demonstrating the effectiveness of this scheme are presented in Section 5.

5.2 The Krylov Subspace Projection Method

We describe the Krylov subspace projection method for approximating $\varphi_1(\Delta t L)$ in (5.2). We motivate this by showing how the leading powers of $\Delta t L$ in L are captured by the subspace.

Recalling (1.14), the series definition of $\varphi_1(\Delta t L)$ is,

$$\varphi_1(\Delta t L) \equiv \sum_{k=0}^{\infty} \frac{(\Delta t L)^k}{(k+1)!}. \tag{5.6}$$

The challenge in using the scheme (5.2) is to efficiently compute, or approximate, the action of φ_1 on the vector g_n^{etd}. Using (5.6) we can re-write the scheme in (5.2) as

$$u_{n+1}^{etd} = u_n^{etd} + \Delta t \sum_{k=0}^{\infty} \frac{(\Delta t L)^k}{(k+1)!} g_n^{etd}. \tag{5.7}$$

The sum in (5.6) is useful in motivating a polynomial Krylov subspace approximation. The m-dimensional Krylov subspace for the matrix L and vector $g \in \mathbb{R}^N$ is defined by:

$$\mathcal{K}_m(L, g) = \operatorname{span}\{g, Lg, \ldots, L^{m-1}g\}. \tag{5.8}$$

If the sum in (5.6) is approximated by the first m terms, this is equivalent to approximation in the subspace $\mathcal{K}_m(L, g_n^{etd})$ in (5.8). We now derive some simple results about the general subspace $\mathcal{K}_m(L, g)$, with arbitrary vector g, before using the results with $g = g_n^{etd}$ to demonstrate how they are used in the evaluation of (5.2).
The Arnoldi algorithm (see e.g. [55], [53]), a modified GramSchmidt process, is used to produce an orthonormal basis $\{v_1, \ldots, v_n\}$ for the space $\mathcal{K}_m(L, g)$ such that

$$\operatorname{span}\{v_1, v_2, \ldots, v_m\} = \operatorname{span}\{g, Lg, \ldots, L^{m-1}g\}. \tag{5.9}$$

It produces two matrices $V_m \in \mathbb{R}^{N\times m}$, whose columns are the v_k, and an upper Hessenburg matrix $H_m \in \mathbb{R}^{m\times m}$. The matrices L,H_m and V_m are related by

$$L = V_m^T H_m V^T, \tag{5.10}$$

see, for example, equation (2) in [53], of which (5.10) is a consequence. From (5.10) if follows that,

$$V_m H_m V_m^T = V_m V_m^T L V_m V_m^T. \tag{5.11}$$

For any $x \in \mathcal{K}_m(L, g)$,

$$V_m V_m^T x = x,$$

since $V_m V_m^T x$ represents the orthogonal projection into the space $\mathcal{K}_m(L, g)$. Therefore, since $L^k g \in \mathcal{K}_m(L, g_n)$, we also have that

$$V_m V_m^T L^k g = L^k g$$

for $0 \leq k \leq m - 1$. We now consider the relationship between $L^k g$ and $V_m H_m^k V_m^T g$.

Lemma 5.2.1. *Assume* $0 \leq k \leq m-1$. *Then for* H_m, V_m *corresponding to the Krylov subspace* $\mathcal{K}(L, g)$,

$$V_m H_m^k V_m^T g = L^k g. \tag{5.12}$$

Proof. We have that $V_m H_m^k V_m^T = (V_m H_m V_m^T)^k$ since $V_m^T V_m = I$. Let $\pi \equiv V_m V_m^T$, the projector into $\mathcal{K}(L, g)$, so that (5.11) can be more briefly written $V_m H_m V_m^T = \pi L \pi$. Then by (5.11) we find

$$\begin{aligned} V_m H_m^k V_m^T g &= (\pi L \pi)^k g \\ &= (\pi L \pi)^{k-1} \pi L \pi g = (\pi L \pi)^{k-1} L g \\ &= (\pi L \pi)^{k-2} \pi L \pi L g = (\pi L \pi)^{k-2} L^2 g \\ &= \ldots = L^k g. \end{aligned}$$

□

Now consider using the vector $g = g_n^{etd}$, to generate the subspace $\mathcal{K}_m(L, g_n^{etd})$, and the corresponding matrices H_m, V_m, by the Arnoldi algorithm. By Lemma 5.2.1 we have that, up to $k = m$,

$$V_m H_m^k V_m^T g_n^{etd} = L^k g_n^{etd}.$$

Thus, inserting the approximation $L \approx V_m H_m^k V_m^T$ in $\varphi_1(\Delta t L)$ the first m terms in the sum in (5.7) are correctly approximated. The Krylov approximation is then

$$\begin{aligned} \Delta t \varphi_1(\Delta t L) g_n \approx \Delta t \varphi_1(\Delta t V_m H_m V_m^T) g_n &= \Delta t V_m \varphi_1(\Delta t H_m) V_m^T g_n \\ &= ||g_n|| \Delta t V_m \varphi_1(\Delta t H_m) e_1. \end{aligned} \tag{5.13}$$

Let us introduce a shorthand notation for the Krylov approximation of the $\varphi-$function. Analogous to (5.4), let

$$\tilde{p}_\tau \equiv \tau V_m \varphi_1(\tau H_m) V_m^T \approx p_\tau \tag{5.14}$$

for some $\tau \in \mathbb{R}$. Using (5.13) and (5.14) we then approximate (5.5) by

$$u_{n+1}^{etd} = u_n^{etd} + \tilde{p}_{\Delta t} \left(L u_n + F_n \right).$$

The key here is that the $\varphi_1(\Delta t H_m)$ now needs to be evaluated instead of $\varphi_1(\Delta t L)$. m is chosen such that $m \ll N$, and a classical method such as a rational Padé is used for $\varphi_1(\Delta t H_m)$, which would be prohibitively expensive for $\varphi_1(\Delta t L)$ for large N. One step of the ETD1 scheme (5.2), under the approximation $\varphi_1(\Delta t L) \approx V_m \varphi_1(\Delta t H_m) V_m^t$, becomes

$$\begin{aligned} u_{n+1} &= u_n + \Delta t V_m \varphi_1(\Delta t H) V_m^T g_n \\ &= ||g_n|| \Delta t V_m \varphi_1(\Delta t H) e_1, \end{aligned} \tag{5.15}$$

where e_1 is the first basis vector in $\mathbb{R}^m$.

5.3 Recycling the Krylov subspace

In the Krylov subspace projection method described in §5.2, the subspace $\mathcal{K}_m(L, g_n)$ and thus the matrices H_m and V_m depend on g_n. At each step it is understood that a new subspace must be formed, and H_m, V_m be re-generated by the Arnoldi method, since g_n changes. In [96] it is demonstrated that splitting the timestep into two substeps, and recycling H_m and V_m, i.e. recycling the Krylov subspace, can be viable (in that it does not decrease the local order of the scheme, and apparently decreases the error). Here we expand on this concept with a more detailed analysis of the effect of this kind of recycled substepping applied to the locally second order ETD1 scheme (5.2).

The idea is to replace a single step of length Δt of (5.15) with S substeps of length δt, such that $\Delta t = S\delta t$. We denote the approximations used in this scheme analogously to the notation for ETD1 earlier, without the *etd* subscript, and extend the notation slightly to keep track of substeps. Thus,

$$u_{n+\frac{i}{S}} \approx u(t_n + i\delta t);$$

and

$$F_{n+\frac{j-1}{S}} \approx F(t_n + i\delta t, u_{n+\frac{i}{S}}).$$

At the start of the step we calculate H_m, V_m, from g_n,

$$u_{n+\frac{1}{S}} = u_n + \delta t V_m \varphi_1(\delta t H_m) V_m^T g_n, \tag{5.16}$$

and for the remaining $S-1$ steps,

$$u_{n+\frac{j}{S}} = u_{n+\frac{j-1}{S}} + \delta t V_m \varphi_1(\delta t H_m) V_m^T \left(L u_{n+\frac{j-1}{S}} + F_{n+\frac{j-1}{S}} \right), \; 1 < j \leq S, \tag{5.17}$$

where the matrices H_m and V_m *are not re-calculated for any substep*, $j > 1$. We call substeps of the form (5.17) 'recycled steps' and substeps of the form (5.16) 'initial steps'. The approximation to $u(t_n + \Delta t)$ at the end of the step of length Δt is then given by

$$u_{n+1} = u_{n+\frac{S-1}{S}} + \delta t V_m \varphi_1(\delta t H_m) V_m^T \left(L u_{n+\frac{S-1}{S}} + F_{n+\frac{S-1}{S}} \right). \tag{5.18}$$

The recycling steps (5.16), (5.17) can be succinctly expressed using the definition of $\tilde{p}_\tau$:

$$u_{n+\frac{1}{S}} = u_n + \tilde{p}_{\delta t} \left(L u_n + F_n \right), \tag{5.19}$$

$$u_{n+\frac{j}{S}} = u_{n+\frac{j-1}{S}} + \tilde{p}_{\delta t} \left(L u_{n+\frac{j-1}{S}} + F_{n+\frac{j-1}{S}} \right), \; 1 < j \leq S. \tag{5.20}$$

5.3.1 The local error of the recycling scheme

We now derive an expression for the local error of the scheme defined by (5.16), (5.17). In particular this is used to derive an expression for the leading term of the local error, and here show that the leading term decreases with the number of substeps S. As we are examining the local error, we use the local error assumption that $u_n = u(t_n)$, i.e. that the value at the start of the step is exact and thus the error is only over one step.

We also make an assumption about the accuracy of the initial Krylov approximation with respect to the error of the scheme. Let the error in the polynomial Krylov approximation over a single step (including the error from the approximation of $\varphi_1(\tau H_m)$ using, e.g. Padé approximation), with subspace dimension of size m, be given by $\mathcal{E}_{n+1}^m$, so that,

$$\tilde{p}_\tau g_n = p_\tau g_n + \mathcal{E}_{n+1}^m. \tag{5.21}$$

Assumption 5.3.1. *The Krylov approximation error $\mathcal{E}_{n+1}^m$ is much less than the error of ETD1, and thus does not affect the leading term of the local error of ETD1.*

Bounds on $\mathcal{E}_{n+1}^m$ can be found in for example [54, 53]. Practically, we can always reduce Δt or increase m until Assumption 5.3.1 is satisfied.
For the local error of the recycling scheme, the following result will be used.

Lemma 5.3.2. *For any $\tau_1, \tau_2 \in \mathbb{R}$, and any vector $v \in \mathbb{R}^N$,*

$$p_{\tau_1} v + p_{\tau_2} \left(L p_{\tau_1} v + v \right) = p_{\tau_1+\tau_2} v,$$

and the same relation holds for the Krylov approximations, that is,

$$\tilde{p}_{\tau_1} v + \tilde{p}_{\tau_2} \left(L \tilde{p}_{\tau_1} v + v \right) = \tilde{p}_{\tau_1+\tau_2} v.$$

Proof. We prove the second equation. The first can be proved using an almost identical argument, replacing $\tilde{p}_\tau$ by p_τ where appropriate.
By the definitions of $\tilde{p}_\tau$, and φ_1, i.e. $\tilde{p}_\tau = V\varphi_1(\tau H_m)V_m^T = V_m H_m^{-1}\left(e^{\tau_2 H_m} - I\right) V_m^T$ we have

$$\tilde{p}_{\tau_2} \left(L \tilde{p}_{\tau_1} v + v \right) = V_m H_m^{-1} \left(e^{\tau_2 H_m} - I \right) V_m^T \left(L V_m H_m^{-1} \left(e^{\tau_1 H_m} - I \right) V_m^T + I \right) v.$$

Simplifying and making use of $H_m = V_m^T L V_m$ (5.10) this becomes

$$\tilde{p}_{\tau_2} \left(L \tilde{p}_{\tau_1} v + v \right) = V_m H_m^{-1} \left(e^{(\tau_2+\tau_1) H_m} - e^{\tau_1 H_m} \right) V_m^T v.$$

Now using the definition of $\tilde{p}_{\tau_1}$,

$$\begin{aligned} &\tilde{p}_{\tau_1} v + \tilde{p}_{\tau_2} \left(L \tilde{p}_{\tau_1} v + v \right) \\ &= V_m H_m^{-1} \left(e^{\tau_1 H_m} - I \right) V_m^T v + V_m H_m^{-1} \left(e^{(\tau_2+\tau_1) H_m} - e^{\tau_1 H_m} \right) V_m^T v \qquad (5.22) \\ &= V_m H_m^{-1} \left(e^{(\tau_2+\tau_1) H_m} - I \right) V_m^T v, \end{aligned}$$

which is $\tilde{p}_{\tau_1+\tau_2}v$ as desired. □

Without recycling substeps, a single ETD1 step (5.2) of length Δt, using the polynomial Krylov approximation, would be:

$$u_{n+1}^{etd} = u_n^{etd} + \tilde{p}_{\Delta t} g_n^{etd}. \tag{5.23}$$

The errors in this approximation, arising from both the use of the Krylov approximation (see e.g. [45]) and the ETD1 scheme are well known (see e.g. [22], [29]). We wish to compare u_{n+1}^{etd} with the u_{n+1} obtained after some number S of recycled substeps. We can write

$$u_{n+1} = u_n + \tilde{p}_{\Delta t} g_n + R_{n+1}^S,$$

where R_{n+1}^S represents the deviation from (5.23). Then we have:

Lemma 5.3.3. *The approximation $u_{n+\frac{j}{S}}$ produced by j substeps of the recycling scheme (5.19), (5.20), satisfies*

$$u_{n+\frac{j}{S}} = u_n + \tilde{p}_{j\delta t} g_n + R_{n+\frac{j}{S}}^S, \tag{5.24}$$

with

$$R_{n+\frac{j}{S}}^S = \sum_{k=1}^{j} (I + \tilde{p}_{\delta t} L)^{j-k} \tilde{p}_{\delta t} (F_{n+\frac{k-1}{S}} - F_n). \tag{5.25}$$

Proof. By induction. First, for the case $j = 1$, $u_{n+\frac{1}{S}}$ is given by (5.19) and $R_{n+\frac{j}{S}}^S$ is required to be zero. Equation (5.25) gives $R_{n+\frac{1}{S}}^S = \tilde{p}_{\delta t}(F_{n+\frac{0}{S}} - F_n) = 0$ as desired. Assume now (5.24) holds for some $j \geq 1$. Then $u_{n+\frac{j+1}{S}}$ is obtained by a step of (5.20),

$$u_{n+\frac{j+1}{S}} = u_{n+\frac{j}{S}} + \tilde{p}_{\delta t} \left(L u_{n+\frac{j}{S}} + F_{n+\frac{j}{S}} \right)$$

using (5.24) we find,

$$u_{n+\frac{j+1}{S}} = u_{n+\frac{j}{S}} + \tilde{p}_{\delta t} \left(L u_n + L \tilde{p}_{j\delta t} g_n + L R_{n+\frac{j}{S}}^S + F_{n+\frac{j}{S}} \right),$$

and since $Lu_n = g_n - F(t_n, u(t_n))$,

$$\begin{aligned}
u_{n+\frac{j+1}{S}} &= u_{n+\frac{j}{S}} + \tilde{p}_{\delta t}\left(g_n + L\tilde{p}_{\delta t}g_n + LR^S_{n+\frac{j}{S}} + F_{n+\frac{j}{S}} - F(t_n, u(t_n))\right) \\
&= u_{n+\frac{j}{S}} + \tilde{p}_{\delta t}\left(g_n + L\tilde{p}_{\delta t}g_n\right) + \tilde{p}_{\delta t}LR^S_{n+\frac{j}{S}} + \tilde{p}_{\delta t}(F_{n+\frac{j}{S}} - F_n) \\
&= u_n + \tilde{p}_{j\delta t}g_n + \tilde{p}_{\delta t}\left(g_n + L\tilde{p}_{\delta t}g_n\right) + (I + \tilde{p}_{\delta t}L)R^S_{n+\frac{j}{S}} + \tilde{p}_{\delta t}(F_{n+\frac{j}{S}} - F_n).
\end{aligned}$$

Thus by Lemma 5.3.2 we have that,

$$u_{n+\frac{j+1}{S}} = u_n + \tilde{p}_{(j+1)\delta t}g_n + \tilde{p}_{\delta t}(F_{n+\frac{j}{S}} - F_n) + (I + \tilde{p}_{\delta t}L)R^S_{n+\frac{j}{S}}.$$

To complete the proof we require that:

$$R^S_{n+\frac{j+1}{S}} = \tilde{p}_{\delta t}(F_{n+\frac{j}{S}} - F_n) + (I + \tilde{p}_{\delta t}L)R^S_{n+\frac{j}{S}}. \tag{5.26}$$

By the induction hypothesis that (5.25) holds for j,

$$\begin{aligned}
&\tilde{p}_{\delta t}(F_{n+\frac{j}{S}} - F_n) + (I + \tilde{p}_{\delta t}L)R^S_{n+\frac{j}{S}} \\
&= \tilde{p}_{\delta t}(F_{n+\frac{j}{S}} - F_n) + (I + \tilde{p}_{\delta t}L)\sum_{k=1}^{j}(I + \tilde{p}_{\delta t}L)^{j-k}\tilde{p}_{\delta t}(F_{n+\frac{j-1}{S}} - F_n) \\
&= \sum_{k=1}^{j+1}(I + \tilde{p}_{\delta t}L)^{j+1-k}\tilde{p}_{\delta t}(F_{n+\frac{k-1}{S}} - F_n) = R^S_{n+\frac{j+1}{S}}.
\end{aligned} \tag{5.27}$$

Hence the lemma is proved. □

At the end of S substeps of a substepping scheme (5.19), (5.20) we have

$$u_{n+1} = u_n + \tilde{p}_{\Delta t}g_n + R^S_{n+1}, \tag{5.28}$$

with

$$R^S_{n+1} = \sum_{k=1}^{S}(I + \tilde{p}_{\delta t}L)^{S-k}\tilde{p}_{\delta t}(F_{n+\frac{k-1}{S}} - F_n). \tag{5.29}$$

Using (5.29) we now express the leading order term of the local error in terms of S. First we examine the leading order term of R^S_{n+1}.

Lemma 5.3.4. *The term $R^S_{n+\frac{j}{S}}$ in Lemma 5.3.3, when expanded in powers of Δt, satisfies*

$$R^S_{n+\frac{j}{S}} = \frac{j(j-1)}{2}\delta t^2 V_m V_m^T \frac{dF(t_n, u_n)}{dt} + O(\Delta t^3). \tag{5.30}$$

Proof. By induction. Since $(F_{n+\frac{0}{S}} - F_n) = 0$, then, $R^S_{n+\frac{1}{S}} = 0$, so that (5.30) is true for $j = 1$.

Now assume the result holds for some j. Then we can express the term $F_{n+\frac{j}{S}}$ follows:

$$\begin{aligned}
F(t_{n+\frac{j}{S}}, u_{n+\frac{j}{S}}) &= F(t_{n+\frac{j}{S}}, u_n + (j\delta t g_n + O(\delta t^2))) \\
&= F(t_{n+\frac{j}{S}}, u_n) + \frac{\partial F}{\partial u}(t_{n+\frac{j}{S}}, u_n)(j\delta t g_n + O(\delta t^2)) + O(\delta t^2) \\
&= F(t_n, u_n) + j\delta t\frac{\partial F}{\partial t}(t_n, u_n) + j\delta t\frac{\partial F}{\partial u}(t_n, u_n)g_n + O(\delta t^2) \\
&= F(t_n, u_n) + j\delta t\frac{dF}{dt}(t_n, u_n) + O(\delta t^2).
\end{aligned}$$

We thus have that

$$(F_{n+\frac{j}{S}} - F_n) = j\delta t\frac{dF}{dt}(t_n, u_n) + O(\delta t^2). \tag{5.31}$$

We then insert (5.31) into the inductive expression (5.43) for $R^S_{n+\frac{j}{S}}$ and use the expansion $\tilde{p}_{\delta t} = \delta t V_m V_m^T + O(\delta t^2)$ to give,

$$R^S_{n+\frac{j+1}{S}} = \delta t V_m V_m^T j\delta t\frac{dF}{dt}(t_n, u_n) + (I + \delta t V_m V_m^T L)R^S_{n+\frac{j}{S}} + O(\delta t^3).$$

Using the induction assumption (5.30),

$$R^S_{n+\frac{j+1}{S}} = \delta t V_m V_m^T j\delta t\frac{dF}{dt}(t_n, u_n) + \frac{j(j-1)}{2}\delta t^2 V_m V_m^T\frac{dF(t_n, u_n)}{dt} + O(\Delta t^3),$$

Noting that $\Delta t = S\delta t$ we can write $O(\Delta t^3)$ as $O(\delta t^3)$. Collecting terms we have,

$$R^S_{n+\frac{j+1}{S}} = \left(\frac{j(j-1)}{2} + j\right)\delta t^2 V_m V_m^T\frac{dF(t_n, u_n)}{dt} + O(\Delta t^3).$$

The lemma follows since $\frac{j(j-1)}{2} + j = \frac{j(j+1)}{2}$, which completes the induction argument. $\square$

The leading local error term of the ETD1 scheme without substeps is well known to be $\frac{\Delta t^2}{2}\frac{dF(t)}{dt}$ (see [29]), so that we can finally recover the leading term from Lemma 5.3.3.

Corollary 5.3.5. *It follows from (5.24), the leading term of the recycling scheme after j steps is*

$$u_{n+\frac{j}{S}} = u_n + j\delta t g_n + \frac{j^2\delta t^2}{2}Lg_n + \frac{j(j-1)}{2}\delta t^2 V_m V_m^T \frac{dF(t_n,u_n)}{dt} + O(\delta t^3). \tag{5.32}$$

Corollary 5.3.6. *The local error $u(t_n+\Delta t) - u_{n+1}$ of an ETD1 Krylov recycling scheme is second order for any number S of recycled substeps. Moreover, the local error after j recycled steps is*

$$\begin{aligned} u(t_n + j\delta t) - u_{n+\frac{j}{S}} &= u(t) + j\delta t g(t_n) + \frac{(j\delta t)^2}{2}\left(Lg(t_n) + \frac{df(t)}{dt}\right) \\ &- (u_n + j\delta t g_n + \frac{j^2\delta t^2}{2}Lg_n + \frac{j(j-1)}{2}V_m V_m^T \delta t^2 \frac{dF(t_n,u_n)}{dt}) + O(\delta t^3) \\ &= \frac{(j\delta t)^2}{2}\left(I - \frac{j-1}{j}V_m V_m^T\right)\frac{df(t)}{dt} + O(\delta t^3). \end{aligned} \tag{5.33}$$

In particular

$$u(t_n+\Delta t) - u_{n+1} = \frac{\delta t^2}{2}\left(S^2 - S(S-1)V_m V_m^T\right)\frac{df(t)}{dt} + O(\delta t^2), \tag{5.34}$$

or in terms of Δt

$$u(t_n+\Delta t) - u_{n+1} = \frac{\Delta t^2}{2}\left(I - \frac{S-1}{S}V_m V_m^T\right)\frac{df(t)}{dt} + O(\Delta t^2). \tag{5.35}$$

It is interesting to compare (5.35) with the leading term of the local error of regular ETD1, $\frac{\Delta t^2}{2}\frac{df}{dt}$. Since $V_m V_m^T$ is the orthogonal projector into $\mathcal{K}$, then we can see that the $\frac{\Delta t^2}{2}\frac{S-1}{S}V_m V_m^T\frac{df(t)}{dt}$ part in (5.35) is the projection of the ETD1 error into $\mathcal{K}$, multiplied by a factor $\frac{S-1}{S} \leq 1$. Thus, in the leading term, according to (5.35), the recycling scheme reduces the error of ETD1 by effectively eliminating the part of the error which lives in $\mathcal{K}$. In the limit $S \to \infty$, the entirety of the error in $\mathcal{K}$ will be eliminated. The effectiveness of the recycling scheme will therefore be affected by how much of $\frac{df(t)}{dt}$ can be found in $\mathcal{K}$.

Corollary 5.3.6 shows that using $S > 1$ recycled substeps is advantageous over the basic ETD1 scheme, in the sense of reducing the magnitude of the leading local error term, whenever

$$\left|\left|\left(I - \frac{S-1}{S} V_m V_m^T\right) \frac{df(t)}{dt}\right|\right| < \left|\left|\frac{df(t)}{dt}\right|\right|, \tag{5.36}$$

where $||\cdot||$ is a given vector norm. We can go further and show that increasing S will decrease the Euclidean norm $||\cdot||_2$ of the leading term of the local error. This is done in Lemma 5.3.8 below; first we require a result on $V_m V_m^T$, the projector into the Krylov Supspace $\mathcal{K}$.

Remark 5.3.7. *Let $x \neq 0$ be a vector such that $V_m V_m^T x \neq 0$, then*

$$\left|\left|\left(I - \alpha V_m V_m^T\right) x\right|\right|_2^2 = ||x||_2^2 + \left[(1-\alpha)^2 - 1\right] \left|\left|V_m V_m^T x\right|\right|_2^2, \tag{5.37}$$

for $\alpha \in \mathbb{R}$.

Proof. An elementary result for orthogonal projectors (see, e.g. [97]) is that

$$||x||_2^2 = \left|\left|V_m V_m^T x\right|\right|_2^2 + \left|\left|(I - V_m V_m^T) x\right|\right|_2^2, \tag{5.38}$$

which follows from $V_m V_m^T x \perp (I - V_m V_m^T)x$ (the orthogonality of $V_m V_m^T x$ and $(I - V_m V_m^T)x$) and the definition of the Euclidean norm. Equation (5.37) is a generalisation of (5.38) as can be shown as follows.

Write $x - \alpha V_m V_m^T x = (I - V_m V_m^T)x + (1-\alpha) V_m V_m^T x$, and then, noting that $(I - V_m V_m^T)x \perp (1-\alpha) V_m V_m^T x$,

$$\begin{aligned}
&\left|\left|x - \alpha V_m V_m^T x\right|\right|_2^2 = \\
&\left((I - V_m V_m^T)x + (1-\alpha) V_m V_m^T x\right)^T \left((I - V_m V_m^T)x + (1-\alpha) V_m V_m^T x\right) \\
&= \left|\left|(I - V_m V_m^T)x\right|\right|_2^2 + (1-\alpha)^2 \left|\left|V_m V_m^T x\right|\right|_2^2,
\end{aligned} \tag{5.39}$$

then using (5.38) to substitute for $\left|\left|(I - V_m V_m^T)x\right|\right|_2^2$ yields (5.37). □

Lemma 5.3.8. *Assume $\frac{df(t)}{dt} \neq 0$ and $V_m V_m^T \frac{df(t)}{dt} \neq 0$. Let Err_1 be the local error* *using the recycling scheme over a timestep of length Δt with $S_1 \geq 1$ substeps, and Err_2 the local error after using the recycling scheme over a timestep of the same length with S_2 substeps. Let $S_2 > S_1$. Then,*

$$||Err_2||_2 < ||Err_1||_2 \,.$$

Proof. The local error is given in Corollary (5.3.6). Let $\frac{S_k - 1}{S_k} \equiv \beta_k$, $k = 1, 2$. We need to show that

$$\left|\left|\left(I - \beta_2 V_m V_m^T\right) \frac{df(t)}{dt}\right|\right|_2 < \left|\left|\left(I - \beta_1 V_m V_m^T\right) \frac{df(t)}{dt}\right|\right|_2 .$$

Let $x \equiv \left(I - \beta_1 V_m V_m^T\right) \frac{df(t)}{dt}$, then $\left(I - \beta_2 V_m V_m^T\right) \frac{df(t)}{dt} = x - \left(\frac{\beta_1 - \beta_2}{\beta_1 - 1}\right) V_m V_m^T x$ (showing this involves using $V_m V_m^T V_m V_m^T = V_m V_m^T$). Letting $\gamma \equiv \frac{\beta_1 - \beta_2}{\beta_1 - 1}$, we then need to show

$$\left|\left|\left(I - \gamma V_m V_m^T\right) x\right|\right|_2 < ||x||_2 \,. \tag{5.40}$$

Note that we have that $V_m V_m^T x \neq 0$ from the assumptions. This is because,

$$V_m V_m^T x = \left(V_m V_m^T - \beta_1 V_m V_m^T\right) \frac{df(t)}{dt},$$

since $V_m V_m^T V_m V_m^T \frac{df(t)}{dt} = V_m V_m^T \frac{df(t)}{dt}$, as $V_m V_m^T \frac{df(t)}{dt}$ is already entirely within $\mathcal{K}$. Then,

$$V_m V_m^T x = (1 - \beta_1) V_m V_m^T \frac{df(t)}{dt}.$$

We have that $1 - \beta_1 = \frac{1}{S_1} \neq 0$ and $V_m V_m^T \frac{df(t)}{dt} \neq 0$, so that $V_m V_m^T x \neq 0$.
To prove the lemma we apply (5.37) to x, with γ in place of α. If we have that $[(1 - \gamma)^2 - 1] < 0$, then (5.40) is true since $V_m V_m^T x \neq 0$. An equivalent requirement is $\gamma \in (0, 2)$. Some algebra gives us $\gamma = 1 - \frac{S_1}{S_2}$. Since $S_2 > S_1$, it follows that $\gamma \in (0, 2)$. □

From Lemma (5.3.8) we can see that any number S of recycled Krylov substeps will not only maintain the local error order of the ETD1 scheme, but will also decrease the 2-norm of the leading term with increasing S. Note that the leading term does not tend towards zero as $S \to \infty$, but towards a constant. We thus expect diminishing returns in the increase in accuracy with increasing S, and the existence of an optimal S for efficiency.

5.4 Substepping with the scheme EEM

As an aside we show here how it is possible to apply the analysis of the substepping method to the locally third order exponential integrator scheme EEM. EEM is closely related to ETD1, but can be applied to an ODE that is not semilinear. Applied to the ODE,

$$\frac{du}{dt} = g(u),$$

where $g(u)$ may not be semilinear, the scheme EEM is

$$u_{n+1} = u_n + \Delta t \varphi_1(\Delta t J_n) g_n.$$

Here $J(u) = \frac{\partial u}{\partial g}$ is the Jacobian of g and $J_n = J(u_n)$. Note that EEM is just ETD1 with L replaced by J_n; and indeed is identical in principal to ROS1; here we call EEM the scheme applied to a nonlinear PDE and ROS1 the application to a semilinear one. The Jacobian J_n will change each timestep, but is kept fixed for the entire step, including recycling substeps. Therefore an S step recycling scheme can be defined on EEM in exactly the same way as the recycling scheme for ETD1. Note that the Krylov subspace will be generated for J and g in the EEM case, i.e. $\mathcal{K} = \mathcal{K}(J_n, g_n)$.

We can apply Lemma 5.3.3 to the EEM substepping scheme just by replacing L with J_n and recognising that thus $F_{n+\frac{j+1}{S}} = g_{n+\frac{j+1}{S}} - Lu_{n+\frac{j+1}{S}}$ becomes $F_{n+\frac{j+1}{S}} = g_{n+\frac{j+1}{S}} - J_n u_{n+\frac{j+1}{S}}$.

Corollary 5.4.1. *When applying the Krylov subspace recycling scheme to EEM, the* *approximation satisfies*

$$u_{n+\frac{j}{S}} = u_n + \tilde{p}_{j\delta t} g_n + R^S_{n+\frac{j}{S}}, \tag{5.41}$$

where now the approximate $\varphi-$function

$$\tilde{p}_\tau \equiv \tau V_m \varphi_1(\tau H_m) V_m^T \tag{5.42}$$

approximates

$$\tau\varphi_1(\tau J)$$

and the Krylov subspace is generated from J and g_n. The remainder $R^S_{n+\frac{j}{S}}$ satisfies the recursion relation,

$$R^S_{n+\frac{j+1}{S}} = \tilde{p}_{\delta t}(F_{n+\frac{j}{S}} - F_n) + (I + \tilde{p}_{\delta t} L) R^S_{n+\frac{j}{S}}, \tag{5.43}$$

where $F_{n+\frac{j+1}{S}} = g_{n+\frac{j+1}{S}} - J_n u_{n+\frac{j+1}{S}}$.

Proof. Following through the argument of Lemma 5.3.3 with the new definitions gives proof of (5.43), which is analgous to equation (5.25) in Lemma 5.3.3. □

The remainder for the recycling-EEM scheme is therefore essentially the same in form as the ETD1 recycling scheme, however the Taylor expansion of the remainder is different. We use some results from vector calculus. Let $\hat{J}_i$ be the Hessian matrix

$$\begin{pmatrix} (\hat{g}_i)_{x_1x_1} & (\hat{g}_i)_{x_1x_2} & \cdots \\ (\hat{g}_i)_{x_2x_1} & (\hat{g}_i)_{x_2x_2} & \cdots \\ \cdots & \cdots & \cdots \end{pmatrix},$$

where $\hat{g}_i$ is the ith entry of the vector g. Then let the tensor $\hat{\mathbf{J}}$ be a vector with the matrix $\hat{J}_i$ in its ith entry. It can be shown that the Taylor series of a vector function $g(x)$ satisfies

$$g(x + \Delta x) = g(x) + J\Delta x + \frac{1}{2}\Delta x^T \hat{\mathbf{J}} \Delta x + O(\Delta x^3)$$

It can also be shown that

$$\frac{dx}{dt}\hat{\mathbf{J}} = \frac{d}{dt}J$$

see the Appendix §9.2 for proofs. And following from these it is possible to show that the leading term of the local error of EEM is

$$\frac{1}{6}\Delta t^3 g^T \hat{\mathbf{J}} g.$$

We then proceed in an analogy to Lemma 5.3.4, Taylor expanding the remainder $R^S_{n+\frac{j}{S}}$ from (5.43) to find the local error of the EEM scheme with recycled substeps.

Lemma 5.4.2. *For the EEM recycling scheme, the leading term of $R^S_{n+\frac{j}{S}}$ satisfies*

$$R^S_{n+\frac{j}{S}} = \alpha(j)\delta t^3 V_m V_m^T g_n^T \hat{\mathbf{J}} g_n + O(\delta t^4)$$

Where the function $\alpha(j)$ satisfies

$$\alpha(j) = \frac{j^3}{6} - \frac{j^2}{4} + \frac{j}{12} = \frac{2j^3 - 3j^2 + j}{24}.$$

Proof. By induction. The base case is true for $j = 1$ with $\alpha(1) = 0$ since there is no recycling at that step. Assume true for some j. Consider $g_{n+\frac{j}{S}}$,

$$g_{n+\frac{j}{S}} = g(u_{n+\frac{j}{S}}) = g(u(t) + j\delta t g(t) + \frac{1}{2}(j\delta t)^2 Jg + O(\delta t^3)),$$

to second order this is

$$g_{n+\frac{j}{S}} = g_n + J\left(j\delta t g_n + \frac{1}{2}(j\delta t)^2 J g_n\right) + \frac{(j\delta t)^2}{2} g_n^T \hat{\mathbf{J}} g_n + O(\delta t^3),$$

where we have made use of the local error assumption $u(t_n) = u_n$. Then,

$$\begin{aligned} &F_{n+\frac{j}{S}} - F_n \\ &= g_{n+\frac{j}{S}} - g_n - Ju_{n+\frac{j}{S}} + Ju_n \\ &= \left(J\left(j\delta t g_n + \frac{1}{2}(j\delta t)^2 Jg_n\right) + \frac{(j\delta t)^2}{2} g_n^T \hat{\mathbf{J}} g_n\right) - J\left(j\delta t g_n + \frac{(j\delta t)^2}{2} Jg_n\right) + O(\delta t^3), \end{aligned} \tag{5.44}$$

where we have used that $u_{n+\frac{j}{S}} = u_n j\delta t g_n + \frac{(j\delta t)^2}{2} Jg_n + O(\delta t^3)$ up to second order, since the induction assumption states that R has zero contribution to second order. Equation (5.44) cancels to give

$$F_{n+\frac{j}{S}} - F_n = \frac{(j\delta t)^2}{2} g_n^T \hat{\mathbf{J}} g_n + O(\delta t^3)$$

Now consider

$$\tilde{p}_{\delta t}\left(F_{n+\frac{j}{S}} - F_n\right) = \frac{j^2(\delta t)^3}{2} V_m V_m^T g_n^T \hat{\mathbf{J}} g_n + O(\delta t^4)$$

The induction relation for $R^S_{n+\frac{j+1}{S}}$ then gives us

$$R^S_{n+\frac{j+1}{S}} = \frac{j^2(\delta t)^3}{2} V_m V_m^T g_n^T \hat{\mathbf{J}} g_n + (I + \tilde{p}_{\delta t} J) R^S_{n+\frac{j}{S}} + O(\delta t^4)$$

To leading order this is

$$R^S_{n+\frac{j+1}{S}} = \left(\frac{j^2}{2} + \alpha(j)\right) \delta t^3 V_m V_m^T g_n^T \hat{\mathbf{J}} g_n + O(\delta t^4)$$

So

$$\alpha(j+1) = \frac{j^2}{2} + \alpha(j),\ \alpha(1) = 0$$

which is satisfied by

$$\alpha(j) = \frac{j^3}{6} - \frac{j^2}{4} + \frac{j}{12} = \frac{2j^3 - 3j^2 + j}{24}.$$

$\square$

We now combine the leading term of the remainder R and the known local error of EEM to find the local error of the new recycling scheme.

Corollary 5.4.3. *The leading term of the local error of the S step recycling scheme for EEM at the end of a timestep is*

$$\frac{\Delta t^3}{6}\left(I-\left(\frac{2S^2-3S+1}{2S^2}\right)V_mV_m^T\right)\mathbf{g}^T\hat{\mathbf{J}}\mathbf{g}.$$

From this we can predict similar properties to the ETD1 recycling scheme. This effectively extends the work of [96] (Caarr, Moroney, Turner), where the recycling substepping EEM scheme was used for a single substep.

5.4.1 Efficiency

We now examine the possibility of an optimal S for efficiency. We define the efficiency of the S substep scheme over one step as

$$\text{EFFICIENCY(S)} \equiv \frac{1}{\text{COST(S)} \times \text{E(S)}},$$

where COST(S) is the time cost of the step, and $E(S)$ the error. We represent an improvement in efficiency of the recycling scheme over the standard $S = 1$ scheme by an improvement factor X as follows,

$$\text{EFFICIENCY(S)} = X\ \text{EFFICIENCY(1)}. \tag{5.45}$$

The larger X is, the greater the improvement provided by the recycling scheme. If $X < 1$, then the substepping scheme is less efficient. We examine the factors that affect the magnitude of X.

We consider the cost of a whole step with recycling as the sum of two components. First, $C_I(m, N)$, the initial part of the step, including the call to the Arnoldi algorithm and then the generation of the matrix $\varphi_1(\delta t H_m)$. The dependence of the cost upon the Krylov subspace size m and the system size N is included implicitly. The cost of the individual substeps is given by $C_s(m, N)$, the cost of the subsequent parts. This is the cost of the vector and matrix operations to form the solution at every substep. We have that

$$\mathrm{COST(S)} \equiv C_I(m,N) + SC_s(m,N).$$

Let us use for the error the leading term previously calculated: For the error we use the Euclidean norm of leading term given in (5.35),

$$\mathrm{E(S)} \equiv \frac{\Delta t^2}{2}\left\|\left(I - \frac{S-1}{S}VV^T\right)\frac{df}{dt}\right\|_2.$$

Then equation (5.45) takes the form

$$\frac{1}{E(S)\left(C_I(m,N)+SC_s(m,N)\right)} = \frac{X}{E(1)\left(C_I(m,N)+C_s(m,N)\right)},$$

which after some rearrangement gives

$$X = \frac{E(1)}{E(S)}\frac{1+\frac{C_I(m,N)}{C_s(m,N)}}{\frac{C_I(m,N)}{C_s(m,N)}+S}.$$

We denote the ratio $r_{IS} \equiv \frac{C_I(m,N)}{C_s(m,N)}$. This ratio will be important later. Then,

$$X = \frac{E(1)}{E(S)}\frac{r_{IS}+1}{r_{IS}+S}. \tag{5.46}$$

We can express (5.46) in terms of vector norms. Let

$$N_f \equiv \left\|\frac{df}{dt}\right\|_2,\ N_{Vf} \equiv \left\|VV^T\frac{df}{dt}\right\|_2.$$

By applying (5.37) to $E(S)$, we have that

$$\frac{2}{\Delta t^2}E(S)^2 = N_f^2 - \left(1-\frac{1}{S^2}\right)N_{Vf}^2.$$

We define

$$d \equiv \frac{1}{N_f}\sqrt{N_f^2 - \left(1 - \frac{1}{S^2}\right)N_{Vf}^2} = \sqrt{1 - \left(1 - \frac{1}{S^2}\right)\left(\frac{N_{Vf}}{N_f}\right)^2}, \tag{5.47}$$

so that $E(S) = \frac{\Delta t^2}{2} dN_f$. As a particular case, $E(1) = \frac{\Delta t^2}{2} N_f$. Consequently (5.46) becomes

$$X = \frac{1}{d}\frac{r_{IS}+1}{r_{IS}+S}. \tag{5.48}$$

From (5.48), we see that the magnitude of X is determined by the variables S, r_{IS} and d. Since d is itself determined by S and the ratio $\frac{N_{Vf}}{N_f}$ in (5.47), we have that X is determined by S, r_{IS} and $\frac{N_{Vf}}{N_f}$.

Increasing the ratio $r_{IS} \equiv \frac{C_I(m,N)}{C_s(m,N)}$ simply makes X larger in (5.48) (the function $\frac{1+y}{S+y}$ is an increasing function in y for $S > 1$ and constant for $S = 1$). Recall that the justification for using recycling is that the cost of calling Arnoldi (included in $C_I(m, N)$) is the dominant cost in a step, so that one or more recycled steps (costing $C_s(m, N)$) can be added without greatly increasing cost. This is expressed by a large r_{IS}. We expect this ratio to increase with m as this will increase the cost of Arnoldi more than increasing the cost of the subsequent matrix vector multiplications.

Decreasing d will increase X in (5.48). From (5.47), we see that increasing S decreases d to the limit $\sqrt{1 - \left(\frac{N_{Vf}}{N_f}\right)^2}$. The other effect of increasing S will be to decrease the $\frac{r_{IS}+1}{r_{IS}+S}$ term in (5.48) to the limit 0. This expresses an effect of diminishing returns in increasing S. While increasing S may increase efficiency initially, there is guaranteed to be a S high enough which takes X below 1 and makes the recycling scheme less efficient than ETD1.

As the ratio $\frac{N_{Vf}}{N_f}$ is increased, d will decrease and thus X will increase. Since $\frac{N_{Vf}}{N_f} = \frac{||VV^T\frac{df}{dt}||}{||\frac{df}{dt}||}$, this is a measure of how much of $\frac{df}{dt}$ can be found in $\mathcal{K}$. This corresponds to the discussion after Corollary 5.3.6.

Predicting the size of the ratio $\frac{||VV^T\frac{df}{dt}||}{||\frac{df}{dt}||}$ is not simple a priori. Trivially, it depends weakly on the ratio of m to N - the size of the subspace versus the size of the vectors projected into it, so that we expect worse results with large N. Also important is the form of f - since the Krylov subspace is based on the vector $Lu + f$; the amount of $\frac{df}{dt}$ that lives in $\mathcal{K}$ may be expected to increase if f is made closer to a linear function.

5.5 Using the additional substeps for correctors

Given (5.21), the ETD1 step (5.23) is

$$u_{n+1}^{etd1} = u_n^{etd} + p_{\Delta t} g_n + \mathcal{E}_{n+1}^m, \tag{5.49}$$

and the substepping error expression (5.28) can be expressed as

$$u_{n+1} = u_n + p_{\Delta t} g_n + \mathcal{E}_n^m + R_{n+1}^S. \tag{5.50}$$

Using variation of constants and a Taylor series expansion of $F(t, u(t))$, the exact solution of (5.1) can be expressed as a power series [22],[29],

$$u(t_n + \Delta t) = e^{\Delta t L} u(t_n) + \sum_{k=1}^{k} \Delta t^k \varphi_k(\Delta t L) F^{(k-1)}(t_n, u_n) + O(\Delta t^k), \tag{5.51}$$

with $F^{(k)}(t_n, u_n) = \frac{d^k F}{dt^k}(t_n, u_n)$. Under the local error assumption $u_n = u(t_n)$, the local error of the ETD1 step given in (5.2) is

$$E_{n+1}^{etd} \equiv u(t_n + \Delta t) - u_n^{etd} + p_{\Delta t} g_n = \sum_{k=2}^{\infty} \Delta t^k \varphi_k(\Delta t L) F^{(k-1)}(t_n). \tag{5.52}$$

Since the approximation from a substepping scheme is related to the approximation from the ETD1 scheme (over one step) by $u_{n+1} = u_{n+1}^{etd} + R_{n+1}^S$, we have the local error for the recycling scheme:

$$u(t_n + \Delta t) - u_n = E_{n+1}^{etd} - \mathcal{E}_{n+1}^m - R_{n+1}^S. \tag{5.53}$$

The terms of error expression (5.53) at arbitrary order can be found using (5.51), Lemma 5.3.3, and the information on Krylov projection methods in §5.2. We can see that the expansion will consist of terms involving the value of F or derivatives thereof at various substeps. These terms can be approximated by finite differences of the values for F at the different substeps, and used as a corrector to eliminate terms for the error. We now proceed with an example of this.

We consider extrapolation in the leading error in the case of two substeps, that is $S \equiv 2$. Assume that the error from the Krylov approximation, $\mathcal{E}_n^m$ is negligible compared to E_n and R_n, so that it does not introduce any terms at the first and second and third order expansion of E_n and R_n. Then we can express exactly the leading second and third order error terms.

First we have the leading terms of E_n^{etd} from (5.52),

$$\begin{aligned} E_n^{etd} &= \Delta t^2 \varphi_2(\Delta t L) F^{(1)}(t_n, u_n) + \Delta t^3 \varphi_3(\Delta t L) F^{(2)}(t_n, u_n) + O(\Delta t^4) \\ &= \frac{\Delta t^2}{2!} F^{(1)} + \frac{\Delta t^3 L}{3!} F^{(1)} + \frac{\Delta t^3}{3!} F^{(2)} + O(\Delta t^4). \end{aligned} \tag{5.54}$$

We also have the leading terms of R_{n+1}^2 (recall that here the superscript 2 indicates that the scheme consists of two substeps after each Arnoldi evaluation),

$$\begin{aligned} R_{n+1}^2 &= \tilde{p}_{\frac{\Delta t}{2}} (F_{n+\frac{1}{2}} - F_n) \\ &= \frac{\Delta t}{2} V_m \left(I + \frac{\Delta t H_m}{2} + \dots \right) V_m^T (F_{n+\frac{1}{2}} - F_n) \\ &= \frac{\Delta t}{2} V_m V_m^T (F_{n+\frac{1}{2}} - F_n) + V_m \frac{\Delta t^2 H_m}{4} V_m^T (F_{n+\frac{1}{2}} - F_n) + O(\Delta t^3). \end{aligned} \tag{5.55}$$

Note that the terms in (5.55) are an order higher than written since $F_{n+\frac{1}{2}} - F_n = \frac{\Delta t}{2} F^{(1)}(t_n, u_n) + O(\Delta t^2)$. We then have that

$$\begin{aligned} u(t_n + \Delta t) = u_{n+1} & \\ &+ \frac{\Delta t^2}{2!} F^{(1)} - \frac{\Delta t}{2} V_m V_m^T (F_{n+\frac{1}{2}} - F_n) \\ &+ \frac{\Delta t^3 L}{3!} F^{(1)} + \frac{\Delta t^3}{3!} F^{(2)} - V_m \frac{\Delta t^2 H_m}{4} V_m^T (F_{n+\frac{1}{2}} - F_n) + O(\Delta t^4). \end{aligned} \tag{5.56}$$

The idea now is as follows. Define a corrected approximation:

$$u_{n+1}^{(c)} \equiv u_{n+1} + C - \frac{\Delta t}{2} V_m V_m^T (F_{n+\frac{1}{2}} - F_n). \tag{5.57}$$

In (5.57), C is a corrector intended to cancel out some of the leading terms in (5.56). The term $\frac{\Delta t}{2} V_m V_m^T (F_{n+\frac{1}{2}} - F_n)$ is the only leading term in (5.56) to involve the matrix V_m, and so is added directly to the corrected approximation (5.57) to allow C to be free of dependence on the matrix V_m. Indeed, C will be a linear combination of the the three function values of F, F_n, $F_{n+\frac{1}{2}}$ and F_{n+1}, available at the end of the full step. The approximation to u produced by substeps of the scheme, and thus also to F, is locally second order. We define the C term as follows, with coefficients α, β, γ to be chosen later.

$$\begin{aligned} C &\equiv \Delta t \alpha F_n + \Delta t \beta F_{n+\frac{1}{2}} + \Delta t \gamma F_{n+1} \\ &= \Delta t \alpha F(t_n) + \Delta t \beta F\left(t_n + \frac{\Delta t}{2}\right) + \Delta t \gamma F\left(t_n + \Delta t\right) + \Delta t^3 E_c + O(\Delta t^4) \\ &= \Delta t \left(\alpha + \beta + \gamma\right) F + \Delta t^2 \left(\frac{\beta}{2} + \gamma\right) F^{(1)} + \frac{\Delta t^3}{2}\left(\frac{\beta}{4} + \gamma\right) F^{(2)} + \Delta t^3 E_c + O(\Delta t^4) \end{aligned} \tag{5.58}$$

where we have used that $F_n = F(t_n, u(t_n))$ (under the local error assumptions), $F_{n+\frac{1}{2}} = F\left(t_n + \frac{\Delta t}{2}\right) + O(\Delta t^2)$ and so on. The new term $\Delta t^3 E_c$ is introduced to represent the $O(\Delta t^3)$ error in writing $\Delta t F_{n+\frac{1}{2}}$ as $\Delta t F\left(t_n + \frac{\Delta t}{2}\right)$, and so on.
From (5.58), we must choose the coefficients to satisfy the two conditions

$$\alpha + \beta + \gamma = 0,$$

$$\frac{\beta}{2} + \gamma = \frac{1}{2}.$$

With these values of the parameters, the local error of the corrected approximation is

$$\begin{aligned}
u(t_n+\Delta t) - u^{(c)}_{n+1} = & \\
& \frac{\Delta t^3}{3!}F^{(2)} - V_m\frac{\Delta t^2 H_m}{4}V_m^T(F_{n+\frac{1}{2}} - F_n) + \frac{\Delta t^3}{2}\left(\frac{\beta}{4}+\gamma\right)F^{(2)} \\
& - \Delta t^3 E_c + O(\Delta t^4) \\
& = \frac{\Delta t^3}{3!}F^{(2)} - V_m\frac{\Delta t^2 H_m}{8}V_m^T F^{(1)} + \frac{\Delta t^3}{2}\left(\frac{\beta}{4}+\gamma\right)F^{(2)} \\
& - \Delta t^3 E_c + O(\Delta t^4).
\end{aligned} \tag{5.59}$$

We have three coefficients to determine, and two constraints. We are therefore in a position to pick another constraint to reduce the new leading error in (5.59). It would be helpful to know the form of the error term E_c, introduced by the approximation of F in (5.58). We have:

$$\begin{aligned}
F_{n+\frac{1}{2}} &= F\left(t_{n+\frac{1}{2}}, u\left(t_{n+\frac{1}{2}}\right) - \frac{\Delta t^2}{8}F' + O(\Delta t^3)\right) \\
&= F\left(t_{n+\frac{1}{2}}, u\left(t_{n+\frac{1}{2}}\right)\right) - \frac{\Delta t^2}{8}\frac{\partial F}{\partial u}F' + O(\Delta t^3),
\end{aligned} \tag{5.60}$$

using (5.33). Recall that $\frac{\partial F}{\partial u}$ is the Jacboian. We also have

$$\begin{aligned}
F_{n+\frac{2}{2}} &= F\left(t_{n+1}, u\left(t_{n+1}\right) - \frac{\Delta t^2}{2}\left(I - \frac{1}{2}V_m V_m^T\right)\frac{dF(t)}{dt} + O(\Delta t^3)\right) \\
&= F\left(t_{n+1}, u\left(t_{n+1}\right)\right) - \frac{\Delta t^2}{2}\frac{\partial F}{\partial u}\left(I - \frac{1}{2}V_m V_m^T\right)\frac{dF(t_n)}{dt} + O(\Delta t^3),
\end{aligned} \tag{5.61}$$

E_c is then

$$-\beta\frac{1}{8}\frac{\partial F}{\partial u}\frac{dF(t_n)}{dt} - \gamma\frac{1}{2}\frac{\partial F}{\partial u}\left(I - \frac{1}{2}V_m V_m^T\right)\frac{dF(t_n)}{dt}.$$

Substituting into (5.59):

$$\begin{aligned}
u(t_n+\Delta t) - u^{(c)}_{n+1} = & \\
& + \frac{\Delta t^3}{3!}F^{(2)} - \frac{\Delta t^2}{4}V_m H_m V_m^T(F_{n+\frac{1}{2}} - F_n) - \frac{\Delta t^3}{2}\left(\frac{\beta}{4}+\gamma\right)F^{(2)} \\
& - \Delta t^3\frac{\partial F}{\partial u}\left(\left(\frac{\beta}{8}+\frac{\gamma}{2}\right)I - \frac{\gamma}{4}V_m V_m^T\right)\frac{dF(t_n)}{dt}.
\end{aligned} \tag{5.62}$$

We have the option here to use the final constraint to eliminate the coefficient of $F^{(2)}$ in the leading term:

$$\frac{\Delta t^3}{3!} - \frac{\Delta t^3}{2}\left(\frac{\beta}{4} + \gamma\right) = 0.$$

Note that E_c cannot be eliminated without taking the inverse of $V_m V_m^T$, so this is not an efficient option. It can be seen that the values that satisfy the three constraints are:

$$\alpha = -\frac{5}{6},\ \ \beta = \frac{2}{3},\ \ \gamma = \frac{1}{6}.$$

Of course E_c also depends on the values of α, β, γ, so the magnitude of the third order term will be affected by the choice of these values also through E_c. With the choices given above, the E_c term in (5.62) becomes:

$$-\Delta t^3 \frac{\partial F}{\partial u}\left(\frac{2}{12}I - \frac{1}{24}V_m V_m^T\right)\frac{dF(t_n)}{dt}.$$

Here we have used all the extra information from the two substeps in order to completely eliminate the lowest order from the local error, and a part of the new leading order term for the scheme. A more thorough use of the error expressions in the lemmas here may give rise to recycling schemes that use more substeps and are able to completely eliminate higher order terms from the error, leading to a kind of new exponential Runge-Kutta framework involving recycled Krylov subspaces. Below we demonstrate the efficacy of our two-step corrected recycling scheme with numerical examples.

5.6 Numerical Results

Here perform numerical tests on the schemes. To estimate the error we produced a low Δt comparison solve $\mathbf{u}_{\text{comp}}$ with ETD2 (1.18). Our ETD2 implementation uses phipm.m [55] for each timestep; which requires the following parameters: an initial Krylov subspace dimension, and an error tolerance. For our comparison solve runs, we used 30 and 10^{-7} respectively for these parameters. See §1.5.2 for details on estimating error and the norm used.

In our tests we compare the 2-step corrector scheme against ETD2 and ROS2 (1.21), which are both also second order. Like ETD2, our ROS2 implementation is powered by phipm.m. For these runs of ETD2 and ROS2, we gave phipm.m the parameters of 30 for the initial Krylov subspace dimension, and 10^{-7} for the error tolerance.

To implement our schemes the substepping scheme and the 2-step corrector scheme, we used the function phipade.m [46] to evaluate the $\varphi-$functions of H_m in matrix form. Our schemes require the size m of the Krylov subspace dimension to use. We used $m = 30$ for all tests.

5.6.1 Allen-Cahn type Reaction Diffusion

The Allen-Cahn reaction term is

$$F(u) = u - u^3,$$

so we approximate the solution to the PDE,

$$\frac{du}{dt} = \nabla^2 Du + u - u^3.$$

A reaction-diffusion system with this reaction term and $u \in [-1, 1]$ can be interpreted as a 2-phase system, where $u = 1$ represents complete saturation of one phase, and $u = -1$ represents complete saturation of the second phase. The reaction term tends to cause the system to evolve towards sharp fronts between regions of the first and second phase.

The (1D) spatial domain for this experiment was $\Omega = [0, 100] \subset \mathbb{R}$. This was discretised into a grid of $N = 100$ cells. The discretisation was finite volume; the boundary conditions were no flow, i.e., $\frac{\partial u}{\partial x} = 0$ where $x = 0$ or $x = 100$. There was a uniform diffusivity field of $D(x) = 1.0$, and no advection. The final time was $T = 1.0$. The initial condition was $u(x, 0) = \cos\left(\frac{2\pi x}{N}\right)$.

In Figure 5.1 we show the initial and final conditions of the system. The comparison solve was produced by ETD2 with $\Delta t = 10^{-4}$. Also shown is the result produced by the recycling scheme with 2 substeps and $\Delta t = 10^{-3}$.

In Figure 5.2 a) and c) we show the estimated error against Δt, for the recycling scheme with varying number of substeps S. The behaviour is as expected; increasing S decreases the error. The scheme is also first order as predicted. The diminishing returns of increasing S (see (5.35)) can also be observed; for example compare the significant increase in accuracy in increasing S from 1 to 5, with the lesser increase in accuracy in increasing S from 5 to 10. Figure 5.2 shows this more emphatically - the increase in accuracy in increasing S from 10 to 50 is significant, but the effect of increasing S from 50 to 100 is very small. The limiting value of the error with respect to S discussed above is clearly close to being reached here.

In Figure 5.2 b) and d) we plot estimated error against cputime to demonstrate the efficiency of the scheme with varying S. In this case increasing S appears to increase efficiency until an optimal S is reached, after which it decreases, as predicted. Figure 5.2 d) shows that the optimal S lies between 50 and 100 for this system.

In Figure 5.3 we compare the 2-step corrector (§5.5) with two other second order exponential integrators, ETD2 and ROS2. Plot a) shows estimated error against Δt again. The corrector scheme is second order as intended, and has quite high accuracy compared to the other two schemes, possibly due to the heuristic attempt to decrease the error in the leading term (see discussion in §5.5). In plot b) we investigate efficiency; we see that the 2-step corrector is of comparable efficiency to ROS2.

In Figure 5.2 b) we see that for the same cputime, increasing S from 1 to 10 decreases the estimated error by roughly an order of magnitude. We can see in Figure 5.2 d) that increasing S from 10 to 50 can further decrease error for a fixed cputime, though less significantly. Comparing a fixed cputime in Figure 5.2 b) and Figure 5.3 b) indicates that the second order, 2-step corrector method can produce error more than one order of magnitude smaller than the first order recycling scheme with $S = 10$.

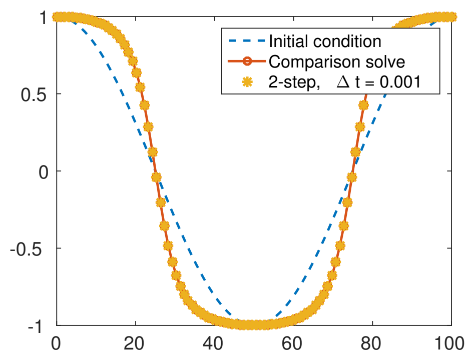


Figure 5.1: The initial and final states of the Allen-Cahn type Reaction Diffusion used.

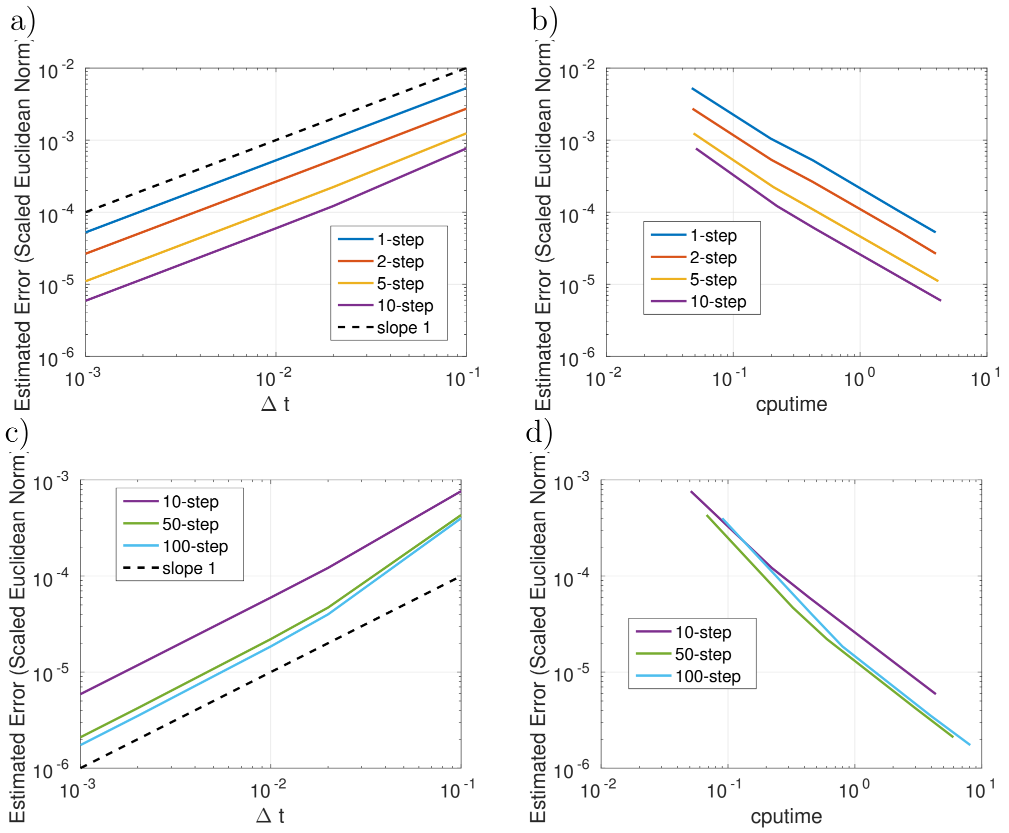


Figure 5.2: Results for the substepping schemes applied to the Allen-Cahn type system. a) and c) display Estimated error against timestep Δt. b) and d) display estimated error against cputime, showing efficiency.

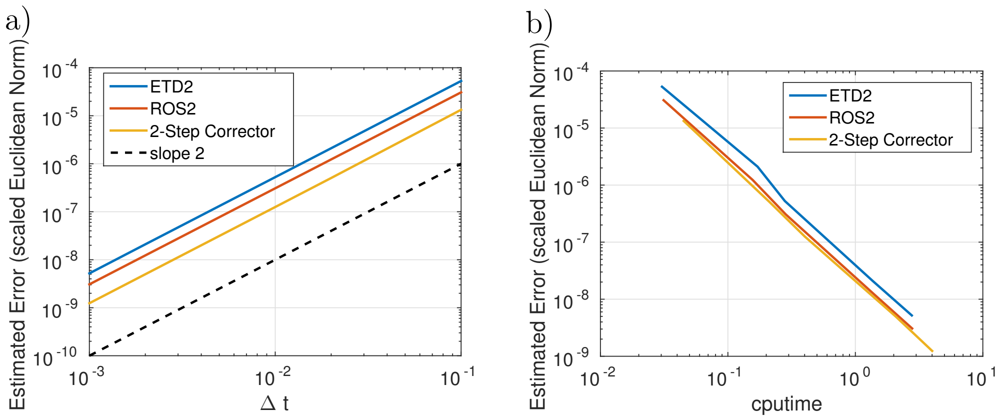


Figure 5.3: AC system, Comparing the second order recycling-corrector scheme with ETD2 and ROS2. a) Estimated error against timestep Δt. b) Estimated error against cputime.

5.6.2 Fracture system with Langmuir-type reaction

This is the same grid and domain as in §3.1.2. We have introduced a Langmuir-type reaction term, that is,

$$F(u, \mathbf{x}) = -\epsilon(\mathbf{x})\frac{u}{1+u},$$

where we define the coefficient ϵ as,

$$\epsilon(\mathbf{x}) = \frac{0.02}{D(\mathbf{x})^2},$$

so we approximate the solution to the PDE,

$$\frac{du}{dt} = \nabla \cdot (\nabla Du + vu) - \epsilon(\mathbf{x})\frac{u}{1+u}.$$

where $D(\mathbf{x})$ is the diffusivity. Recall that the grid is produced such that there is a line of cells, produced by a weighted random walk, where the diffusivity D is 100, representing a fracture. The other cells in the grid have diffusivity value 0.1. Thus the value of ϵ is much lower in the fracture than the rest of the domain (2×10^{-6} compared to 2, respectively). Physically this represents the solute species being much less likely to adsorb to the walls of the porous medium, and be lost, within the fracture. The grid and domain dimensions, initial and boundary conditions, and advection are all the same as the example in §3.1.2. The final time is $T = 2.4$.

In Figure 5.4 we show the final state of the system at time T. The result in plot a) was produced with the 2-step recycling scheme with a timestep $\Delta t = 2.4\times10^{-4}$. Plot b) shows the high accuracy comparison ETD2 solve, produced with $\Delta t = 2.4\times10^{-5}$.

In Figure 5.5 we demonstrate the effect of increasing the number of substeps S on the error. Figure 5.5 a) shows estimated error against timestep Δt, for schemes using up to $S = 10$ substeps, while Figure 5.5 c) shows the same for schemes using S between 10 and 100. For sufficiently low Δt we have the predicted results, with the error being first order with respect to Δt, and decreasing as S increases. For Δt too large, this is not the case. Here the Krylov subspace dimension m is most likely the limiting factor as Assumption 5.3.1 becomes invalid. In Figure 5.5 b) and

d) we show the efficiency by plotting the estimated error against cputime. For Δt low enough that the substepping schemes are effective, the scheme with 10 substeps is the most efficient.

We can see the existence of an optimal S for efficiency, as predicted, in Figure 5.5 d), where the scheme using $S = 50$ is more efficient than the scheme using $S = 100$. Any increase in accuracy by increasing S from 50 to 100 is extremely small (indeed, it is unnoticeable in Figure 5.5 c), and not enough to offset the increase in cputime. In fact, Figure 5.5 d) shows that for this experiment the scheme using $S = 10$ is more efficient than both the $S = 50$ and $S = 100$ schemes. Figure 5.5 c) shows that the $S = 10$ scheme is also slightly more accurate than both. This is likely because at $S = 10$ the improvement in accuracy is already close to the limiting value, and greatly increasing S to 50 or 100 only accumulates rounding errors without further benefit. Figure 5.5 a) shows that the improvement from $S = 1$ to $S = 10$ is quite significant on its own.

In Figure 5.6 we compare the 2-step corrector scheme against two other second order exponential integrators, ETD2 and ROS2. Figure 5.6 a) shows estimated error against Δt, and we see that, like Figure 5.5 a), the Krylov recycling scheme does not function as intended above a certain Δt threshold; again this is due to the timestep being too large with respect to m. The standard exponential integrators do not have this problem, as their timesteps are driven by phipm.m, which takes extra (non-recycled, linear) substeps to achieve a desired error. Below the Δt threshold, the 2-step corrector scheme functions exactly as intended, exhibiting second order convergence and high accuracy. In Figure 5.6 b) we can observe that the 2-step corrector scheme is more efficient than the other two schemes for lower Δt, and of comparable efficiency for larger Δt.

It is interesting to compare Figure 5.5 a) and Figure 5.6 a) and note that the threshold Δt for the corrector scheme seems to be lower than for the substepping schemes.

In Figure 5.5 b) we can again see that for a fixed cputime, increasing S from 1 to 10 decreases error by roughly one order of magnitude; however Figure 5.5 d) shows no improvement in increasing S from 10 to 50. Comparing Figure 5.5 b) and Figure 5.6 b) shows that the second order corrector scheme can be almost three orders of magnitude more accurate for a fixed cputime than the first order recycling scheme with $S = 10$.

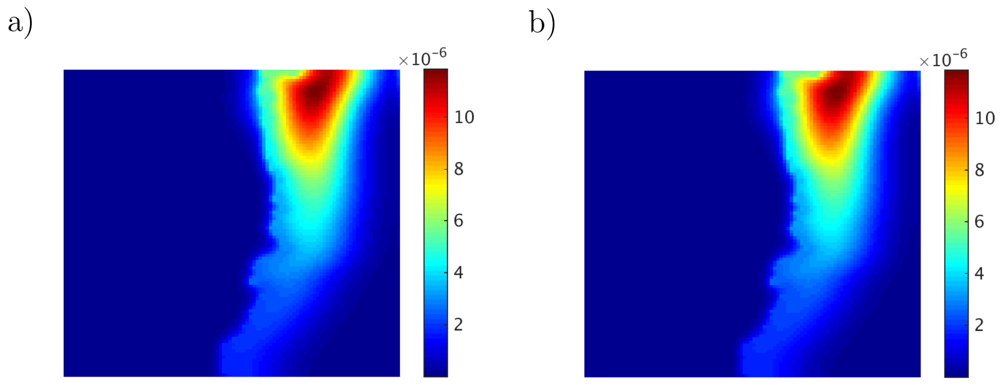


Figure 5.4: The final state of the fracture system with Langmuir type reaction. a) Result produced by the 2-step scheme with $\Delta t = 2.4 \times 10^{-4}$. b) Result produced by ETD2 with $\Delta t = 2.4 \times 10^{-5}$.

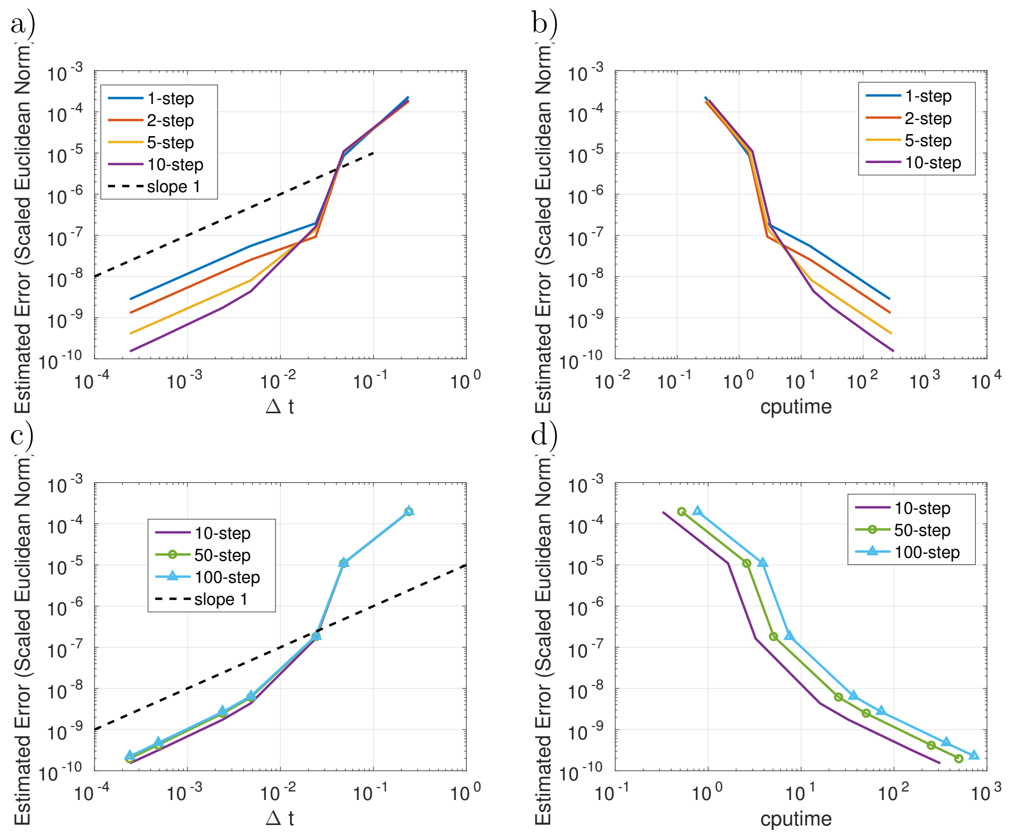


Figure 5.5: Results for the substepping schemes applied to the Langmuir type reaction system. a) and c) display Estimated error against timestep Δt. b) and d) display estimated error against cputime, showing efficiency. In c), points for the 50 step scheme are marked with circles, and points with the 100 step scheme are marked with triangles, to help distinguish the (very similar) results for the two schemes. This is also done in plot d) for consistency.

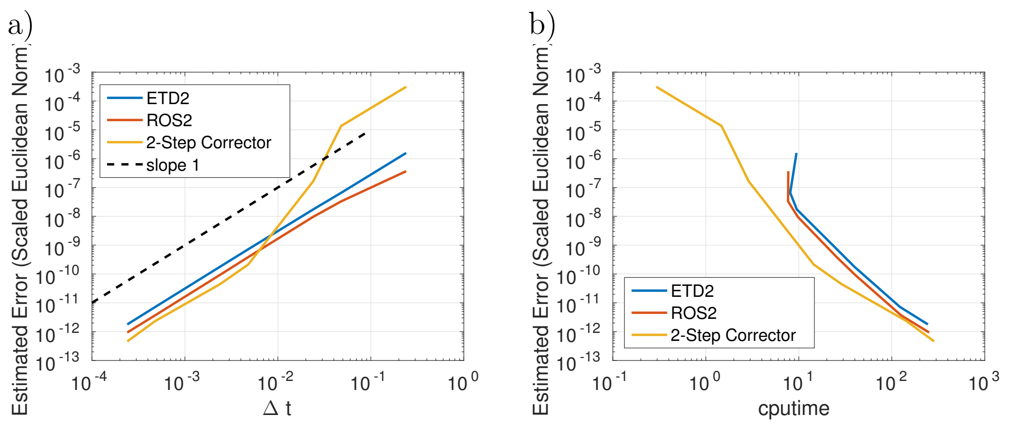


Figure 5.6: Langmuir type reaction system, Comparing the second order recycling-corrector scheme with ETD2 and ROS2. a) Estimated error against timestep Δt. b) Estimated error against cputime.

5.7 Conclusions

We have extended the notion of recycling a Krylov subspace for increased accuracy in the sense of [96]. We have applied this method to the first order ETD1 scheme and examined the effect of taking an arbitrary number S of substeps. The local error has been expressed in terms of S, and the expression shows that the local error will decrease with S down to a finite limit. Consideration of this has led to the suggestion that there exists an optimal S at which a maximal efficiency increase can be gained by using this method. Both of these have also been demonstrated with numerical experiments. An second order modification of this scheme which uses the additional information at the substeps to form a corrector has been developed and shown to be successful, with efficiency comparable to or slightly better than ETD2 and ROS2 in our tests.

The schemes currently rely on Assumption 5.3.1, essentially requiring that Δt be sufficiently small and m be sufficiently large, to be effective. Numerical experiments have shown how having Δt too large can cause the schemes to become inaccurate as the error of the initial Krylov approximation becomes significant. It is already well established how the Krylov approximation error can be controlled by adapting m and the use of non-recycling substeps. Applying these techniques to the schemes presented here in future work would allow them to be effective over wider Δt ranges. Currently, the schemes seem to be quite successful in providing accuracy improvements over ETD1, when Δt is not large enough to invalidate Assumption 5.3.1.

Chapter 6

Semi-Exponential Runge-Kutta Type Methods

6.1 Introduction

Again we consider the semilinear ODE, restated here for convenience,

$$\begin{aligned} \frac{du}{dt} &= Lu + F(t, u(t)) \\ u(0) &= u_0, \quad t \in [0, \infty), \end{aligned} \tag{6.1}$$

which has exact solution satisfying [29]

$$u(t_n + \Delta t) = u(t_n) + \Delta t \varphi_1(\Delta t L)(Lu_n + F) + \sum_{v=2}^{\Upsilon} (\Delta t L)^v \varphi_v(\Delta t) F^{(v-1)}(t_n, u(t_n)) + O(\Delta t^{\Upsilon+1}). \tag{6.2}$$

where the first two terms constitute the scheme ETD1, see §1.4.1. Higher order ETD schemes are constructed by approximating the successive φ terms in the sum. Typically a $j+1$ (local) order ETD or Runge-Kutta ETD (RKETD) scheme is constructed by finding a way to approximate the terms up to j in the sum. This is done to increase the stiff order of the scheme, as described in §1.5.3. The aim here is to develop a new class of RKETD schemes that, in principle, only require one matrix exponential evaluation. Specifically, we investigate here the effect of lifting this requirement to produce cheaper exponential integrator schemes, with possibly

worse stiff order properties. The idea is that each function φ_j is itself a power series in Δt, and evaluating the first term (order $O(1)$) is sufficient for a classical order condition. This type of idea is referred to as 'weakening' the order condition in [29], where an order condition for the scheme to match the term involving $\Delta t^j \varphi_j(\Delta t L)$, is replaced with a condition to match $\Delta t^j \varphi_j(0)$. We see that this makes the condition into one for just the leading term (independent of $\Delta t L$; of order $O(1)$). This still gives the desired (nonstiff) order but allows the rest of the φ function to escape into higher order terms, producing a slightly less accurate scheme as a result.
The design of the schemes is based on two key ideas. The first is mentioned above, that we shall make no attempt to match the successive φ_k terms in the sum in (6.2). Instead we shall work only with ETD1 approximations, and the resulting error terms of the desired order will be corrected for successively in an RK-type fashion. In other words, apart from the ETD1 part, every single order condition will be 'weakened'.

The second idea is to use the linearity of the ETD1 part, or more specifically of φ_1, to allow us to evaluate the ETD1 approximation at different intermediate points of the timestep, having only evaluated a single matrix exponential. This is done using a result derivable from Lemma 5.3.2. The specific form we need is

Corollary 6.1.1. *For a vector v, the φ_1 function satisfies*

$$2\delta t\varphi_1(2\delta tL)v = \delta t\varphi_1(\delta tL)v + \delta t\varphi_1(\delta tL)\left(L\delta t\varphi_1(\delta tL)v + v\right), \tag{6.3}$$

and similarly for the Krylov approximation $V_m\varphi_1(2\delta tH_m)V_m^T \approx \varphi_1(2\delta tL)$.

Proof. Let $\tau_1 = \tau_2 = \delta t$ in Lemma 5.3.2 and apply the definitions of $p_{\delta t}$ (5.4) and $\tilde{p}_{\delta t}$ (5.14). $\square$

Corollary 6.1.1 tells us that, having evaluated the matrix exponential function $\varphi_1(\delta tL)$ once for one substep value δt, it is possible to produce the value at successive integer multiples of δt, with only the cost of a few more matrix - vector multiplications. This can be used to evaluate ETD1 solutions at different parts of the timestep.

Corollary 6.1.1 holds for Krylov approximations, i.e. if we surround each φ function with the matrices V, V^T and replace L with H, which will be used in practice. We formulate the scheme here with the assumption of 'perfect' matrix exponentials - without reference to the error or the approximation in the initial ETD1 solve. Since in practice we use Krylov subspace projection, this means we use Assumption 5.3.1. We will use the code phipm [55] to ensure that this is met.

Work with similar aims include the RKETD type solvers of Tokman [31, 32, 33] which achieve a high classical order with few matrix exponential evaluations, and also [98], which moves a lot of the activity within the Krylov subspace, greatly reducing the overall work.

6.2 Method for the Semi-Exponential RK Scheme

6.2.1 General scheme

The general idea of the scheme is as follows.

- Starting with $u_n \approx u(t_n)$, the goal is as always to generate the approximation at the next timestep $u_{n+1} \approx u(t_n + \Delta t)$.
- We split the timestep Δt into S smaller substeps of length δt, such that $\Delta t = S\delta t$.
- Corollary 6.1.1 is then used to generate S ETD1 approximations, one at each substep. These are locally second order approximations to $u(t_n + \delta t), u(t_n + 2\delta t), \ldots, u(t_n + \Delta t)$.
- The form of the local error for these approximations is known and can be approximated to form correctors. The locally second order approximations are used together to calculate finite difference correctors, which can at best be third order accurate.
- After applying the correctors we now have locally third order approximations for $u(t_n + \delta t), u(t_n + 2\delta t), \ldots, u(t_n + \Delta t)$.

- The process can be repeated until we have S order accuracy.

For our approximations produced by the scheme, there are three things to keep track of. First the timestep level, which is familiar and represented with a subscript n, i.e. u_n. There is also the substep level, which we will represent with a subscript i, where $i \in \{0, \ldots, S\}$, and the correction stage level, which we will represent with a subscript j, where $j \in \{0, \ldots, S-1\}$. That is, let $u_{n,i,j}$ be the approximation at the nth timestep, on the ith substep (so that $u_{n,i,j} \approx u(t_n + i\delta t)$), after the application of the jth stage of correctors. Consider the start of the step as just described. The scheme uses 6.1.1 to first produce every $u_{n,i,0}$. The definition is

$$u_{n,i,0} \equiv u_n + i\delta t \varphi_1 (i\delta t L) (L u_n + F(t_n, u_n)) \approx u(t_n + i\delta t). \tag{6.4}$$

The scheme then proceeds to successively generate the corrected approximations $u_{n,i,1}, u_{n,i,2}$, and so on up to $u_{n,i,S-1}$. The general definition is

$$\begin{aligned} u_{n,i,j} \equiv\ & u_n + i\delta t \varphi_1 (i\delta t L) (L u_n + F(u_n)) \\ & + \text{Corrector from stage } j \\ & \approx u(t_n + i\delta t). \end{aligned} \tag{6.5}$$

The form of the correctors and how they are calculated discussed is discussed below. We now fix some notation. Recall that $u_{n,i,j} \in \mathbb{R}^K$, where K is the number of cells in the spatial discretisation. It will be convenient to group vectors like this together in matrices. We will use the hat $\hat{}$ notation for matrices composed of vectors like this. Let

$$\hat{u}_{n,j} = (u_{n,0,j}, \ldots, u_{n,S,j}) \in \mathbb{R}^{K \times S+1} \tag{6.6}$$

be the matrix which has $u_{n,i,j}$ as its ith column, so that $\hat{u}_{n,j}$ contains the approximation at every substep after the jth corrector stage. Next, let

$$\hat{F}_{n,j} = (F(u_{n,0,j}), \ldots, F(u_{n,S,j})) \in \mathbb{R}^{R \times S+1}$$

be the matrix which has $F(u_{n,i,j})$ as its ith column, i.e. containing the evaluation of the nonlinearity function F with the approximation at every substep after the jth corrector stage.

The corresponding exact versions are

$$u(\hat{t_n}) = \left(u\left(t_n\right), u\left(t_n + \delta t\right) \ldots, u\left(t_n + \Delta t\right)\right),$$

and

$$F(\hat{t_n}) = \left(F\left(u\left(t_n\right)\right), F\left(u\left(t_n + \delta t\right)\right), \ldots, F\left(u\left(t_n + \Delta t\right)\right)\right).$$

We will need to consider the time derivatives of $F(u(t_n))$. Let

$$F^{(\zeta)} = \frac{d^\zeta}{dt^\zeta} F(u(t_n)),$$

and let

$$\hat{dF} = \left(\Delta t F^{(1)}, \Delta t^2 F^{(2)} \ldots, \Delta t^S F^{(S)}\right) \in \mathbb{R}^{K \times S}.$$

6.2.2 Corrector for $\hat{u}_{n,j}$

Restating (6.2) using the notation introduced above, the true solution of the semi-linear equation (6.1) is,

$$u\left(t_n + i\delta t\right) = u\left(t_n\right) + i\delta t \varphi_1(i\delta t L) \sum_{v=2}^{\Upsilon} \left(i\delta t L\right)^v \varphi_v\left(i\delta t\right) F^{(v-1)}(t_n, u(t_n)) + O(\delta t^{\Upsilon+1}). \tag{6.7}$$

We can use this to express the local error of the $u_{n,i,0}$. See §1.5.1 for a description of local error. Let $u_{n,i}^{\text{loc}}$ be the result of replacing u_n with $u(t_n)$ in (6.4). That is,

$$u_{n,i}^{\text{loc}} \equiv u(t_n) + i\delta t \varphi_1\left(i\delta t L\right)\left(L u_n + F(u(t_n))\right). \tag{6.8}$$

The local error of $u_{n,i,0}$ is then given by

$$E_{n,i}^{loc} \equiv u\left(t_n + i\delta t\right) - u_{n,i}^{\text{loc}},$$

which we can now express using (6.7) and (6.8) as

$$E_{n,i}^{loc} = \sum_{\upsilon=2}^{\Upsilon} (i\delta t)^{\upsilon}\, \varphi_{\upsilon}\,(i\delta t L)\, F^{(\upsilon-1)}(t_n, u(t_n)) + O(\delta t^{\Upsilon+1}). \tag{6.9}$$

Using the series expansion of the $\varphi-$functions (1.15) this is,

$$E_{n,i}^{loc} = \sum_{\upsilon=2}^{\Upsilon} (i\delta t)^{\upsilon} \sum_{\eta=0}^{\infty} \frac{1}{(\upsilon+\eta)!} (i\delta t)^{\eta} L^{\eta} F^{(\upsilon-1)}(t_n, u(t_n)) + O(\delta t^{\Upsilon+1}).$$

We can move all the terms in the sum which are of order greater than Υ in δt into the $O(\delta t^{\Upsilon+1})$ to get,

$$E_{n,i}^{loc} = \sum_{\upsilon=2}^{\Upsilon} (i\delta t)^{\upsilon} \sum_{\eta=0}^{\Upsilon-\upsilon} \frac{1}{(\upsilon+\eta)!} (i\delta t)^{\eta} L^{\eta} F^{(\upsilon-1)}(t_n, u(t_n)) + O(\delta t^{\Upsilon+1}).$$

Changing the order of the sums and using $i\delta t = \frac{i}{S}\Delta t$ we get,

$$E_{n,i}^{loc} = \sum_{\eta=0}^{\Upsilon-2} (\Delta t)^{\eta+1} \sum_{\upsilon=2}^{\Upsilon-\eta} \frac{1}{(\upsilon+\eta)!} (\Delta t)^{\upsilon-1} \left(\frac{i}{S}\right)^{\upsilon+\eta} L^{\eta} F^{(\upsilon-1)}(t_n, u(t_n)) + O(\Delta t^{\Upsilon+1}). \tag{6.10}$$

Next we define the following vector,

$$\mathbf{x}_{\eta,i} \equiv \begin{pmatrix} \left(\frac{i}{s}\right)^{\eta+2} \frac{1}{(\eta+2)!} \\ \left(\frac{i}{s}\right)^{\eta+3} \frac{1}{(\eta+3)!} \\ \vdots \\ \left(\frac{i}{s}\right)^{\eta+S} \frac{1}{(\eta+S)!} \end{pmatrix}.$$

This allows us to express the inner sum in (6.10) in terms of $\hat{dF}$ and $\mathbf{x}_{\eta,i}$,

$$E_{n,i}^{loc} = \sum_{\eta=0}^{\Upsilon-2} (\Delta t)^{\eta+1}\, L^{\eta} \hat{dF}[:, 1 : \Upsilon-\eta-1] \mathbf{x}_{\eta,i}^{S}[1 : \Upsilon-\eta-1] + O(\Delta t^{\Upsilon+1}), \tag{6.11}$$

where we have used Matlab style indexing for matrices and vectors. In general for a matrix A, $A[x, y]$ denotes the entry at row x and column y of A. Numbers separated by a colon are a range; $\mathbf{x}_{\eta,i}^{S}[1 : \Upsilon-\eta-1]$ in (6.11) denotes the entries 1 through $\Upsilon-\eta-1$ of the vector $\mathbf{x}_{\eta,i}^{S}$. In the general expression $A[x, y]$, x and y can be replaced with ranges or a single colon. A single colon means ‘all rows’, or ‘all columns’; $\hat{dF}[:, 1 : \Upsilon - \eta - 1]$ in (6.11) denotes every row of columns one through $\Upsilon - \eta - 1$ of $\hat{dF}$.

We are interested in a matrix determined by

$$\hat{E}_n^S \equiv \left(E_{n,0}^{loc}, E_{n,1}^{loc}, \ldots, E_{n,S}^{loc}\right), \tag{6.12}$$

to use as a corrector for $\hat{u}_{n,i}$ (6.6). The semi-exponential RK scheme proceeds by producing successively more accurate approximations to the matrix $\hat{E}_n^S$ in (6.12). From (6.11) we have an expression for each column of $\hat{E}_n^S$, which we will use to define these correctors. Define the matrix $\hat{x}_j^S$ as,

$$\hat{x}_j^S \equiv \left(\mathbf{0}, \mathbf{x}_{j,0}^S, \mathbf{x}_{j,1}^S, \ldots, \mathbf{x}_{j,S}^S\right) \in \mathbb{R}^{S-1 \times S+1},$$

where $\mathbf{0} \in \mathbb{R}^{S-1}$ is the column vector with all entries zero. Then from (6.11), we see that the ith column of the matrix given by

$$\sum_{\eta=0}^{\Upsilon-2} \Delta t^{\eta+1} L^{\eta} \hat{dF}[:, 1 : \Upsilon - \eta - 1] \hat{x}_\eta^S [1 : \Upsilon - \eta - 1, :]$$

is $E_{n,i}^{\text{loc}} + O(\Delta t^{\Upsilon+1})$. We have the shown the following.

Lemma 6.2.1. *The local error of $\hat{u}_{n,j}$ can be expressed as follows.*

$$\begin{aligned} \hat{E}_n^S &\equiv \left(E_{n,0}^{loc}, E_{n,1}^{loc}, \ldots, E_{n,S}^{loc}\right) \\ &= \sum_{\eta=0}^{\Upsilon-2} \Delta t^{\eta+1} L^{\eta} \hat{dF}[:, 1 : \Upsilon - \eta - 1] \hat{x}_\eta^S [1 : \Upsilon - \eta - 1, :] + O(\Delta t^{\Upsilon+1}) \end{aligned} \tag{6.13}$$

We have written this result in terms of generally oversized dF and $\hat{x}_\eta^S$ for practical convenience, which we now explain. Over the S internal stages of the scheme, (6.13) is used to produce correctors up to second order for the first stage, third order for the second stage, and so on, up to order $S + 1$ for the final stage. It is therefore better to calculate once and for all each $\hat{x}_\eta^S \in \mathbb{R}^{S \times S+1}$ suitable for use at the $S + 1$

stage, and use the appropriate submatrices for preceding stages. The way we have written Lemma 6.2.1 makes it explicit which columns of the $\hat{x}^S_\eta$ are to be used.

6.2.3 Differencing to Approximate dF

Given the ability to calculate $\hat{dF}$, we could use Lemma 6.2.1 to extrapolate $\hat{U}_{n,s}$ to arbitrary order. This is not possible in general, so we approximate the derivatives using finite differences.

First we consider a linear combination,

$$\chi \equiv \sum_{i=0}^{S} \alpha_i F\left(t + \frac{i}{S}\Delta t\right).$$

Taylor expansion gives

$$\chi = \sum_{i=0}^{S} \beta_i \Delta t^i F^{(i)}(t) + O(\Delta t^{S+1}),$$

where the relationships between the α and β parameters are given by

$$\begin{pmatrix} 1 & \left(\frac{1}{S}\right)^0 & \left(\frac{2}{S}\right)^0 & \dots & \left(\frac{S}{S}\right)^0 \\ 1 & \left(\frac{1}{S}\right)^1 & \left(\frac{2}{S}\right)^1 & \dots & \left(\frac{S}{S}\right)^1 \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \dots \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \dots \end{pmatrix}$$

Let $\underline{\alpha}_k$ be the vector of α coefficients that solves this matrix equation when all the β values are zero except β_k. Then we see that $F_k \underline{\alpha}_k = \Delta t^k F^{(k)} + O(\Delta t^{S+1})$. Then we can define a differencing matrix $\hat{D}$ as the matrix having $\underline{\alpha}_k$ as its kth column. This provides a way to approximate dF.

Remark 6.2.2. *Let $\hat{D} \equiv (\underline{\alpha}_1, \underline{\alpha}_2, \dots, \underline{\alpha}_S)$, then,*

$$\hat{F}(t_n)\hat{D} = \hat{dF} + O(\Delta t^{S+1}).$$

If

$$\hat{F}_{n,j} = \hat{F}(t_n) + O(\Delta t^{\Upsilon+1}),$$

with $\Upsilon < S$, *then*

$$\hat{F}_{n,j}\hat{D} = \hat{dF} + O(\Delta t^{\Upsilon+1}).$$

It is straightforward to mechanically produce $\hat{D}$ once and for all for a given scheme; we implemented a simple Matlab function to do so.
Remark 6.2.2 shows how we can use $\hat{F}_{n,s}$ to approximate dF, since this is what is available instead of F_k. The limit on the accuracy of $\hat{F}_{n,s}$ limits the accuracy of the approximation to $\hat{dF}$ and thus the accuracy of the correction to $\hat{u}_{n,j}$. We proceed with the standard Runge-Kutta methodology here, building increasingly accurate approximations using previous approximations.

6.3 Matrices for Second, Third and Fourth Order Schemes

We simply state the D and X matrices for the schemes here.

6.3.1 Second Order

The second order scheme uses $S = 2$ substeps. The matrices are,

$$\hat{D} = \begin{pmatrix} -1 \\ 1 \end{pmatrix},$$

and,

$$\hat{x}_0^2 = \begin{pmatrix} 0 & \frac{1}{2!} \\ 0 & \frac{1}{3!} \end{pmatrix}.$$

6.3.2 Third Order

The third order scheme uses $S = 3$ substeps. The matrices are;

$$\hat{D} = \begin{pmatrix} -3 & 4 \\ 4 & -8 \\ -1 & 4 \end{pmatrix},$$

$\hat{x}_0^3$ is ,

$$\hat{x}_0^3 = \begin{pmatrix} 0 & \left(\frac{1}{2}\right)^2 \frac{1}{2!}, & \frac{1}{2!} \\ 0 & \left(\frac{1}{2}\right)^3 \frac{1}{3!}, & \frac{1}{3!} \end{pmatrix}$$

and $\hat{x}_1^3$ is

$$\hat{x}_1^3 = \begin{pmatrix} 0 & \left(\frac{1}{2}\right)^2 \frac{1}{3!}, & \frac{1}{3!} \\ 0 & \left(\frac{1}{2}\right)^3 \frac{1}{4!}, & \frac{1}{4!} \end{pmatrix}.$$

6.3.3 Fourth Order

The fourth order scheme uses $S = 4$ substeps. The matrices are;

$$\hat{D} = \begin{pmatrix} -5.5 & 18 & -27 \\ 9 & -45 & 81 \\ -4.5 & 36 & -81 \\ 1 & -9 & 27 \end{pmatrix}.$$

$\hat{x}_0^4$ is,

$$\hat{x}_0^4 = \begin{pmatrix} 0 & \left(\frac{1}{3}\right)^2 \frac{1}{2!}, & \left(\frac{2}{3}\right)^2 \frac{1}{2!}, & \frac{1}{2!} \\ 0 & \left(\frac{1}{3}\right)^3 \frac{1}{3!}, & \left(\frac{2}{3}\right)^3 \frac{1}{3!}, & \frac{1}{3!} \\ 0 & \left(\frac{1}{3}\right)^4 \frac{1}{4!}, & \left(\frac{2}{3}\right)^4 \frac{1}{4!}, & \frac{1}{4!} \end{pmatrix},$$

and $\hat{x}_1^4$ is

$$\hat{x}_1^4 = \begin{pmatrix} 0 & \left(\frac{1}{3}\right)^2 \frac{1}{3!}, & \left(\frac{2}{3}\right)^2 \frac{1}{3!}, & \frac{1}{3!} \\ 0 & \left(\frac{1}{3}\right)^3 \frac{1}{4!}, & \left(\frac{2}{3}\right)^3 \frac{1}{4!}, & \frac{1}{4!} \\ 0 & \left(\frac{1}{3}\right)^4 \frac{1}{5!}, & \left(\frac{2}{3}\right)^4 \frac{1}{5!}, & \frac{1}{5!} \end{pmatrix}$$

and $\hat{x}_2^4$ is

$$\hat{x}_2^4 = \begin{pmatrix} 0 & \left(\frac{1}{3}\right)^2 \frac{1}{4!}, & \left(\frac{2}{3}\right)^2 \frac{1}{4!}, & \frac{1}{4!} \\ 0 & \left(\frac{1}{3}\right)^3 \frac{1}{5!}, & \left(\frac{2}{3}\right)^3 \frac{1}{5!}, & \frac{1}{5!} \\ 0 & \left(\frac{1}{3}\right)^4 \frac{1}{6!}, & \left(\frac{2}{3}\right)^4 \frac{1}{6!}, & \frac{1}{6!} \end{pmatrix}.$$

6.3.4 Algorithm for the Scheme

Using Remark 6.2.2, we can use $\hat{F}_{n,s}$ to generate a finite difference approximation to the local error given by (6.13),

$$\begin{aligned} &\sum_{\eta=0}^{\Upsilon-2} \Delta t^{\eta+1} L^{\eta} \hat{F}_{n,s} \hat{D}[:, \Upsilon-\eta-1] \hat{x}_{\eta}^{S}[1:\Upsilon-\eta-1, :] \\ &= \sum_{\eta=0}^{\Upsilon-2} \Delta t^{\eta+1} L^{\eta} d\hat{F}[:, 1:\Upsilon-\eta-1] \hat{x}_{\eta}^{S}[1:\Upsilon-\eta-1, :] + O(\Delta t^{\Upsilon+1}), \end{aligned} \tag{6.14}$$

if

$$\hat{F}_{n,j} = \hat{F}(t_n) + O(\Delta t^{\Upsilon+1}).$$

Algorithm 3 demonstrates one step of the scheme. Line 1 is the formation of $\hat{u}_{n,0}$ from the previous step's result $\hat{u}_n$ using Remark 6.1.1. Lines 2 through 5 are the loop over each of the S internal stages. At each internal stage a new $\hat{u}_{n,s}$ is produced using the $\hat{u}_{n,s-1}$ from the previous stage. Each $\hat{u}_{n,s}$ is one order more accurate than the previous. At line 3, $\hat{F}_{n,s}$ is produced simply by the action of F on each column of $\hat{u}_{n,s-1}$. Line 4 is the application of the corrector already discussed at length. Finally, line 6 takes the final column of $\hat{u}_{n,S}$, which approximates $u(t+\Delta t)$, as the final result for this timestep, $\hat{u}_{n+1}$. The scheme then moves on to another timestep.

```
   Data: $u_n$ from previous step, Precomputed: $\hat{D}$ and $\hat{x}_1^S, \ldots, \hat{x}_{S-1}^S$.
1  $\hat{u}_{n,0} = (u_{n,0,0}, \ldots, u_{n,S,0})$ ;
2  for  $s = 1 : S-1$ do
3      $\hat{F}_{n,s} = F(\hat{u}_{n,s-1})$ ;
4      $\hat{u}_{n,s} = \hat{u}_{n,0} + \sum_{\eta=0}^{s-1} \Delta t^{\eta+1} L^{\eta} \hat{F}_{n,s} \hat{D}[:, \Upsilon-\eta] \hat{x}_{\eta}^{S}[1:\Upsilon-\eta, :]$ ;
5  end
6  $u_{n+1} = \hat{u}_{n,S-1}(:, S+1)$ ;
```

Algorithm 3: The algorithm for one timestep of the semi RK scheme

6.4 A Third order example

As an example, we look at one step of the third order scheme in detail. The matrices for the scheme are stated above in §6.3.2.

First the initial substep values are generated by Remark 6.1.1. That is,

$$u_{n,0,0} = u_{n-1}$$

$$u_{n,1,0} = u_n + \frac{1}{2}\Delta t\varphi_1\left(\frac{1}{2}\Delta tL\right)(Lu_n + F)$$

$$u_{n,2,0} = u_n + \Delta t\varphi_1(\Delta tL)(Lu_n + F),$$

and,

$$\hat{u}_{n,0} = (u_{n,0,0}, u_{n,1,0}, u_{n,2,0}).$$

Also,

$$\hat{F}_{n,0} = (F(u_{n,0,0}), F(u_{n,1,0}), F(u_{n,2,0})).$$

The local errors for each of these are known;

$$E^{loc}_{n,1} = \frac{\Delta t^2}{4}\frac{1}{2}\frac{dF}{dt}(t_n) + \frac{\Delta t^3}{8}\frac{1}{6}L\frac{dF}{dt}(t_n) + \frac{\Delta t^3}{8}\frac{1}{6}\frac{d^2F}{dt^2}(t_n) + O(\Delta t^4)$$

and

$$E^{loc}_{n,1} = \Delta t^2\frac{1}{2}\frac{dF}{dt}(t_n) + \Delta t^3\frac{1}{6}L\frac{dF}{dt}(t_n) + \Delta t^3\frac{1}{6}\frac{d^2F}{dt^2}(t_n) + O(\Delta t^4),$$

then the local error is

$$\hat{E}^S_n = \left(0, E^{loc}_{n,1}, E^{loc}_{n,1}\right) + O(\Delta t^4),$$

from (6.2). This corresponds to (6.13),

$$\hat{E}^S_n = \Delta t\hat{dF}\hat{x}^3_0 + \Delta t^2 LdF[:,1]\hat{x}^3_0[1,:] + O(\Delta t^4). \tag{6.15}$$

The goal is to use the right hand side of this equation as a corrector. We can approximate $\hat{dF}$ using $\hat{F}_{n,0}\hat{D}$. The accuracy of the approximation is limited by how well $\hat{F}_{n,0}$ approximates $\hat{dF}$, i.e. $\hat{F}_{n,0} = F_n + O(\Delta t^2)$, so $\hat{F}_{n,0}\hat{D} = \hat{dF} + O(\Delta t^2)$.

Attempting to approximate (6.15) would give us

$$\Delta t\hat{F}_{n,0}\hat{D}\hat{x}_0^3 + \Delta t^2 L\hat{F}_{n,0}\hat{D}[:,1]\hat{x}_0^3[1,:] = \Delta t dFX_1 + \Delta t^2 LdF[:,1]\hat{x}_0^3[1,:] + O(\Delta t^3).$$

There is no point attempting to eliminate the third order terms in (6.15) (i.e. $\Delta t^2 L\hat{F}_{n,0}\hat{D}(:,1)X_2(1,:)$) in this way due to the additional third order error added by the approximation $\Delta t\hat{F}_{n,0}\hat{D}\hat{x}_0^3$. Instead we eliminate only the second order terms. Focusing on the second order terms, (6.15) becomes

$$E_{n,1}^{loc} = \Delta t d\hat{F}[:,1]\hat{x}_0^3[:,1] + O(\Delta t^3).$$

Attempting to approximate this with $\hat{F}_{n,0}\hat{D}$ is successful,

$$\Delta t\hat{F}_{n,0}\hat{D}[:,1]\hat{x}_0^3[:,1] = \Delta t d\hat{F}[:,1]\hat{x}_0^3[:,1] + O(\Delta t^3),$$

and thus we can make the new third order approximation

$$\hat{u}_{n,1} \equiv \hat{u}_{n,0} + \Delta t\hat{F}_{n,0}\hat{D}[:,1]\hat{x}_0^3[:,1],$$

which has local error $O(\Delta t^3)$.

This concludes stage $s = 1$. Stage $s = 2$ begins with a new generation of F evaluations,

$$\hat{F}_{n,1} \equiv (F(u_{n,0,1}), F(u_{n,1,1}), F(u_{n,2,1})) = \hat{F}(t_n) + O(\Delta t^3).$$

Now we can approximate (6.15) completely,

$$\Delta t\hat{F}_{n,1}\hat{D}\hat{x}_0^3 + \Delta t^2 L\hat{F}_{n,1}\hat{D}[:,1]\hat{x}_1^3[1,:] = \Delta t d\hat{F}\hat{x}_0^3 + \Delta t^2 Ld\hat{F}[:,1]\hat{x}_1^3[1,:] + O(\Delta t^4).$$

Thus we define,

$$\hat{u}_{n,1} \equiv \hat{u}_{n,0} + \Delta t\hat{F}_{n,1}\hat{D}\hat{x}_0^3 + \Delta t^2 L\hat{F}_{n,1}\hat{D}[:,1]\hat{x}_1^3[1,:]$$

which satisfies has local error $O(\Delta t^4)$.

This concludes stage $s = 2$. We cannot proceed further since we are limited by the accuracy of the approximation $\hat{D}\hat{F}(t_n)$, i.e. we would require more internal steps to generate any better finite difference approximations. Thus we produce the final result from the timestep as the best approximation of $u(t_n + \Delta t)$,

$$u_{n+1} \equiv \hat{u}_{n,2}[:,3].$$

6.5 Numerical Results

For error estimation we used the 4th order exponential Runge Kutta method of Strehmel and Weiner as quoted in [29], table (5.17), and given originally in [99]. For shorthand we refer to this method as 'RKETD4-sw' here. Our implementation of RKETD4-sw calls phipm.m four times per timestep, once for each internal stage. For our comparison solves we gave phipm.m the parameters 30 and 10^{-15} for initial Krylov subspace dimension and substep error tolerance, respectively.

We compare our methods with a third order exponential Runge Kutta method, which is given in table (5.8) of [29], and with RKETD4-sw. We refer to the third order method as 'RKETD3' here, and its implementation is similar to RKETD4-sw, with three calls to phipm.m each timestep. For these tests, phipm.m was given parameters 30 and 10^{-9} for initial Krylov subspace dimension and substep error tolerance, respectively.

For our schemes, which we refer to as 'Semi-RKETD3' for the third order scheme and 'Semi-RKETD4' for the fourth order, the Krylov subspace dimension $m = 30$ was used.

6.5.1 Allen-Cahn type Reaction Diffusion

This is the same system, domain and grid as §5.6.1. We performed tests with two variations of the system, one with $D(x) = 1$ as in §5.6.1, and one with $D(x) = 0.01$. The comparison solve used $\Delta t = 10^{-6}$.

The results for the small D system are in Figure 6.1, while the results for the larger

D system are in Figure 6.2. Plot a) in both figures shows the error convergence. Semi-RKETD3 is order three as expected for both systems; however Semi-RKETD4 is only fourth order for the low D system; its order has been reduced to three as might be expected as the system is more stiff.

Plot b) in both Figures 6.1 and 6.2 shows error against cputime, demonstrating efficiency. For the small D system, 6.1 b) shows Semi-RKETD4 is more efficient than RKETD4-sw, due to having almost identical error (plot a) but faster execution time. For the large D system, 6.2 b) shows Semi-RKETD4 is only more efficient than RKETD4-sw for low cputime (corresponding to low Δt). Semi-RKETD4 is still faster, but also less accurate due to the order reduction (plot a).

In both systems Semi-RKETD3 is more efficient than RKETD3 (Figures 6.1 and 6.2 b). This is because Semi-RKETD3 has faster execution time but almost identical error (Figures 6.1 and 6.2 a). We had expected Semi-RKETD3 to be less accurate than a corresponding third order RKETD method, and indeed it is in both cases, but the difference is slight here.

We can compare Figure 6.2 b) with Figure 5.3 b), from chapter 5. Comparing the second order corrector scheme from Chapter 5 with the second order Semi-RKETD2 from this chapter, the two plots indicate that the corrector scheme is more accurate than Semi-RKETD2. This may be because of the heuristic attempt to eliminate part of the leading term of the local error in the construction of the corrector scheme. The third order schemes, Semi-RKETD3 and Semi-RKETD4, are more accurate than the second order corrector scheme for a fixed cputime.

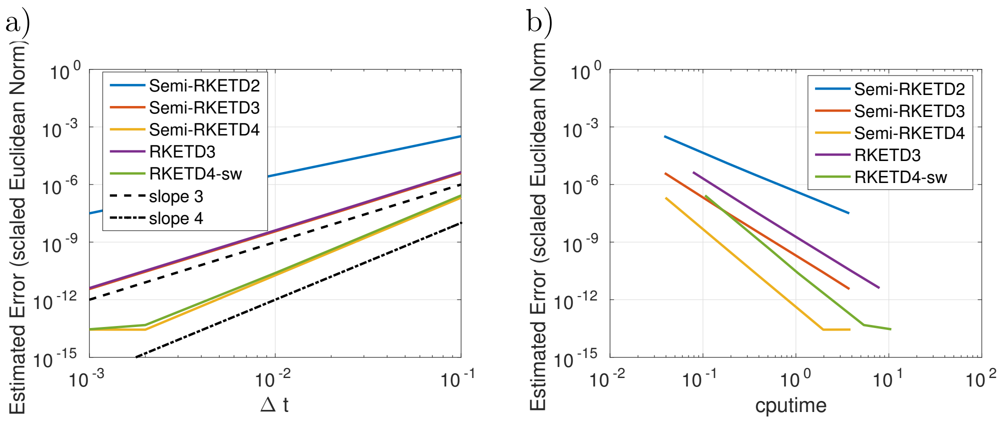


Figure 6.1: Results for the 3rd and 4th order Semi-ETDRK schemes applied to the Allen-Cahn type system with uniform diffusivity $D = 0.01$. a) Estimated error against timestep Δt. b) Estimated error against cputime, showing efficiency.

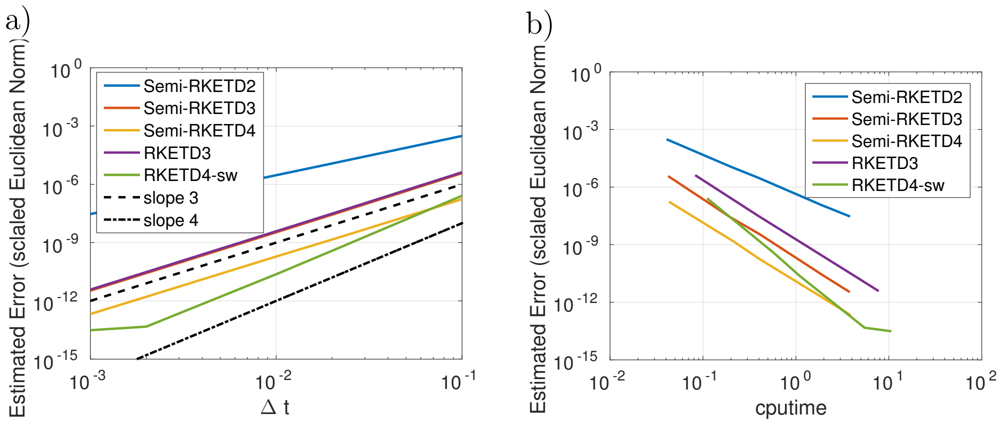


Figure 6.2: Results for the 3rd and 4th order Semi-ETDRK schemes applied to the Allen-Cahn type system with uniform diffusivity $D = 1$. a) Estimated error against timestep Δt. b) Estimated error against cputime, showing efficiency.

6.5.2 Fracture system

We use the same example as §5.6.2. Our comparison solve uses $\Delta t = 2.4 \times 10^{-5}$. Figure 6.3 shows the results. Again we see the order reduction phenomenon as Semi-RKETD4 is third order. There is also a Δt above which Semi-RKETD3 and Semi-RKETD4 are highly inaccurate. From Figure 6.3 we see that Semi-RKETD4 is of comparable efficiency or slightly better than RKETD4-sw when Δt is low enough that it is stable.

We can compare Figure 6.3 b) with Figure 5.6 b) from the previous chapter. We can compare the second order schemes; the 2-step corrector scheme from Chapter 5 and Semi-RKETD2 from this chapter. For larger cputime values, e.g. 10^2, the efficiency is roughly similar. For smaller cuptime values, e.g. around 10^0, Semi-RKETD2 appears to produce smaller error.

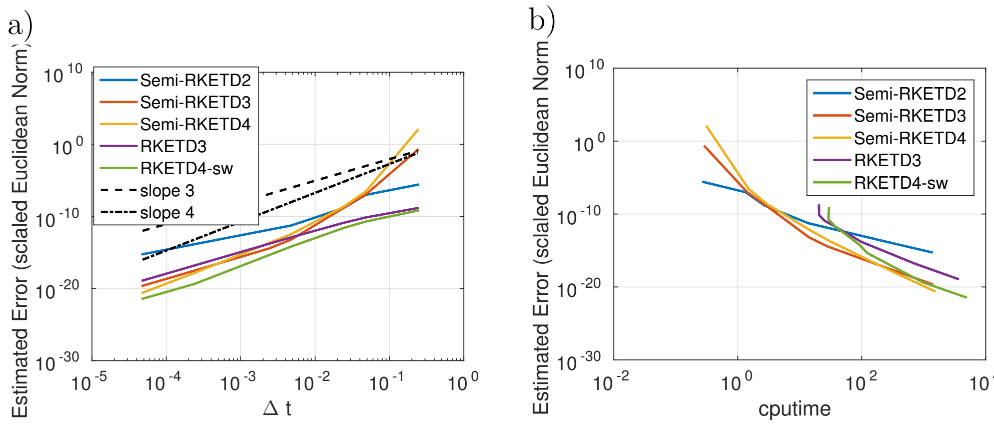


Figure 6.3: Results for the 3rd and 4th order Semi-ETDRK schemes applied to the fracture system with Langmuir reaction. a) Estimated error against timestep Δt. b) Estimated error against cputime, showing efficiency.

6.6 Conclusions

The schemes developed here are 'semi-exponential' integrators; some of the advantageous stiff order properties of regular exponential integrators schemes have been deliberately abandoned in order to produce faster schemes. This is reflected in our numerical experiments - the new schemes will typically not converge with order greater than three in most circumstances, whereas regular exponential integrators will. For moderately low Δt, the new schemes exhibit efficiency comparable to existing RKETD schemes. Some of our results indicate the potential for rapid growth of error, or even instability, for higher Δt ranges. This is not surprising as the new schemes can be expected to sacrifice some of the superior stability properties of full exponential integrator schemes. When applied in the appropriate Δt range and to suitable (perhaps not highly stiff) systems, the new schemes could be competitive, providing increased efficiency due to reduced cputimes. Future work could include adaptive schemes which prevent potential instability problems.

Chapter 7

Schemes that are asynchronous due to communication delay

7.1 Introduction

A different form of asynchronicity, compared to that introduced in Chapter 2, has the potential to play a large part in the design of schemes for very high scale parallel computing. For machines with very many cores (at the so called exa- or peta-scales), communication between the processor cores can become the limiting factor for efficiency, and it has been suggested that new numerical schemes should be designed to take fuller advantage of the exascale, which are robust to potentially large, random communication delays [100].

The schemes examined in Chapter 1 do not lend themselves naturally to large scale paralellization in this way (though a parallel version of a DES code for plasma simulation does exist and can be considered [69]), however our experiments in Chapter 3 do demonstrate that it is possible for schemes to exist that are robust to significant and uncontrolled asynchronicity. Some first steps towards schemes with the suited properties for adaptation to the exascale were taken in [101], where analysis and experimentation were performed on asynchronous schemes for linear PDEs in one dimension. Essentially a standard domain decomposition (e.g., [102]) was assumed, with the spatial domain mapped onto separate processing elements, with overlap regions that allow information from one subdomain to affect the neighbouring subdomain, and so on. Typically after each timestep to evolve its subdomain, a processing element must transmit its new data in its overlap regions to the processing elements containing the neighbouring subdomains, and await the latest information from its neighbours about their evolution. In essence the processing elements must synchronise; a potentially very expensive process for highly parallel machines. The innovation in [101] was to allow random communication delays, where a processing element may with some probability use overlap data from a previous step and not wait for fresh data from a processing element containing a neighbouring subdomain. The first schemes analysed in [101] were straightforward finite difference discretisations for spatial derivatives and Euler-type first order stepping in time. These were applied to linear advection-diffusion PDEs. With the random communication delays, these were shown to have the same sufficient stability constraints as without the delays. The error was also analysed and additional terms of order $\frac{\Delta t}{\Delta x^2}$ were shown to be present on overlap nodes. Thus order reduction was shown to be theoretically possible in overlap regions if the ratio $\frac{\Delta t}{\Delta x^2}$ is kept constant, which can be avoided, and also modifications to the scheme to avoid this problem were introduced. Overall it was demonstrated that robustness to communication delays is quite possible when forward Euler timestepping is used.

Here we examine schemes for linear PDEs in one dimension (particularly advection-diffusion systems), which are discretised in space by finite difference or finite volume methods (equivalent in 1D, when the numerical flux is taken to be a simple two cell finite difference type as used in earlier chapters), and then stepped forward in time by taking the matrix exponential of the resulting ODE system. The same domain decomposition with communication delays is applied as in [101], discussed above. This is inspired by the recent popularity of Exponential Integrators for solving semi-linear ODE systems (see [22], [34] for example, and §1.4 §5.1 here). It is now possible to efficiently and accurately approximate the action of a matrix exponential on a vector by methods based on Krylov subspace projection [55] or Leja point interpolation [52]. Thus it is of interest how schemes based on matrix exponentials or their approximations might adapt to asynchronous large scale parallelization.

We describe domain decomposed exponential timestepping schemes with and without random communication delay in §7.2. We present several numerical examples of these schemes in §7.3 and some conclusions in §7.4.3. We do not analytically prove any properties for the schemes described here, but it should be straightforward to reproduce something like the analysis in [101] given the formulations given in §7.2.

7.2 Method

We start by describing the basic domain decomposition. Consider a one dimensional domain Ω consisting of N nodes. Decompose this domain into D subdomains, which do not overlap and which together cover the whole of Ω. Consider the subdomains indexed by d, and let d_s be the starting node of domain d, and d_e be the ending node of domain d, so that domain d contains the nodes indexed by $\{d_s, d_s+1, \ldots, d_e\}$. Assign to D different processing elements one of the subdomains. The processing element is then responsible for advancing its subdomain from the initial condition data at time $t = 0$, to the final state at time $t = T$. Each processing element calculates the evolution of its subdomain in parallel with the others.

Of course, the state of the system in one subdomain at any given time can affect the evolution of the system in other subdomains at any future time, so the evolution of all the D different subdomains cannot proceed completely in parallel. Consider extensions of each domain d which overlap with neighbouring domains, and let o_l be the (fixed) length of the overlap regions. Refer to Figure 7.1; and consider a subdomain d to be adjacent to another subdomain d'. Define the start and end nodes of the extension of domain d by $d_S \equiv d_s - o_l$ and $d_E \equiv d_E + o_l$ respectively. The domain d then uses data from its extension $\{d_S, \ldots, d_E\}$ to calculate the data at the next timestep in the interior of d, $\{d_s, \ldots, d_e\}$. The overlap regions $\{d_S, \ldots d_s - 1\}$ and $\{d_e + 1, \ldots d_E\}$ lie within the interiors of adjacent subdomains, for example the right overlap region of domain d lies within subdomain d' in Figure 7.1. After each timestep, the overlap regions of a subdomain are updated with the appropriate data from the interior of neighbouring subdomains. That is, the data is communicated between the appropriate processing elements and stored in local memory there, to

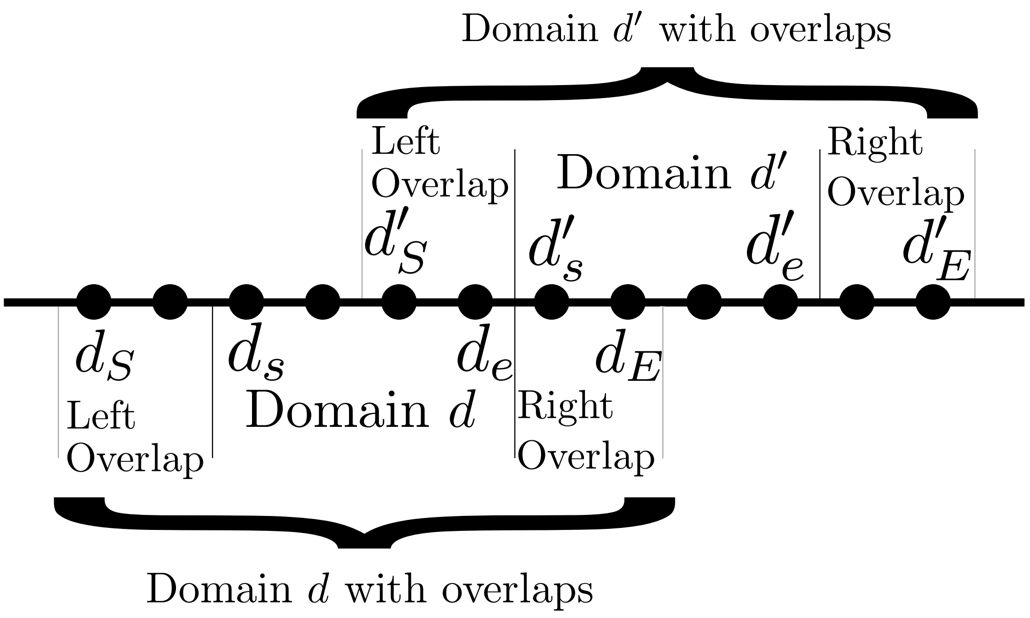


Figure 7.1: Domains and overlaps for simple one dimensional domain decomposition. Illustrates our terminology. Here we show two adjacent domains labelled d and d'. The size of a domain here is four nodes, and the overlap length, o_l, is two nodes.

be used for the next timestep. In this way the state of the system in one subdomain affects the system in other subdomains in the future.

We now describe a simple domain decomposition scheme for linear PDEs which is based on locally evaluating the matrix exponential. This can be applied to any linear PDE in principle; we focus here on the diffusion equation,

$$\frac{du}{dt} = \nabla \cdot D \nabla u. \tag{7.1}$$

We may consider the spatial discretisation as a finite volume type, with each node as the centre of a cell and cell faces being equidistant between neighbouring nodes. We may express the relationship between two adjacent nodes through a face k by the connection matrix L_k (already introduced in §2.4). The spatial semi-discretisation of the PDE will then be

$$\frac{d\mathbf{u}}{dt} = \sum_{\text{faces } k} L_k \mathbf{u}. \tag{7.2}$$

Define the sum of connection matrices $L_{k_1}, \dots L_{k_2}$ to be L_{k_1,k_2}, i.e.,

$$L_{k_1,k_2} = \sum_{k=k_1}^{k_2} L_k, \tag{7.3}$$

and the solution vector restriction $\mathbf{u}_{k_1,k_2}$ to be

$$\mathbf{u}_{k_1,k_2} = [0, \ldots, u_{k_1}, \ldots, u_{k_2}, 0, \ldots]^T. \tag{7.4}$$

Then we consider part of the ODE system (7.2) on a domain d,

$$\frac{d\mathbf{u}_{d_S,d_E}}{dt} = L_{d_S+1,d_E}\mathbf{u}_{d_S,d_E}. \tag{7.5}$$

Note that the subscript on L_{d_S+1,d_E} follows since, in this one dimensional setting, the connection matrix L_k for face acts on nodes $k-1$ and k. Thus the accumulated connection matrix L_{d_S+1,d_E} will act on every node in $\mathbf{u}_{d_S,d_E}$.
Since L_{d_S,d_E} is nonzero only inside a block and $\mathbf{u}_{d_S,d_E}$ has only nonzero entries in a range, the solution to (7.5) (which is a matrix exponential) can be found using a smaller matrix than the whole domain, and thus more cheaply. Note that the way connection matrices are defined means that we are essentially imposing Neumann boundary conditions on each domain by default. That is, there are zero Neumann boundary conditions (no-flow) on the edge of each domain's overlaps. Now we define the update rule for the subdomain d as

$$\mathbf{u}^{n+1}_{d_S,d_E} = e^{\Delta t L_{d_S+1,d_E}}\mathbf{u}^n_{d_S,d_E}. \tag{7.6}$$

After each timestep the overlap regions for subdomain d are overwritten by the appropriate data in the neighbouring domains $(d-1)$ and $(d+1)$ as follows,

$$\begin{aligned}\mathbf{u}_{d_S,d_E}(d_S : d_s - 1) &\leftarrow \mathbf{u}_{(d-1)_S,(d-1)_E}(d_S : d_s - 1)\\ \mathbf{u}_{d_S,d_E}(d_e + 1 : d_E) &\leftarrow \mathbf{u}_{(d+1)_S,(d+1)_E}(d_e + 1 : d_E),\end{aligned} \tag{7.7}$$

where we have used Matlab notation for vector indexing. Finally after n timesteps on each domain we can reconstruct the whole system as

$$\mathbf{u}^n = \sum_{d=1}^{D} I_{d_s,d_e}\mathbf{u}^n_{d_S,d_E}, \tag{7.8}$$

where I_{d_s,d_e} is the restricting identity matrix with

$$I_{d_s,d_e} = \left(\begin{array}{ccccc} 0 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 0 \end{array} \right) \begin{array}{l} \} \text{ Rows } 0 \text{ to } d_s - 1 \\ \\ \} \text{ Rows } d_s \text{ to } d_e \\ \\ \} \text{ Rows } d_e + 1 \text{ to } N \end{array} .$$

Now the modification of the scheme which allows communication delay is simply to allow the possibility of (7.7) not happing on some occasions; thus allowing that sometimes interprocessor communication does not happen fast enough and stale data is used during some updates on some processors. This would correspond in practice to the processor waiting for some fixed amount of (wallclock) time for new data, before proceeding with stored data.

Another expression of the scheme (allowing and not allowing communication delay), closer to the language of [101], is as follows. We can observe that the scheme presented is equivalent to

$$\mathbf{u}^{n+1} = \sum_{d=1}^{D} I_{d_s,d_e} e^{\Delta t L_{d_S+1,d_E}} \mathbf{u}^n_{d_S,d_E}. \tag{7.9}$$

The communication delay scheme is then

$$\mathbf{u}^{n+1} = \sum_{d=1}^{D} I_{d_s,d_e} e^{\Delta t L_{d_S+1,d_E}} (\hat{I}^n_d \mathbf{u}^n + \bar{I}^n_d \mathbf{u}^{n-1}), \tag{7.10}$$

where the matrix $\hat{I}_d^n$ is the diagonal matrix with entries $(\hat{I}_d^n)_{i,i} = 1$ for $i = d_s, \ldots, d_e$, $(\hat{I}_d^n)_{i,i} = r_d^n$ for $i = d_S, \ldots, d_s - 1, d_e + 1, \ldots, d_E$, and all other entries zero. That is,

$$\bar{I}_d^n = \begin{pmatrix} 0 &&&&&&&&&& \\ & r_d^n &&&&&&&&& \\ && \ddots &&&&&&&& \\ &&& r_d^n &&&&&&& \\ &&&& 1 &&&&&& \\ &&&&& \ddots &&&&& \\ &&&&&& 1 &&&& \\ &&&&&&& r_d^n &&& \\ &&&&&&&& \ddots && \\ &&&&&&&&& r_d^n & \\ &&&&&&&&&& 0 \end{pmatrix} \begin{array}{l} \} \text{ Rows } 0 \text{ to } d_S - 1 \\ \\ \} \text{ Rows } d_S \text{ to } d_s - 1 \\ \\ \\ \} \text{ Rows } d_s \text{ to } d_e \\ \\ \\ \} \text{ Rows } d_e + 1 \text{ to } d_E \\ \\ \} \text{ Rows } d_E + 1 \text{ to } N \end{array}$$

The matrix $\bar{I}_d^n$ is the diagonal matrix with $(\bar{I}_d^n)_{i,i} = 1 - r_d^n$ for $i = d_S, \ldots, d_s - 1, d_e + 1, \ldots, d_E$ (the same indexes for which $\hat{I}_d^n$ has entries r_d^n); and zero for all other entries. Thus $\hat{I}_d^n + \bar{I}_d^n = I_{d_S, d_E}$. That is,

$$\hat{I}_d^n = \begin{pmatrix} 0 &&&&&&&&&& \\ & 1 - r_d^n &&&&&&&&& \\ && \ddots &&&&&&&& \\ &&& 1 - r_d^n &&&&&&& \\ &&&& 0 &&&&&& \\ &&&&& \ddots &&&&& \\ &&&&&& 0 &&&& \\ &&&&&&& 1 - r_d^n &&& \\ &&&&&&&& \ddots && \\ &&&&&&&&& 1 - r_d^n & \\ &&&&&&&&&& 0 \end{pmatrix} \begin{array}{l} \} \text{ Rows } 0 \text{ to } d_S - 1 \\ \\ \} \text{ Rows } d_S \text{ to } d_s - 1 \\ \\ \\ \} \text{ Rows } d_s \text{ to } d_e \\ \\ \\ \} \text{ Rows } d_e + 1 \text{ to } d_E \\ \\ \} \text{ Rows } d_E + 1 \text{ to } N \end{array}$$

The random variable r_d^n represents communication delay and can take values either 0 or 1. If $r_d^n = 1$ then $\hat{I}_d^n = I_{d_S,d_E}$, $\bar{I}_d^n$ is the empty matrix in and the update (7.10), the partial solution on domain d is updated using entirely new data - there is no delay in communication. If $r_d^n = 0$ there is a communication delay and domain d updates using data from one timestep behind.

We have described here only the possibility of delays by one step. We introduce a formulation allowing longer communication delays later in §7.4.

In addition to (7.10) we also describe the scheme as Algorithm 4, which more clearly illustrates the parallel concepts such as domain decomposition and communication delay. Most of Algorithm 4 is split over the domains d as shown by line 1. Lines 2 and 3 are preparation of the 1D domain. In line 4 the restriction of the solution vector $\mathbf{u}$ to the nodes of the domain d plus the overlap nodes is prepared, as is the corresponding discretisation matrix for the domain L_{d_S,d_E}. In line 5 the main timestepping loop for the domain d begins. Line 6 checks that new data for the overlap regions is available, and the data for the overlap nodes in $\mathbf{u}_{d_S,d_E}$ is updated if it is (lines 7 and 8), this is (7.7). Note that for the very first timestep the if statement starting at line 6 can be ignored since d is starting from the initial conditions. Lines 10 and 11 are the update of the solution according to (7.6). We denote the time as t_d because it is local to the domain d. In line 12 the data from the internal nodes of d which are overlap nodes for other domains, is transmitted to the appropriate other nodes. Finally in line 15 the data from all domains is brought together and the final solution is made from the internal nodes of each domain according to (7.8).

Lines 6 and 12 in Algorithm 4 represent the communication of data between domains (and thus processing elements) and the possibility of communication delay (the if statement at line 6). Note that in our experiments we *simulate* this communication on a serial implementation. Essentially the freshest overlap data is always available so that line 12 is irrelevant, and lines 7 and 8 are either carried out or not depending on the evaluation of some random variable (replacing line 6) which we can control the properties of.

```
   Data: Grid structure, Initial concentration values, $T$, number of overlap
         nodes $o_l$.
1  Parfor Each Domain $d$ do
2      Get start and end nodes $d_s, d_e$ ;
3      Get start and end nodes with overlap, $d_S = d_s - o_l$ and $d_E = d_e + o_l$ ;
4      Prepare $\mathbf{u}_{d_S,d_E}$ and $L_{d_S+1,d_E}$ ;
5      while $t_d < T$ do
6          if New overlap data available then
7              Replace $\mathbf{u}_{d_S,d_E}(d_S : d_s - 1)$ with new data ;
8              Replace $\mathbf{u}_{d_S,d_E}(d_e + 1 : d_E)$ with new data ;
9          end
10         $\mathbf{u}_{d_S,d_E} = e^{\Delta t L_{d_S,d_E}} \mathbf{u}_{d_S,d_E}$ ;
11         $t_d = t_d + \Delta t$ ;
12         Transmit $\mathbf{u}_{d_S,d_E}(d_s : d_s + o_l)$ and $\mathbf{u}_{d_S,d_E}(d_e - o_l : d_e)$ to other domains
             as new overlap data ;
13     end
14 end
15 $\mathbf{u} = \sum_d I_{d_s,d_e} \mathbf{u}_{d_S,d_E}$
```

Algorithm 4: Pseudo code for the exponential DD scheme with communication delay. See text for full description.

7.2.1 A Stability Result

Here we follow a similar argument to [101, §3], which relates the stability of the communication-delay forward Euler scheme to the stability of the underlying forward Euler scheme. There it is shown that if the underlying Euler scheme satisfies a sufficient stability condition, then so does the communication delay scheme.

Similar to [101, §3], we consider the following re-expression of (7.10),

$$\begin{pmatrix} \mathbf{u}^{n+1} \\ \mathbf{u}^{n} \end{pmatrix} = \begin{pmatrix} \sum_{d=1}^{D} I_{d_s,d_e} e^{\Delta t L_{d_S+1,d_E}} \hat{I}_d^n & \sum_{d=1}^{D} I_{d_s,d_e} e^{\Delta t L_{d_S+1,d_E}} \bar{I}_d^n \\ 0 & I \end{pmatrix} \begin{pmatrix} \mathbf{u}^{n} \\ \mathbf{u}^{n-1} \end{pmatrix}. \tag{7.11}$$

Let the matrix be

$$M_n \equiv \begin{pmatrix} \sum_{d=1}^{D} I_{d_s,d_e} e^{\Delta t L_{d_S+1,d_E}} \hat{I}_d^n & \sum_{d=1}^{D} I_{d_s,d_e} e^{\Delta t L_{d_S+1,d_E}} \bar{I}_d^n \\ 0 & I \end{pmatrix}. \tag{7.12}$$

The final approximation produced by the scheme will be

$$\begin{pmatrix} \mathbf{u}^N \\ \mathbf{u}^{N-1} \end{pmatrix} = M_{N-1}M_{N-2}\dots M_2 \begin{pmatrix} \mathbf{u}^2 \\ \mathbf{u}^1 \end{pmatrix}, \tag{7.13}$$

and a sufficient condition for stability is that the infinity norm of the accumulation $M_{N-1}M_{N-2}\dots M_2$ is less than or equal to one; i.e.

$$||M_{N-1}M_{N-2}\dots M_2||_\infty \leq 1,$$

a sufficient condition for which is

$$||M_n||_\infty \leq 1 \;\; n \in [2, N-1].$$

Ideally we would like to have that this holds if the underlying ODE system produced by the spatial discretisation is stable. It is straightforward to show stability in the case of the following, slightly stronger assumption.

Assumption 7.2.1. *The underlying ODE system in every domain d satisfies the sufficient stability condition in the sense that,*

$$||e^{\Delta t L_{d_S+1,d_E}}||_\infty \leq 1, \;\; \forall \Delta t > 0.$$

This requires all the eigenvalues of L_{d_S+1,d_E} to be negative.

Then,

Lemma 7.2.2. *Given Assumption 7.2.1, then*

$$||M_n||_\infty \leq 1,$$

and so the exponential communication delay scheme is stable.

Proof. We will show that the absolute sum of every row of M is either 1, or bounded by $||e^{\Delta t L_{d_S+1,d_E}}||_\infty$ for some d. Therefore every row sum of M_n is less than or equal to one by Assumption 7.2.1.

For every row in the 'bottom half' of M_n - the $(0, I)$ part - the absolute row sum is 1. Consider the other rows, which depend on exponential terms.

The non-overlapping structure of the identity like matrices I_{d_s,d_e} ensures that every row of the upper part of M_n is contributed to by only one domain d, despite the sum in (7.12). That is, every row in the upper part of M_n is a row from

$$(I_{d_s,d_e} e^{\Delta t L_{d_S+1,d_E}} \hat{I}_d^n, I_{d_s,d_e} e^{\Delta t L_{d_S+1,d_E}} \bar{I}_d^n) \equiv M_{n,d},$$

for some d. Now also the other two identity like matrices $\bar{I}_d^n$ and $\hat{I}_d^n$ do not overlap and have entries either zero or unity, so that the columns of $M_{n,d}$ are either empty or columns of $e^{\Delta t L_{d_S+1,d_E}}$, and each column of $e^{\Delta t L_{d_S+1,d_E}}$ appears only once in $M_{n,d}$. Thus considering the absolute row sums of $M_{n,d}$ we have that $||M_{n,d}||_\infty = ||e^{\Delta t L_{d_S+1,d_E}}||_\infty$. So every row in the upper part of M_n is bounded by a $||e^{\Delta t L_{d_S+1,d_E}}||_\infty$ for some d. □

7.3 Numerical Results

Here we present experiments using the schemes described above. Our results are in Figure 7.2 through Figure 7.3. We describe the figures here. It is necessary to denote schemes and their parameters (subdomains used, number of cells in overlap) by a key. We write exp for the exponential domain decomposition method with no communication delay, expN for the same exponential method with communication delay (take N to stand for 'noise' or 'noisy communication'), and euN for the Euler stepping method with communication delay. When a scheme uses x subdomains and has an overlap of y cells, we write 'xDOy'. So for example, exp 2DO1 is the synchronous exponential solver with two subdomains and an overlap of one cell, while euN 5DO3 is the Euler stepping method with five subdomains and an overlap of three cells. In all the experiments in this chapter the test PDE is the diffusion equation (7.1). In Figure 7.2 through Figure 7.3, the ordering of plots is as follows. Plot a) shows the initial condition alongside the true solution (we calculate this as simply by taking the matrix exponential of the discretisation matrix, multiplied by the final time T, and multiplying this by the initial condition vector). The approximate solution produced by one of the expN schemes is also shown in plot a) for comparison. In plot b) we present error against timestep Δt for the synchronous exponential solvers. It is useful to see how these schemes behave before the communication delay is added, and this will be discussed below.

For our experiments we used a single processor and simulated communication delay by having the schemes randomly not update overlap regions according to a certain probability. We chose to look at a twenty percent chance and a fifty percent chance of communication failure. Note that communication can only be delayed by one step in these examples. As the experiments are stochastic, each run was repeated 10 times and the mean error calculated; this is what is reported.

7.3.1 System One - Simple Diffusion

In experiment one, Figure 7.2, we have a 100 node system with diffusivity uniformly set to unity. The final time is $T = 100.0s$, and the domain has size $100.0m$. There are no-flow conditions on the boundaries. Plot a) shows the initial condition, the exact solution, and the approximation produced by one of the schemes.

Plot b) shows the results with no communication delay. It is interesting to note that the number of overlap nodes seems to be of great importance for these schemes, as increasing only from one to two to three causes a huge improvement in accuracy and convergence rate. Increasing the number of domains worsens the performance of the schemes, as would be expected. We use solvers with two and five domains here to illustrate the difference. Of course, if the number of domains were one, the solver would be exact. While the number of domains makes a difference, it is the overlap that is of greater importance, as it appears to determine the rate of convergence.

All this remains true when we introduce communication delay. Plots c) and d) show the results with twenty percent and fifty percent chance of communication delay respectively. All the previous comments about overlap and number of domains still hold true, but it can be observed that communication delays make the schemes less accurate, with increases in error of over an order of magnitude for fifty percent communication delay chance. The schemes still exhibit convergence and stability.

In plots e) and f) we show results with simple Euler-type DD-communication delay schemes. The communication delay probabilities are the same. These schemes are unstable and we do not plot our results for $\Delta t > 0.667$ (150 timesteps) as the error increases by several orders of magnitude at this point and renders the plots unreadable. It is interesting that, while the number of domains is clearly important, the number of overlap nodes makes seems to make minimal difference for these Euler-type schemes.

Finally in plots g) and h), we compare the exponential and Euler-type schemes for accuracy.

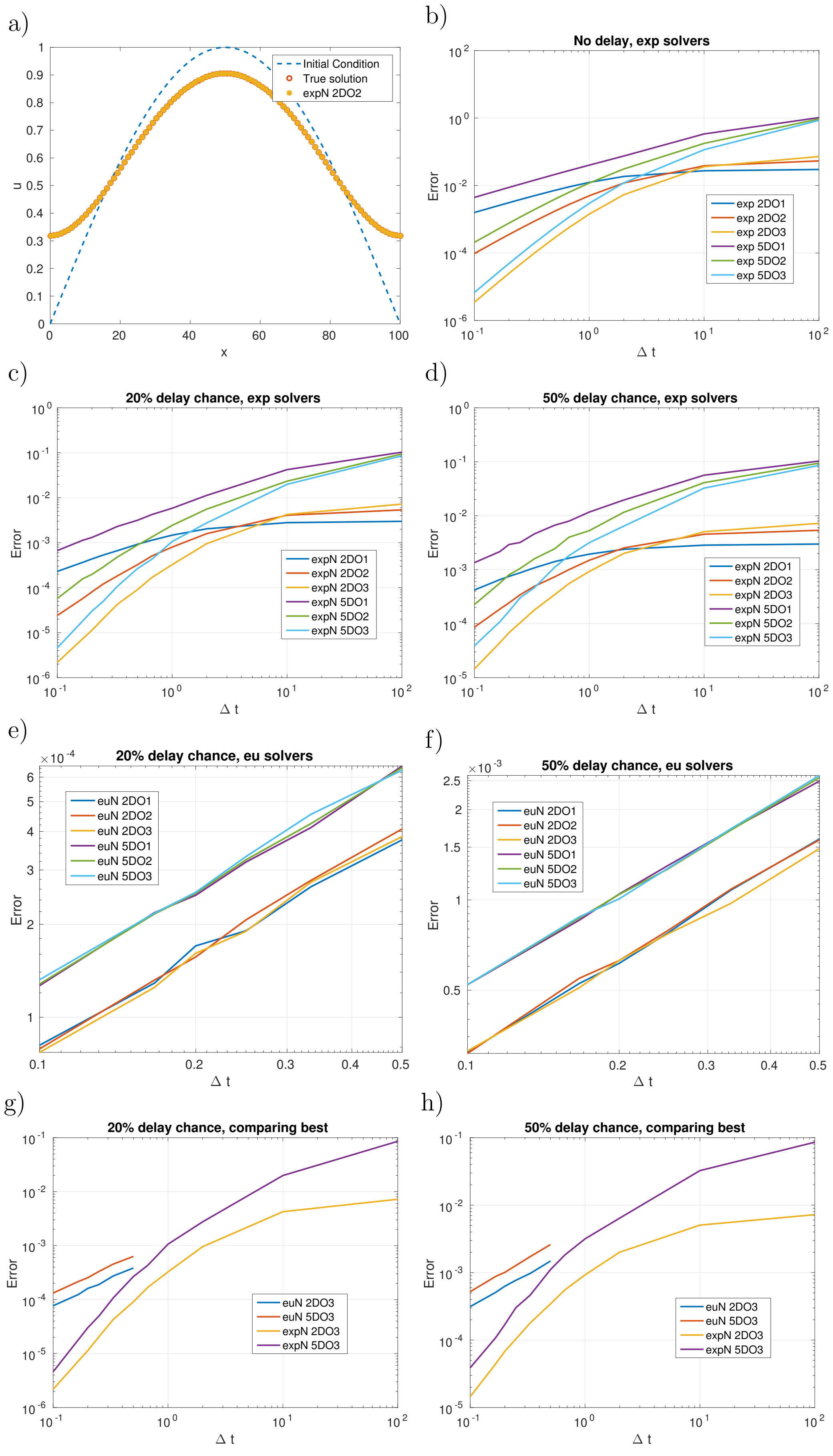


Figure 7.2: Experiment one. See §7.3.1 for details.

7.3.2 System Two - Random Diffusivity Field

The next experiment, Figure 7.3, takes place on the same domain as experiment one, but now the diffusivity field is random. For simplicity the field was simply generated by picking uniform random numbers between zero and unity. Thus there is no, for example, spatial correlation in the diffusivity, which is unphysical but is an interesting test for the schemes. A different initial condition can also be seen in plot a). The boundary conditions are again no-flow, and the final time is again $T = 100.0s$.

The results for this experiment lead to the same conclusions as the previous one. We see the same observations about the effect of overlap on the exponential type schemes; how the communication delay worsens accuracy but does not prevent convergence, and the stability of the exponential type schemes.

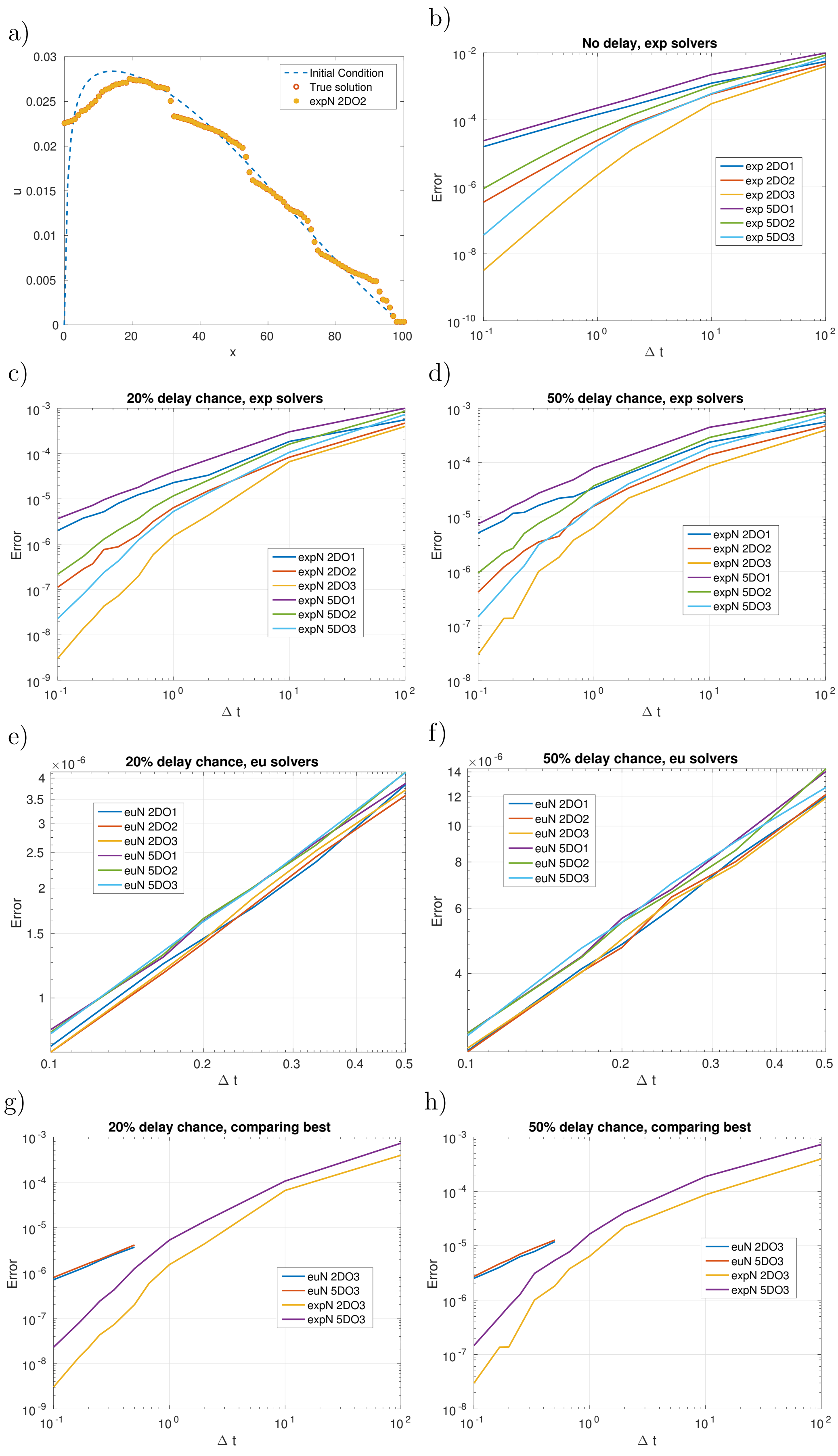


Figure 7.3: Experiment two. See §7.3.2 for details.

7.4 Allowing More Previous Steps

In a more realistic situation we would expect communication delays of more than one step to occur. In this section we investigate simulations of communication-delay schemes where this is the case. Now, at the point where a domain would receive information from its neighbouring domain from the previous step, it has a chance of receiving the information from that domain at an older step.

For example, consider a domain d which has an overlap with a neighbour domain d'. Domain d has just calculated its values step n and needs information from domain d' in the overlap region, at step n, in order to calculate its values for step $n+1$. In the previous schemes this information either is or is not communicated in time. If it is not, then domain d will proceed with the progression from step n to step $n+1$ using its own stored values for the overlap region. In the schemes we propose here, it is possible for domain d to receive, with the probability specified below, the overlap region information from d' at step n, or $n-1$, or $n-2, \ldots n-\text{psteps}$, where the parameter psteps is defined in advance by the user.

For our experiments here, we have chosen to have the probability of d receiving older steps described by an exponential probability distribution, with step n being most likely and step psteps being least likely. Moreover, when the random value indicates that a step older than $n-\text{psteps}$ be used, then domain d instead updates using its own stored values instead, behaving like the schemes in the previous section. Depending on the particular exponential distribution used, there can result quite a significant number of such events. It seems reasonable to impose some limit on how 'out of date' information can be before it can be used, which is why we have implemented the psteps parameter. It also seems reasonable to have the domain proceed with its own stored data, as in the previous sections, when the only information available is judged to be too old, since this strategy seems to be successful in the previous sections. One caveat of the current method is that it is possible for newer data to be overwritten with data from an older step. That is, consider some domain d updating on step n with data from step $n_1 < n$ from an adjacent domain. Then it is possible for a later step on d to update with data from step $n_2 < n_1$, which is likely undesirable. We presume that a real parallel implementation would have measures to prevent this.

The exponential distribution requires a mean to be defined, for our experiments it is defined as a factor of psteps. That is, we introduce another parameter called normfac and the mean of the exponential distribution is then defined to be

$$\text{psteps} \times \text{normfac}.$$

For example, we might have defined psteps = 20 and normfac = 0.25, in which case the exponential distribution used has a norm of 5. We used the built in Matlab function exprnd for the random number generation in our experiments.

We have repeated our experiments from §7.3 with the new schemes described here, and the results follow in Figure 7.4 and Figure 7.6. Again, we have not implemented a truly asynchronous processing application, but simulated one in Matlab, and are now simulating a communication delay following an exponential distribution.

We note here that it would be straightforward to extend the stability result of §7.2.1 to this scheme. Expressing this scheme in the same manner as (7.10) is a simple matter of defining psteps many non-overlapping identity-like matrices like $\bar{I}_d^n$ and $\hat{I}_d^n$ instead of two, to represent the effect of information from previous steps being used. The matrix M_n in §7.3 would then have psteps horizontal block elements instead of two. The argument for stability would then be essentially the same.

7.4.1 Numerical Results - System One

This is the same system as in §7.3.1. The results for the new schemes with a variety of domains, overlaps, and values of psteps and normfac is given in Figure 7.4. We also show in Figure 7.5 some examples of how the communication delay for the new schemes follows an exponential distribution, up to the imposition of a maximum delay of psteps. See the figure caption for full details.

From Figure 7.4 we see that conclusions similar to those for the previous schemes can be drawn. The schemes will converge at first order or slightly worse for sufficiently low Δt; and increasing the number of domains decreases accuracy while increasing the overlap width between domains increases it. However it does seem that in many of the plots there is little difference between using two domains with an overlap of two and two domains with an overlap of three.

Regarding the factor psteps, we see from comparing the pairs of plots a) and b); c) and d); and e) and f), that increasing psteps for fixed normfac seems to cause a decrease in accuracy. Increasing psteps for fixed normfac means allowing 'older' information to be used for domain updates, and fewer events in which domains disregard information from their neighbours in favour of updating using their own stored information in the overlap zones from the previous step. If increasing psteps makes the accuracy of the schemes worse, this might indicate that the method of the schemes of the previous sections is advantageous, at least for the diffusion-only systems looked at here.

By comparing the triplets of plots a), c), e) and b), d), f), we can examine the effect of increasing normfac with a fixed psteps. We see a decrease in accuracy as we increase normfac in this way. Since increasing normfac increases the norm of the exponential distribution, this will increase the average 'age' of previous steps used, which likely explains the decrease in accuracy.

We can observe a certain amount of clustering of the lines for the scheme with two domains in some of the plots, when psteps is greater (plots b) d) and f)), however in all the plots the lines for the scheme with two domains and overlap two ($2DO2$) and for the scheme with two domains and overlap three ($2DO3$) seem to converge together as $\Delta t \to 0$. Perhaps by allowing information from older steps to be used a limiting factor has been introduced, preventing the increase in accuracy we might have expected from increasing the overlap width.

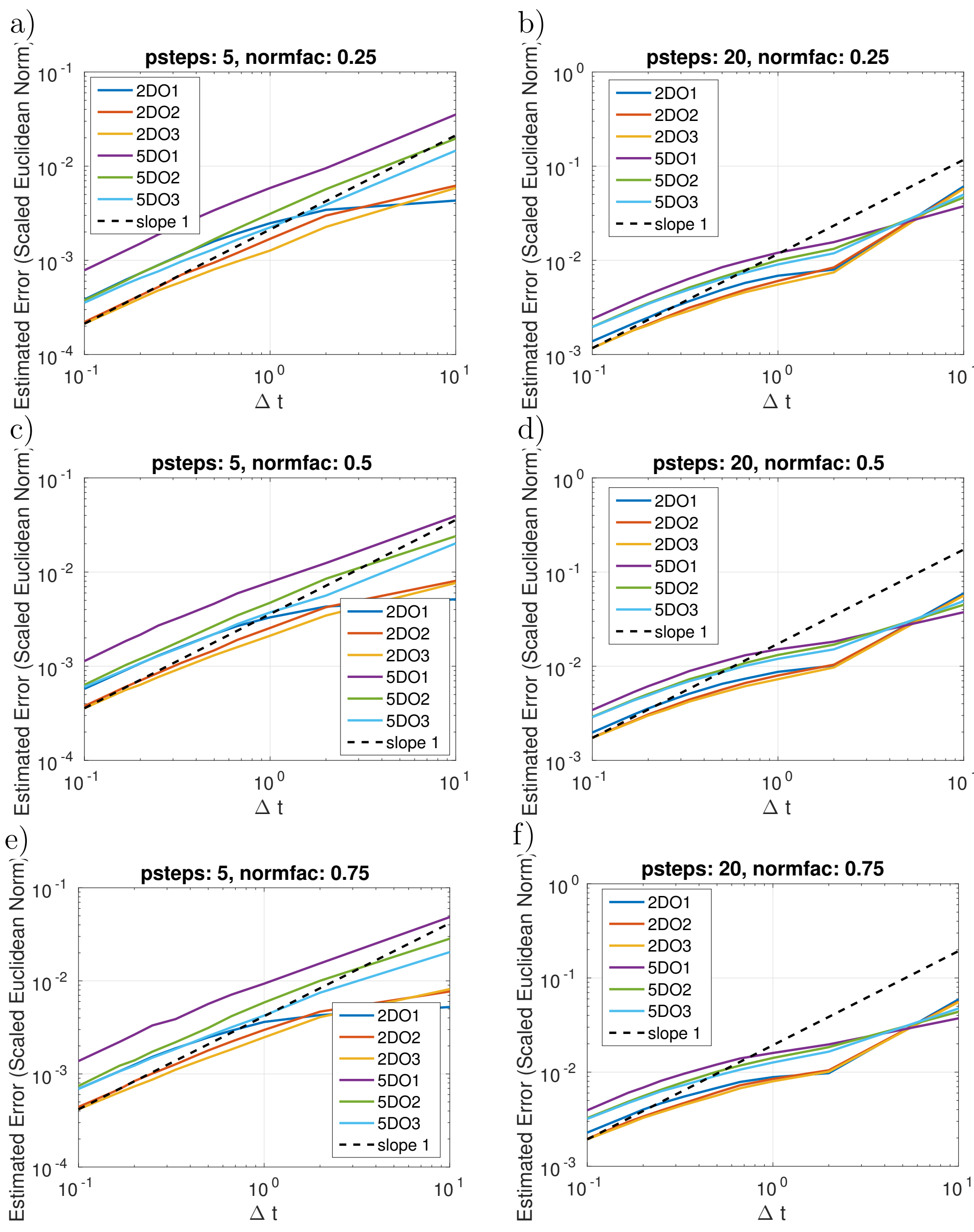


Figure 7.4: Results for the first system, showing error against timestep, for various values of psteps and normfac.

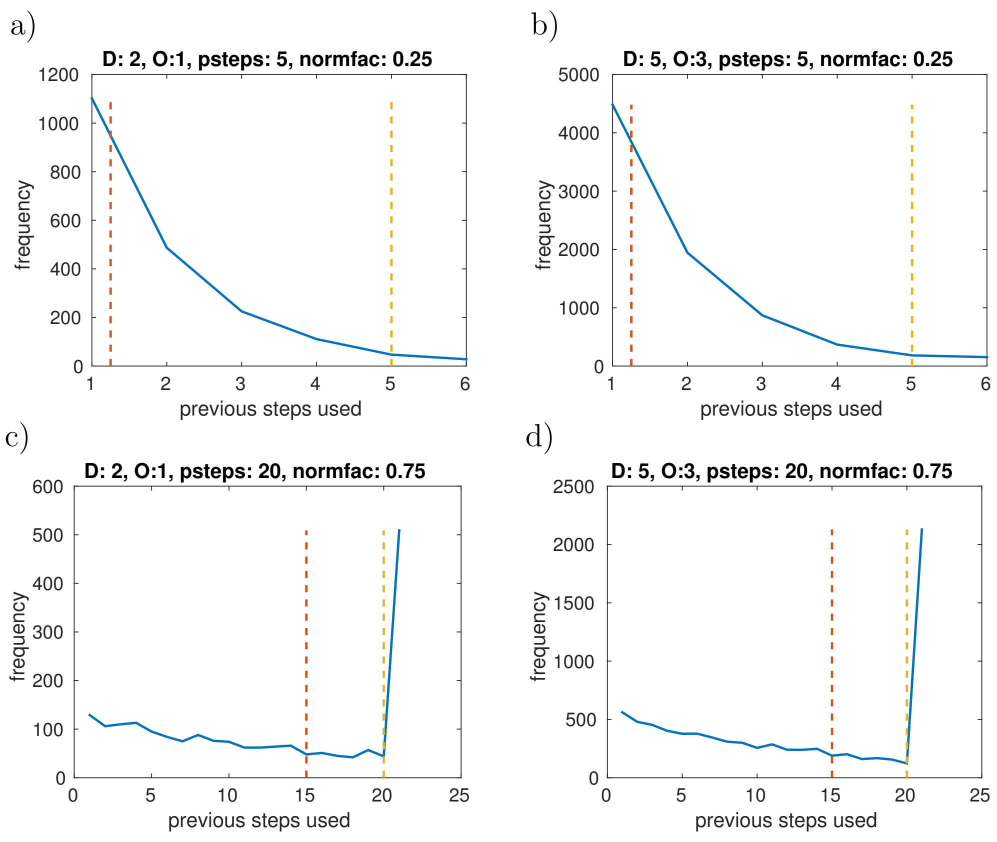


Figure 7.5: Demonstrating how communication delay is made to follow an exponential distribution, for system 1, for a few different domains, overlaps, and values of psteps and normfac. To read these plots, consider a domain at timestep $n-1$ requiring information from a neighbouring domain also at time $n-1$ in order to progress. According to the random number generator, information at a step $n-i$ will be transmitted. The vertical axis of the plots is that number i. The horizontal axis of the plots is the frequency, across every timestep across every domain, for one solve with 1000 timesteps. The information is only considered acceptable if $i \leq$ psteps. On each plot, psteps is shown as a yellow line. The single point plotted to the right of the yellow line is the sum of *every* time the exponential distribution gave a value $i >$ psteps. In this case the information from step $n-i$ was not used and the domain proceeded with its own stored information instead. It can be seen that certain combinations of parameters (plots c) and d)) can cause this to happen very often. The mean of the exponential distribution used is given by psteps $\times$ normfac and is plotted as a red line in the plots.

7.4.2 Numerical Results - System Two

This is the same system as in §7.3.2. The results for the new schemes with a variety of domains, overlaps, and values of psteps and normfac is given in Figure 7.6. Like in the previous sections, we also show in Figure 7.7 some examples of how the communication delay for the new schemes follows an exponential distribution, up to the imposition of a maximum delay of psteps. See the figure caption in Figure 7.5 for a full description.

The conclusions we can draw from Figure 7.6 are largely the same as those we can draw from Figure 7.4, and elaborated in §7.4. The same connections between number of domains, overlap, and accuracy can be observed, as can the roughly first order convergence of all the schemes. By comparing the pairs of plots a) and b); c) and d); and e) and f), we can observe the effect of increasing psteps for fixed normfac, and like in the previous example, we see that this decreases accuracy. By comparing the triplets of plots a), c), e) and b), d), f), we can examine the effect of increasing normfac with a fixed psteps. Again like the previous experiments, this seems to decrease the accuracy of the schemes.

We can observe similar clustering of the lines for the schemes in the plots, as described for the previous experiment. The clustering seems to be affecting also the schemes with five domains in this experiment, especially at larger psteps and normfac values (see plot f) for example).

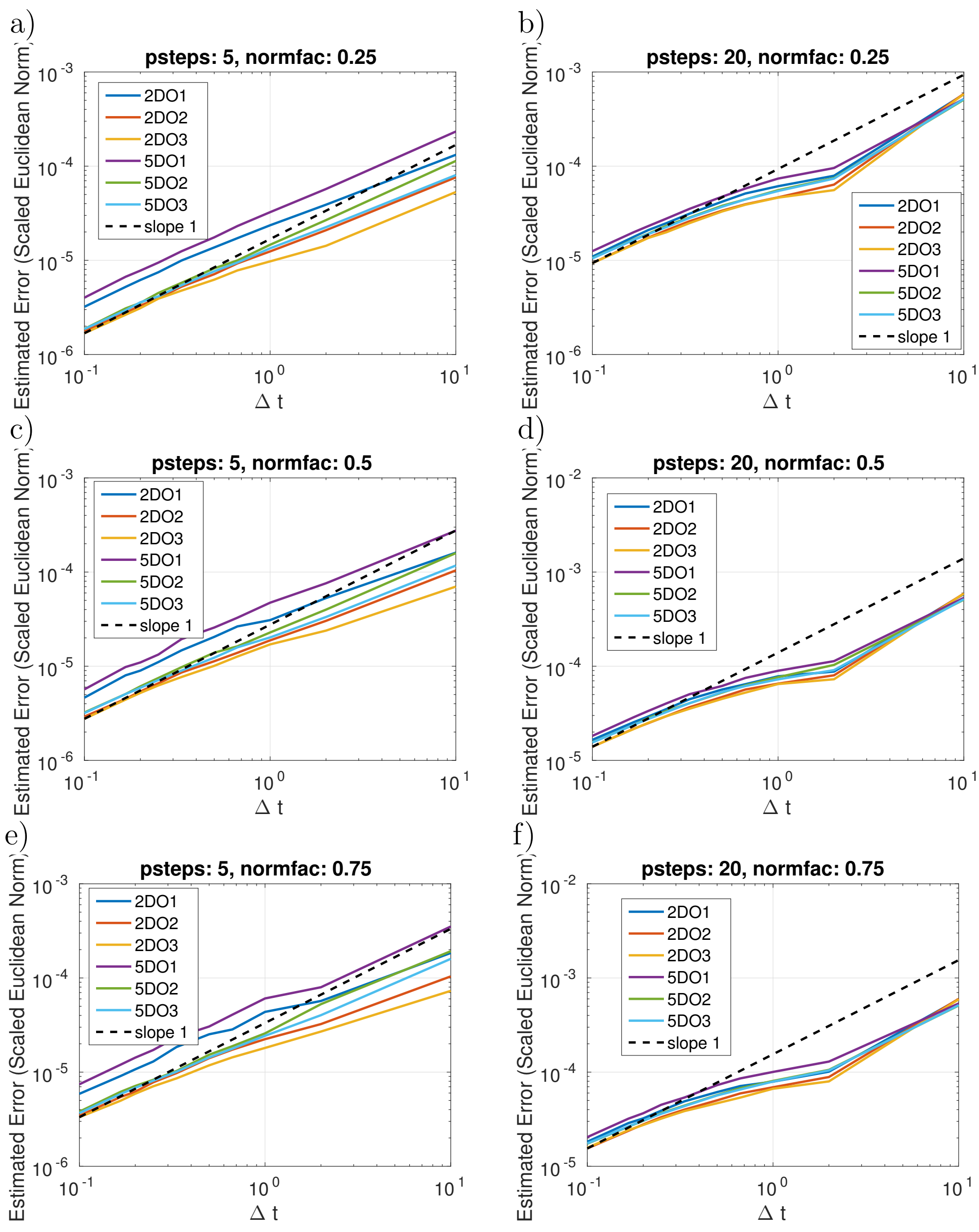


Figure 7.6: Results for the second system, showing error against timestep, for various values of psteps and normfac.

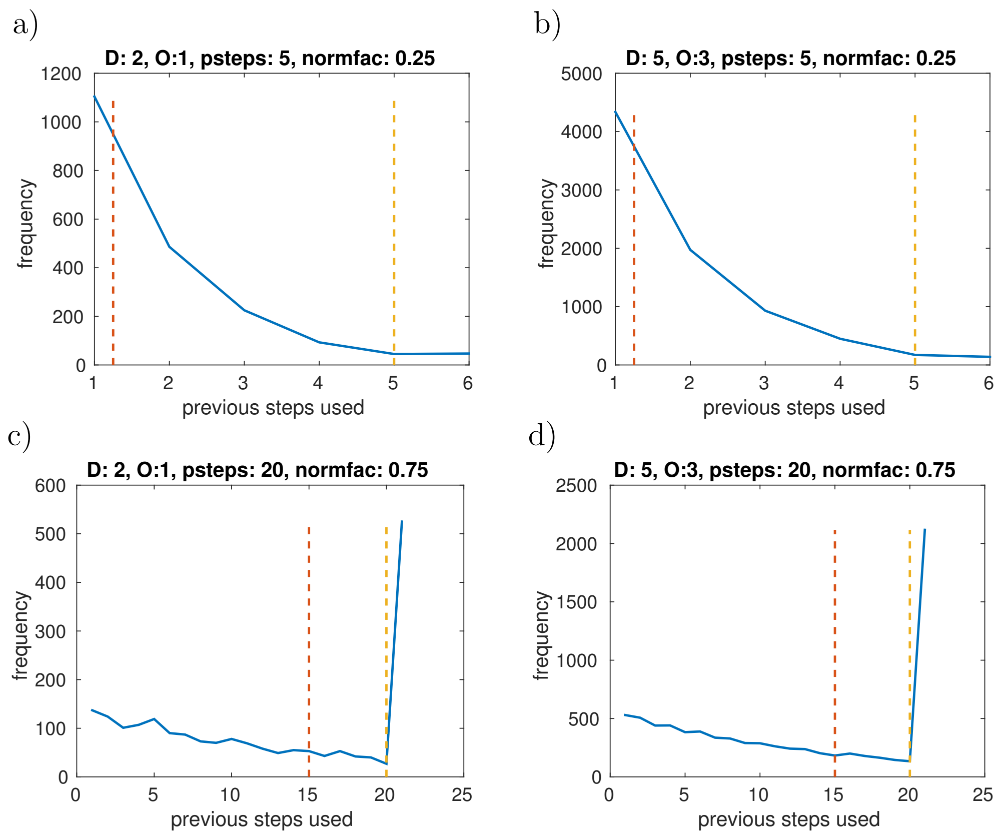


Figure 7.7: Demonstrating how communication delay is made to follow an exponential distribution, for system 1, for a few different domains, overlaps, and values of psteps and normfac. See the caption for Figure 7.5 for a full explanation.

7.4.3 Conclusions

The concept of domain decomposition schemes that allow communication delay, introduced in [101], has been extended by the introduction of a scheme which advances by means of a local matrix exponential on each domain, and which allows random communication delay of data in overlap regions between domains. This scheme has been tested for one dimensional diffusion problems, and has been demonstrated to converge to the correct solution as the timestep value decreases to zero. The accuracy of the scheme is worsened by increasing both the probability of communication delay and the number of domains, as would be expected, but convergence is not prevented by communication delay.

The exponential scheme seems to be especially sensitive to the size of the overlap region between domains, with an increase of even one or two nodes causing a significant increase in convergence rate. The effect of a moderate size overlap region is not nullified by a high probability of communication delay. Such schemes may be expected to provide high accuracy in general if sufficiently large overlap regions are allowed. It is worth noting that larger overlap regions might increase the probability of communication delay as more data would need to be communicated between nodes. There may be optimal values of o_l which provide maximal efficiency by balancing these two effects.

By introducing the possibility for the schemes to use older steps according to an exponential distribution we have attempted to make the simulation of communication delay more realistic. The schemes still converge, which is promising. The same relations between domain number and overlap width can be observed; in general, fewer domains and wider overlaps are better. We also introduced the psteps parameter to allow us to define a limit on how long a communication delay can be tolerated before information is discarded and a domain proceeds with its own stored data, as in the schemes described at the start of this chapter. This seems like a reasonable precaution. We can observe from our results that increasing psteps with fixed normfac tends to decrease the accuracy of our new schemes. The effect of psteps with fixed normfac is to allow older steps to be used and decrease the number of instances when a domain will proceed using its own stored data. The observation that this decreases the accuracy of the schemes may indicate that allowing domains to proceed with their own data is a beneficial strategy, at least for the simple diffusion problems we have used for our examples.

Overall we have added a modest contribution to the promising results of [101], by providing experimental results of how another type of scheme, linear exponential integrators, can be resistant to communication delay in a domain decomposition.

Chapter 8

Conclusion

In Chapter 2 we advanced the Discrete Event methodology of [1] by developing schemes for this methodology in a new paradigm (face-based as opposed to cell-based events). The scheme BAS was presented, which is the simplest scheme of this class we can design. BAS is based on the underlying physics of the system being simulated in that it is based on inter-cell flux.

Modifications to BAS were then introduced. The mass tracking concept attempts to reduce the error introduced by decoupling events on faces of the same cell. The exact mass transfer method takes advantage of the simplicity of the reduced system that must be approximated at each cell face by exactly solving the linear ODE there. We adapted the 'flux capacitor' concept introduced in [1] to produce 'cascading' versions of the schemes, where additional events can be triggered on high activity faces, bypassing the priority queue that schedules pending events. Finally a leapfrog-like method for including reaction terms in each of the new schemes was introduced.

In Chapter 3 numeral tests were performed on the new schemes. The most important result from these is the convergence of every scheme, with apparent first order, in ΔM, where the mass unit ΔM is a global control on the amount of mass that can be transferred in each event. Decreasing ΔM increases accuracy at the cost of increasing cputime, so it appears that ΔM in this way acts like the timestep length in fixed timestep schemes or the error tolerance in adaptive timestepping schemes.

From the numerical tests in Chapter 3 a number of relations between scheme parameters were strongly implied, these were, Error $= O(\Delta t \text{ (average)})$, $N = O(\Delta M^{-1})$

(where N is the total amount of events in the course of the solve), Δt (average) $=$ $O(\Delta M)$, and Δt (average) $= O(N^{-1})$. These relations seem to become valid as ΔM becomes sufficiently low.

In Chapter 4 we presented results towards analysis of the new schemes. The convergence of BAS with ΔM in a very simplified scenario was proved, and the steps for a potential proof for BAS in general were laid out. Similarly, analysis of the convergence of EAS was performed, which takes advantage of the structure of EAS to express it as a product of matrix exponentials, which can be examined using the BCH formula. Possible further work was discussed in §4.6.

In Chapter 5 we developed a first order ETD-like scheme which recycles the Krylov subspace up to S times to produce S substeps each timestep. The $S = 1$ case corresponds to ETD1. The cheap extra substeps provide additional accuracy, and it was proved that increasing S decreases the local error of the scheme to a limiting value. It was thus conjectured that there exists an optimal S, dependent on the underlying problem and other scheme parameters, to provide optimal efficiency. Numerical testing bore this out, and showed that using an S greater than 1 can indeed increase efficiency. A second order corrector scheme was also developed, which uses the data at the substeps to approximate the leading term of the error in the scheme, to calculate a corrector and increase accuracy.

In Chapter 6 we schematically developed a class of 'semi-exponential' Runge Kutta like methods. These methods satisfy the requirements of being an ETD-type scheme up to agreeing with ETD1, and beyond that satisfy classical order conditions. They can be implemented with a single matrix exponential approximation per timestep, but this comes as a compromise for sacrificing some of the stability and stiff order properties of true exponential integrators.

Numerical tests show these new schemes are generally effective up to order three, but the fourth order scheme is generally reduced to third order, likely a result of sacrificing the stiff order properties of true exponential integrators. The schemes also do not fare well for large Δt values, but can be competitively efficient when Δt is not too great.

In Chapter 7 the concept of domain decomposition schemes that allow communication delay, introduced in [101], was been extended by the introduction of a scheme which advances by means of a local matrix exponential on each domain, essentially a linear exponential integrator. It was shown that such a scheme has favourable stability properties in spite of the communication delay. Experiments were performed with the new scheme, showing convergence to a correct solution and general reliability for various numbers of domains, communication delay probability, and number of previous steps allowed. This chapter added contribution to the promising results of [101], by providing experimental results of how another type of scheme, linear exponential integrators, can be resistant to communication delay in a domain decomposition.

We finish with some final concluding remarks. For practical use, in the reliable efficient simulation of porous media systems, the scheme of choice from this work would be the Krylov recycling schemes of Chapter 5. The methodology introduced there builds on an already well established exponential integrator scheme and provides a way to improve efficiency by adding recycled substeps. Currently the method requires Δt to be sufficiently small or the Krylov subspace dimension m to be sufficiently great that the error of the initial Krylov approximation is not dominant over the scheme error. Technology exists in the literature to control the Krylov error using adaptive (non-recycling) substeps and adaptive varying of m. An implementation incorporating both this and recycling substeps could provide a scheme that is advantageous in general.

The two classes of Asynchronous scheme discussed in this thesis, the DES based methodology of Chapters 2, 3, and 4, and the communication delay scheme of Chapter 7, are essentially proofs of concept. The work in these chapters introduces new schemes that use the concept of asynchronicity in different ways.

In Chapters 2, 3, and 4 a new class of schemes within the discrete event simulation paradigm have been developed, and numerical evidence strongly indicates that they are ‘correct’ in the sense of converging to a reliable solution (agreeing with a classical solve) when the parameter ΔM is decreased. The current implementations of these schemes are not competitively efficient, see §3.3 for more detailed discussion of this, but their general level of success indicates that it should be possible to design a wide range of viable asynchronous schemes of this type, some of which may prove well suited to fast, efficient simulation of certain problems.

The communication delay scheme of Chapter 7 contributes to the ongoing study of schemes that are robust to random communication delays, as already discussed.

Overall, multiple new types of scheme have been developed, some belonging to the class of exponential integrators (Chapters 5 and 6), and some that can be described as asynchronous (the discrete event based schemes of Chapter 2, 3, 4; and the communication delay scheme of Chapter 7). Some schemes can be described as belonging to both classes: the EAS scheme introduced in Chapter 2 can be thought of applying an exponential integrator to the reduced system on a face during each event; and the communication delay scheme in Chapter 7 is essentially a domain decomposed exponential integrator applied to a simple linear problem.

Chapter 9

Appendix

9.1 Priority Queue for Asynchronous Schemes

For reference, we describe here the priority queue used for the schemes in Chapters 2, 3, and 4. It is a straightforward modification of a textbook implementation; see for example [103], or [104, Chapter 4] or [105, §1.9.4].

9.1.1 Binary Heap

The standard way to implement a priority queue is a binary heap. The heap itself can be represented by a single C++ vector. Consider a vector of face update times called *heap*. Let the entries of *heap* be indexed by i, so that we will call the update time stored in entry i of *heap*, $\hat{t}_i$. In order to function as a priority queue we must be able to access the minimum $\hat{t}_i$ from *heap* at any time. A simple but inefficient way to do this would be to keep *heap* totally ordered, but of course this would require an expensive re-sort every time there is a change. Instead *heap* has the following properties:

- Each entry of *heap* has two 'children'. Entry i of *heap* has children $2i$ and $2i+1$, if the indexing is such that i starts at one. It follows that the 'parent' of an entry can be found by calculating $floor(i/2)$. For example, entry 1 has children 2 and 3. Entry 3 has children 6 and 7. The parent of entry 11 is entry 5.

- The update time value $\hat{t}_i$ stored in entry i is less than that of its children, that is, $\hat{t}_i < \hat{t}_{2i}$ $\hat{t}_i < \hat{t}_{2i+1}$. This is called the *heap property.*

The heap property imposes a partial ordering that is more efficient to maintain as the priority queue is operated on. The structure will function as a priority queue since we are always able to access the minimum update time, since it will always be in entry 1. When a new entry is added to the queue, it is added at the end, then the queue is re-sorted by swapping the new entry with its parent until the heap property is restored. This can be done in $O(\log n)$ time, as does a corresponding procedure for sorting an element down through the heap. Construction of the queue is performed simply by sequentially adding the times at the end of the queue and sorting them into the heap until all are present.

This is a textbook implementation of a priority queue as a binary heap. We need to be able to access a particular face k, at any point, to adjust its update time and thus position in the heap. This can be done by adding additional vectors to keep track of a face's position in *heap*.

9.1.2 Accessible Binary Heap

Each entry $\hat{t}_i$ in *heap* must correspond to a unique face index k, for $\hat{t}_i$ is the update time for one of the faces. In addition to *heap*, we therefore keep an additional vector, $faces$, which contains the faces $1, \ldots, K$ arranged in order of priority. This vector can be maintained as a binary heap by simply having it mimic the vector *heap* - whenever a sorting procedure would swap two entries in *heap*, they are also swapped in $faces$. With this addition the priority queue can output both the lowest update time (the first entry in *heap*), and the index of the face with this time (the first entry in $faces$).

This still does not allow us to access a face by its index k and change its position in the priority queue by adjusting its update time. To this end we introduce a third vector, called *positions*. Now, entry k of *positions* always corresponds to face k. That is, entry 1 of *positions* corresponds to face 1, entry 2 to face 2, and so on. The value of entry k in *positions*, is the entry number of face k in the vectors *heap* and *faces*. For example, say the third entry of *heap* contains the update time for the face with index 10. Then *positions*(10) = 3. Thus we have a straightforward way of accessing the update time for a face in the priority queue, given the face's unique index k. Like with *faces*, we keep *positions* up to date by updating it every time there is a swap of elements in *heap* during a re-sort procure.

In practice we use the *positions* vector to adjust the priority queue every time we re-calculate the update time on the face. The appropriate entry in *heap* is found and adjusted, then a sort-up or sort-down procedure is called as necessary to restore the heap property.

9.2 Some Vector Calculus Results

We prove a needed vector calculus result for Chapter 5 here. Consider a vector function g taking vector argument x,

$$g = g(x),\ g \in \mathbb{R}^M,\ x \in \mathbb{R}^N.$$

Let $\hat{J}_i$ be the Hessian matrix

$$\begin{pmatrix} (\hat{g}_i)_{x_1x_1} & (\hat{g}_i)_{x_1x_2} & \cdots \\ (\hat{g}_i)_{x_2x_1} & (\hat{g}_i)_{x_2x_2} & \cdots \\ \cdots & \cdots & \cdots \end{pmatrix},$$

where $\hat{g}_i$ is the ith entry of the vector g. Then let the tensor $\hat{\mathbf{J}}$ be a vector with the matrix $\hat{J}_i$ in its ith entry. The Taylor series of a vector function $g(x)$ satisfies

$$g(x+\Delta x) = g(x) + J\Delta x + \frac{1}{2}\Delta x^T \hat{\mathbf{J}} \Delta x + O(\Delta x^3).$$

We will show that the second order term is indeed

$$\frac{1}{2}\Delta x^T \hat{\mathbf{J}} \Delta x.$$

We will do this by showing that the second order term of the Taylor series of $g_i(x+\Delta x)$ is

$$\Delta x^T \hat{J}_i \Delta x.$$

so that the claim above follows.

Define a simple 'filtering' of $\Delta x \equiv (\Delta x_1, \Delta x_2, \ldots, \Delta x_N)^T$ as follows:

$$\mathcal{F}(n)\Delta x = (0, 0, \ldots, 0, \Delta x_{n+1} \Delta x_{n+2}, \ldots, \Delta x_N)^T,$$

i.e., Δx with the first n elements replaced by 0.

Expanding $g_i(x+\Delta x)$ in the Δx_1 argument,

$$\begin{aligned} & g_i(x+\Delta x) \\ & = g_i(x+\mathcal{F}(1)\Delta x) + \Delta x_1 (g_i)_{x_1}(x+\mathcal{F}(1)\Delta x) \\ & + \frac{\Delta x_1^2}{2}(g_i)_{x_1 x_1}(x+\mathcal{F}(1)\Delta x) + O(\Delta x_1^3). \end{aligned} \tag{9.1}$$

After expanding the term $g_i(x+\mathcal{F}(1)\Delta x)$ in every argument we get

$$\begin{aligned} & g_i(x+\Delta x) \\ & = g_i(x) + \sum_{j=1}^{N} \Delta x_j (g_i)_{x_j}(x+\mathcal{F}(j)\Delta x) \\ & + \sum_{j=1}^{N} \frac{\Delta x_j^2}{2}(g_i)_{x_j x_j}(x+\mathcal{F}(j)\Delta x) + O(\Delta x_1^3, \Delta x_2^3, \ldots, \Delta x_N^3). \end{aligned} \tag{9.2}$$

For brevity we will from now on write the higher term denoting higher orders $O(\Delta x_1^3, \Delta x_2^3, \ldots, \Delta x_N^3)$ as simply $O(\Delta x^3)$. To leading order, we have that

$$(g_i)_{x_j x_j}(x+\mathcal{F}(j)\Delta x) = (g_i)_{x_j x_j}(x) + O(\Delta x_1, \Delta x_2, \ldots, \Delta x_N),$$

so we can simplify the last term in (9.2),

$$\begin{aligned}
&g_i(x+\Delta x) \\
&= g_i(x) + \sum_{j=1}^{N} \Delta x_j (g_i)_{x_j}(x+\mathcal{F}(j)\Delta x) \\
&+ \sum_{j=1}^{N} \frac{\Delta x_j^2}{2}(g_i)_{x_j x_j}(x) + O(\Delta x^3)
\end{aligned} \tag{9.3}$$

by moving higher order terms into the $O(\Delta x^3)$ part.

Next consider the expansion

$$\begin{aligned}
&(g_i)_{x_j}(x+\mathcal{F}(j)\Delta x) \\
&= (g_i)_{x_j}(x+\mathcal{F}(j+1)\Delta x) \\
&+ \Delta x_{j+1}(g_i)_{x_j x_{j+1}}(x+\mathcal{F}(j+1)\Delta x) + O(\Delta t^3)
\end{aligned} \tag{9.4}$$

Repeatedly expanding gives us, up to first order terms,

$$\begin{aligned}
&(g_i)_{x_j}(x+\mathcal{F}(j)\Delta x) \\
&= (g_i)_{x_j}(x) \\
&+ \sum_{k=j+1}^{N} \Delta x_k (g_i)_{x_j x_k}(x+\mathcal{F}(j+1)\Delta x) + O(\Delta x^2).
\end{aligned} \tag{9.5}$$

Again moving the higher order parts of the expansion of $(g_i)_{x_j x_k}(x+\mathcal{F}(j+1)\Delta x)$ in to $O(\Delta x^2)$ gives,

$$\begin{aligned}
&(g_i)_{x_j}(x+\mathcal{F}(j)\Delta x) \\
&= (g_i)_{x_j}(x) \\
&+ \sum_{k=j+1}^{N} \Delta x_k (g_i)_{x_j x_k}(x) + O(\Delta x^2)
\end{aligned} \tag{9.6}$$

We then substitute (9.6) into (9.3), and the second order terms collect as,

$$\sum_{j=1}^{N} \Delta x_j \sum_{k=j+1}^{N} \Delta x_k (g_i)_{x_j x_k}(x) + \sum_{j=1}^{N} \frac{\Delta x_j^2}{2}(g_i)_{x_j x_j}(x). \tag{9.7}$$

Now consider

$$\Delta x^T \hat{J}_i \Delta x = \Delta x^T \begin{pmatrix} \sum_{k=1}^{N} \Delta x_k (g_i)_{x_1 x_k} \\ \sum_{k=1}^{N} \Delta x_k (g_i)_{x_2 x_k} \\ \cdots \end{pmatrix}$$

$$\begin{aligned} &= \sum_{j=1}^{N} \Delta x_j \sum_{k=1}^{N} \Delta x_k (g_i)_{x_j x_k} \\ &= \sum_{j=1}^{N} \Delta x_j \left(\sum_{k=1}^{j-1} \Delta x_k (g_i)_{x_j x_k} + \Delta x_j (g_i)_{x_j x_j} + \sum_{k=j+1}^{N} \Delta x_k (g_i)_{x_j x_k} \right) \end{aligned} \tag{9.8}$$

That is,

$$\begin{aligned} &\frac{1}{2} \Delta x^T \hat{J}_i \Delta x = \\ &\frac{1}{2} \sum_{j=1}^{N} \Delta x_j \sum_{k=1}^{j-1} \Delta x_k (g_i)_{x_j x_k} + \frac{1}{2} \sum_{j=1}^{N} \Delta x_j \sum_{k=j+1}^{N} \Delta x_k (g_i)_{x_j x_k} \\ &+ \sum_{j=1}^{N} \frac{\Delta x_j^2}{2} (g_i)_{x_j x_j} \end{aligned} \tag{9.9}$$

The claim follows from showing (9.7) and (9.9) are equivalent, which requires the proof the following.

$$\sum_{j=1}^{N} \Delta x_j \sum_{k=1}^{j-1} \Delta x_k (g_i)_{x_j x_k} = \sum_{j=1}^{N} \Delta x_j \sum_{k=j+1}^{N} \Delta x_k (g_i)_{x_j x_k}.$$

This is equivalent to,

$$\sum_{j=1}^{N} \sum_{k=1}^{j-1} \beta(j,k) = \sum_{j=1}^{N} \sum_{k=j+1}^{N} \beta(j,k),$$

where the function $\beta(j,k) \equiv \Delta x_j \Delta x_k (g_i)_{x_j x_k}$ is symmetric, that is, $\beta(j,k) = \beta(k,j)$. Now consider the double sum on the left hand side of the equation. It can be viewed as the sum of all the elements of the lower triangular part of a a matrix β with elements $(\beta)_{j,k} = \beta(j,k)$. Similarly, the right hand side of the equation can be viewed as the sum of all the elements of the upper triangular part of β. Since $\beta(j,k)$ is symmetric, the matrix β is symmetric, and thus the two sums are equal. This completes the proof of the claim.

It can also be shown that, if $x = x(t)$ then

$$\frac{dx}{dt}\hat{\mathbf{J}} = \frac{d}{dt}J,$$

where $J = \frac{\partial g}{\partial x}$ is the Jacobian of g. Consider

$$\frac{d}{dt}J = \frac{d}{dt}\begin{pmatrix} (g_1)_{x_1} & (g_1)_{x_2} & \dots \\ (g_2)_{x_1} & (g_2)_{x_2} & \dots \\ \dots & \dots & \dots \end{pmatrix},$$

each entry becomes a series

$$\frac{d}{dt}J = \begin{pmatrix} \sum_{j=1}(g_1)_{x_1x_j}(x_j)_t & \sum_{j=1}(g_1)_{x_2x_j}(x_j)_t & \dots \\ \sum_{j=1}(g_2)_{x_1x_j}(x_j)_t & \sum_{j=1}(g_2)_{x_2x_j}(x_j)_t & \dots \\ \dots & \dots & \dots \end{pmatrix}.$$

Now consider

$$\frac{dx}{dt}\hat{J}_i.$$

It is the row vector,

$$\frac{dx}{dt}\hat{J}_i = \left(\sum_{j=1}(g_i)_{x_1x_j}(x_j)_t, \sum_{j=1}(g_i)_{x_2x_j}(x_j)_t \dots\right).$$

So that we have

$$\frac{dx}{dt}\hat{\mathbf{J}} = \frac{d}{dt}J.$$

Bibliography

[1] H. Omelchenko and H. Karimabadi. Self-adaptive time integration of flux-conservative equations with sources. *Journal of Computational Physics*, 216:179–194, 2006.

[2] R. L. Burden, J. D. Faires, and A. C. Reynolds. *Numerical analysis.* Prindle, Weber & Schmidt, Boston, Mass., 1978.

[3] D. H. Norrie and G. de Vries. *The finite element method.* Academic Press, New York-London, 1973. Fundamentals and applications.

[4] J. Hyman, J. Morel, M. Shashkov, and S. Steinberg. Mimetic finite difference methods for diffusion equations. *Comput. Geosci.*, 6(3-4):333–352, 2002. Locally conservative numerical methods for flow in porous media.

[5] J. C. Butcher. *Numerical methods for ordinary differential equations.* John Wiley & Sons, Ltd., Chichester, second edition, 2008.

[6] J. Bear. *Dynamics of fluids in porous media.* American Elsiver, 1983.

[7] J. H. Cushman, L. S. Bennethum, and B. X. Hu. A primer on upscaling tools for porous media. *Advances in Water Resources*, 25(8):1043–1067, 2002.

[8] M. K. Hubbert. Darcy's law and the field equations of the flow of underground fluids. *Hydrological Sciences Journal*, 2(1):23–59, 1957.

[9] J. Crank. *The mathematics of diffusion.* Clarendon Press, Oxford, second edition, 1975.

[10] R. B. Bird, Warren E. S., and E. N. Lightfoot. *Transport phenomena.* John Wiley & Sons, 2007.

[11] J. Droniou. Finite volume schemes for diffusion equations: introduction to and review of modern methods. *Math. Models Methods Appl. Sci.*, 24(8):1575–1619, 2014.

[12] S. Patankar. *Numerical heat transfer and fluid flow.* CRC Press, 1980.

[13] H. K. Versteeg and W. Malalasekera. *An introduction to computational fluid dynamics: the finite volume method.* Pearson Education, 2007.

[14] R. J. LeVeque. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

[15] K. E. Gustafson. *Introduction to partial differential equations and Hilbert space methods.* John Wiley and Sons, New York-Chichester-Brisbane, 1980.

[16] R. E. Showalter. *Hilbert space methods for partial differential equations.* Pitman, London-San Francisco, Calif.-Melbourne, 1977. Monographs and Studies in Mathematics, Vol. 1.

[17] D. Henry. *Geometric theory of semilinear parabolic equations*, volume 840 of *Lecture Notes in Mathematics.* Springer-Verlag, Berlin, 1981.

[18] T. Barth and M. Ohlberger. Finite volume methods: foundation and analysis. *Encyclopedia of computational mechanics*, 2004.

[19] I. Aavatsmark. An introduction to multipoint flux approximations for quadrilateral grids. *Comput. Geosci.*, 6(3-4):405–432, 2002. Locally conservative numerical methods for flow in porous media.

[20] K. W. Morton and D. F. Mayers. *Numerical solution of partial differential equations.* Cambridge University Press, Cambridge, second edition, 2005. An introduction.

[21] S. M. Cox and P. C. Matthews. Exponential time differencing for stiff systems. *Journal of Computational Physics*, 176:430–455, 2002.

[22] M. Hochbruck and A Osterman. Exponential integrators. *Acta Numerica*, pages 209–286, 2010.

[23] B. Minchev and W. Wright. A review of exponential integrators for first order semilinear problems.

[24] A. K. Kassam and L. N. Trefethen. Fourth-order time-stepping for stiff PDEs. *SIAM J. Sci. Comput.*, 26(4):1214–1233 (electronic), 2005.

[25] M. Caliari, M. Vianello, and L. Bergamaschi. The LEM exponential integrator for advection-diffusion-reaction equations. *J. Comput. Appl. Math.*, 210(1-2):56–63, 2007.

[26] M. Hochbruck, Alexander O., and J. Schweitzer. Exponential Rosenbrock-type methods. *SIAM J. Numer. Anal.*, 47(1):786–803, 2008/09.

[27] M. Caliari and A. Ostermann. Implementation of exponential Rosenbrock-type integrators. *Appl. Numer. Math.*, 59(3-4):568–581, 2009.

[28] M. Hochbruck and A. Ostermann. Exponential Runge-Kutta methods for parabolic problems. *Appl. Numer. Math.*, 53(2-4):323–339, 2005.

[29] M. Hochbruck and A. Ostermann. Explicit exponential Runge-Kutta methods for semilinear parabolic problems. *SIAM J. Numer. Anal.*, 43(3):1069–1090 (electronic), 2006.

[30] V. T. Luan and A. Ostermann. Explicit exponential runge–kutta methods of high order for parabolic problems. *Journal of Computational and Applied Mathematics*, 256:168–179, 2014.

[31] M. Tokman. Efficient integration of large stiff systems of odes with exponential propagation iterative (epi) methods. *Journal of Computational Physics*, 213(2):748–776, 2006.

[32] M. Tokman. A new class of exponential propagation iterative methods of Runge-Kutta type (EPIRK). *J. Comput. Phys.*, 230(24):8762–8778, 2011.

[33] M. Tokman, J. Loffeld, and P. Tranquilli. New adaptive exponential propagation iterative methods of Runge-Kutta type. *SIAM J. Sci. Comput.*, 34(5):A2650–A2669, 2012.

[34] A. Tambue, G. J. Lord, and S. Geiger. An exponential integrator for advection-dominated reactive transport in heterogeneous porous media. *Journal of Computational Physics*, 229(10):3957–3969, 2010.

[35] P. E. Kloeden, G. J. Lord, A. Neuenkirch, and T. Shardlow. The exponential integrator scheme for stochastic partial differential equations: pathwise error bounds. *J. Comput. Appl. Math.*, 235(5):1245–1260, 2011.

[36] G. J. Lord and A. Tambue. Stochastic exponential integrators for the finite element discretization of SPDEs for multiplicative and additive noise. *IMA J. Numer. Anal.*, 33(2):515–543, 2013.

[37] G. J. Lord and A. Tambue. A modified semi–implict euler-maruyama scheme for finite element discretization of spdes. *arXiv preprint arXiv:1004.1998*, 2010.

[38] A. Tambue, I. Berre, and J. M. Nordbotten. Efficient simulation of geothermal processes in heterogeneous porous media based on the exponential rosenbrock–euler and rosenbrock-type methods. *Advances in Water Resources*, 53:250–262, 2013.

[39] A. Tambue and J. M. T. Ngnotchouye. Weak convergence for a stochastic exponential integrator and finite element discretization of spde for multiplicative\ & additive noise. *arXiv preprint arXiv:1507.07153*, 2015.

[40] A. Tambue. *Efficient Numerical Schemes for Porous Media Flow.* PhD thesis, Heriot-Watt, 2010.

[41] M. J. Gander and S. Güttel. PARAEXP: a parallel integrator for linear initial-value problems. *SIAM J. Sci. Comput.*, 35(2):C123–C142, 2013.

[42] J. L. Lions, Y. Maday, and G. Turinici. Résolution d'EDP par un schéma en temps "pararéel". *C. R. Acad. Sci. Paris Sér. I Math.*, 332(7):661–668, 2001.

[43] A. Martínez, L. Bergamaschi, M. Caliari, and M. Vianello. A massively parallel exponential integrator for advection-diffusion models. *J. Comput. Appl. Math.*, 231(1):82–91, 2009.

[44] L. Bergamaschi, M. Caliari, and M. Vianello. The ReLPM exponential integrator for FE discretizations of advection-diffusion equations. In *Computational science—ICCS 2004. Part IV*, volume 3039 of *Lecture Notes in Comput. Sci.*, pages 434–442. Springer, Berlin, 2004.

[45] R. B. Sidje. Expokit: a software package for computing matrix exponentials. *ACM Transactions on Mathematical Software (TOMS)*, 24(1):130–156, 1998.

[46] H. Berland, B. Skaflestad, and W. M. Wright. Expint—a matlab package for exponential integrators. *ACM Transactions on Mathematical Software (TOMS)*, 33(1):4, 2007.

[47] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Rev.*, 20(4):801–836, 1978.

[48] A. H. Al-Mohy and N. J. Higham. Computing the action of the matrix exponential, with an application to exponential integrators. *SIAM J. Sci. Comput.*, 33(2):488–511, 2011.

[49] I. Moret and P. Novati. An interpolatory approximation of the matrix exponential based on Faber polynomials. *J. Comput. Appl. Math.*, 131(1-2):361–380, 2001.

[50] J. Baglama, D. Calvetti, and L. Reichel. Fast Leja points. *Electron. Trans. Numer. Anal.*, 7:124–140, 1998. Large scale eigenvalue problems (Argonne, IL, 1997).

[51] L. Bergamaschi, M. Caliari, A. Martinez, and M. Vianello. Comparing Leja and Krylov approximations of large scale matrix exponentials. In *Computational Science–ICCS 2006*, pages 685–692. Springer, 2006.

[52] M. Caliari, M. Vianello, and L. Bergamaschi. Interpolating discrete advection-diffusion propagators at Leja sequences. *J. Comput. Appl. Math.*, 172(1):79–99, 2004.

[53] Y. Saad. Analysis of some Krylov subspace approximations to the matrix exponential operator. *SIAM J. Numer. Anal.*, 29(1):209–228, 1992.

[54] M. Hochbruck and C. Lubich. On Krylov subspace approximations to the matrix exponential operator. *SIAM J. Numer. Anal.*, 34(5):1911–1925, 1997.

[55] J. Niesen and W. Wright. A Krylov subspace algorithm for evaluating the ϕ-functions in exponential integrators. *arXiv:0907.4631v1*, 2009.

[56] M. Tokman and J. Loffeld. Efficient design of exponential-Krylov integrators for large scale computing. *Procedia Computer Science*, 1(1):229–237, 2010.

[57] E. Isaacson and H. B. Keller. *Analysis of numerical methods.* Courier Corporation, 2012.

[58] E. Emmrich. *Discrete versions of Gronwall's lemma and their application to the numerical analysis of parabolic problems.* Technische Universität Berlin. Fachbereich 3-Mathematik, 1999.

[59] G. J. Lord, C. E. Powell, and T. Shardlow. *An Introduction to Computational Stochastic PDEs.* Number 50. Cambridge University Press, 2014.

[60] C. Van Loan. The sensitivity of the matrix exponential. *SIAM Journal on Numerical Analysis*, 14(6):971–981, 1977.

[61] A. Prothero and A. Robinson. On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations. *Math. Comp.*, 28:145–162, 1974.

[62] R. Frank, J. Schneid, and C. W. Ueberhuber. The concept of B-convergence. *SIAM J. Numer. Anal.*, 18(5):753–780, 1981.

[63] E. Hairer and G. Wanner. *Solving ordinary differential equations. II*, volume 14 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2010. Stiff and differential-algebraic problems, Second revised edition, paperback.

[64] J. R. Cash. Efficient numerical methods for the solution of stiff initial-value problems and differential algebraic equations. *R. Soc. Lond. Proc. Ser. A Math. Phys. Eng. Sci.*, 459(2032):797–815, 2003.

[65] H. Omelchenko and H. Karimabadi. Event-driven, hybrid particle-in-cell simulation: A new paradigm for multi-scale plasma modeling. *Journal of Computational Physics*, 216:153–178, 2006.

[66] T. Unfer, Jean-Pierre Boeuf, F. Rogier, and F. Thivet. An asynchronous scheme with local time stepping for multi-scale transport problems: application to gas discharges. *J. Comput. Phys.*, 227(2):898–918, 2007.

[67] H. Karimabadi, J. Driscoll, Y. A. Omelchenko, and N. Omidi. A new asynchronous methodology for modeling of physical systems: breaking the curse of courant condition. *Journal of Computational Physics*, 205(2):755–775, 2005.

[68] B. F. Sanders. Integration of a shallow water model with a local time step. *Journal of Hydraulic Research*, 46(4):466–475, 2008.

[69] H. Karimabadi, J. Driscoll, J. Dave, Y. Omelchenko, K. Perumalla, R. Fujimoto, and N. Omidi. Parallel discrete event simulations of grid-based models: Asynchronous electromagnetic hybrid code. In Jack Dongarra, Kaj Madsen, and Jerzy Waniewski, editors, *Applied Parallel Computing. State of the Art in Scientific Computing*, volume 3732 of *Lecture Notes in Computer Science*, pages 573–582. Springer Berlin Heidelberg, 2006.

[70] Y. A. Omelchenko and H. Karimabadi. HYPERS: a unidimensional asynchronous framework for multiscale hybrid simulations. *J. Comput. Phys.*, 231(4):1766–1780, 2012.

[71] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The journal of physical chemistry A*, 104(9):1876–1889, 2000.

[72] L. F. Shampine. Error estimation and control for ODEs. *J. Sci. Comput.*, 25(1-2):3–16, 2005.

[73] S. Osher and R. Sanders. Numerical approximations to nonlinear conservation laws with locally varying time and space grids. *Math. Comp.*, 41(164):321–336, 1983.

[74] V. Savcenco, W. Hundsdorfer, and J. G. Verwer. A multirate time stepping strategy for stiff ordinary differential equations. *BIT*, 47(1):137–155, 2007.

[75] N. F. Otani. Computer modeling in cardiac electrophysiology. *Journal of Computational Physics*, 161(1):21–34, 2000.

[76] C. W. Gardiner. Handbook of stochastic methods for physics, chemistry and the natural sciences. *Applied Optics*, 25:3145, 1986.

[77] D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of computational physics*, 22(4):403–434, 1976.

[78] R. Erban, J. Chapman, and P. Maini. A practical guide to stochastic simulations of reaction-diffusion processes. *arXiv preprint arXiv:0704.1908*, 2007.

[79] R. B. Lowrie, P. L. Roe, and B. Van Leer. Space-time methods for hyperbolic conservation laws. In *Barriers and Challenges in Computational Fluid Dynamics*, pages 79–98. Springer, 1998.

[80] A. Üngör and A. Sheffer. Pitching tents in space-time: Mesh generation for discontinuous galerkin method. *International Journal of Foundations of Computer Science*, 13(02):201–221, 2002.

[81] J. R. Appleyard, I. M. Cheshire, et al. The cascade method for accelerated convergence in implicit simulators. In *European Petroleum Conference*. Society of Petroleum Engineers, 1982.

[82] F. Kwok and H. Tchelepi. Potential-based reduced newton algorithm for non-linear multiphase flow in porous media. *Journal of Computational Physics*, 227(1):706–727, 2007.

[83] R. P. Batycky, M. J. Blunt, M. R. Thiele, et al. A 3d field-scale streamline-based reservoir simulator. *SPE Reservoir Engineering*, 12(04):246–254, 1997.

[84] M. J. Gautier, Y.and Blunt and M. A. Christie. Nested gridding and streamline-based simulation for fast reservoir performance prediction. *Computational Geosciences*, 3(3-4):295–320, 1999.

[85] J. E. Aarnes, V. Kippe, and K. A. Lie. Mixed multiscale finite elements and streamline methods for reservoir simulation of large geomodels. *Advances in Water Resources*, 28(3):257–271, 2005.

[86] P. L. Roe. Approximate riemann solvers, parameter vectors, and difference schemes. *Journal of computational physics*, 43(2):357–372, 1981.

[87] E. F. Toro. *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*. Springer Science & Business Media, 2009.

[88] K. A. Lie, S. Krogstad, I. S. Ligaarden, H. M. Natvig, J. R.and Nilsen, and B. Skaflestad. Open-source matlab implementation of consistent discretisations on complex grids. *Computational Geosciences*, 16(2):297–322, 2012.

[89] R. I. Masel. *Principles of adsorption and reaction on solid surfaces*, volume 3. John Wiley & Sons, 1996.

[90] H. Press, A. Teukolsky, T. Vetterling, and P. Flannery. Numerical recipes in c++. the art of computer programming, 2002.

[91] M. Hausner and J. T. Schwartz. *Lie groups; Lie algebras*. Gordon and Breach Science Publishers, New York-London-Paris, 1968.

[92] K. Goldberg. The formal power series for $\log e^x e^y$. *Duke Math. J.*, 23:13–21, 1956.

[93] R. C. Thompson. Convergence proof for Goldberg's exponential series. *Linear Algebra Appl.*, 121:3–7, 1989. Linear algebra and applications (Valencia, 1987).

[94] M. Newman and R. C. Thompson. Numerical values of Goldberg's coefficients in the series for $\log(e^x e^y)$. *Math. Comp.*, 48(177):265–271, 1987.

[95] H. Yoshida. Construction of higher order symplectic integrators. *Phys. Lett. A*, 150(5-7):262–268, 1990.

[96] E. Carr, T. Moroney, and I. Turner. Efficient simulation of unsaturated flow using exponential time integration. *Applied Mathematics and Computation*, 217:6587–6596, 2011.

[97] Y Saad. *Iterative Methods for Sparse Linear Systems.* ITP, 1996.

[98] P. Tranquilli and A. Sandu. Exponential-Krylov methods for ordinary differential equations. *J. Comput. Phys.*, 278:31–46, 2014.

[99] K. Strehmel and R. Weiner. *Linear-implizite Runge-Kutta-Methoden und ihre Anwendung*, volume 127. BG Teubner, 1992.

[100] J. Dongarra et al. The international exascale software project roadmap. *International Journal of High Performance Computing Applications*, 2011.

[101] D.A. Donzis and K. Aditya. Asynchronous finite-difference schemes for partial differential equations. *J. Comput. Phys.*, 274:370–392, 2014.

[102] O. Toselli, A.and Widlund. *Domain decomposition methods: algorithms and theory*, volume 3. Springer, 2005.

[103] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT Press, Cambridge, MA, third edition, 2009.

[104] P. Berlioux and P. Bizard. *Algorithms. 2*. John Wiley & Sons, Ltd., Chichester, 1990. Data structures and search algorithms, Translated from the French by Jack Howlett.

[105] G. Brassard and P. Bratley. *Algorithmics: theory & practice*. Prentice-Hall, Inc., 1988.