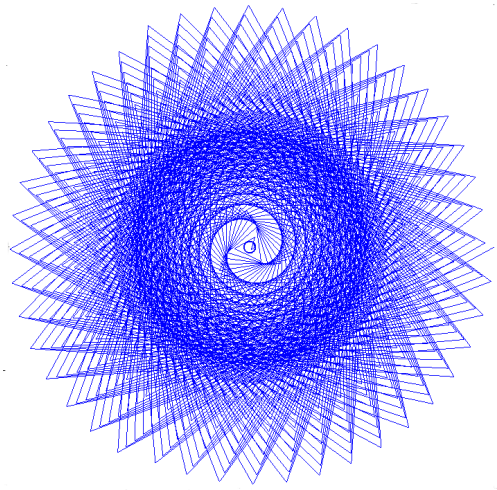
Dynamical Numerics for Numerical Dynamics

Robert H. C. Moir



Contents

1 Introduction

A wide variety of physical and social systems are modeled as continuous-time dynamical systems. And since most phenomena are nonlinear the models obtained for such systems are difficult to analyze analytically. This makes the use of numerical methods and computation an essential tool for the analysis of dynamical system models. Since numerical methods involve the approximation of continuously evolving systems by a discrete algorithm amenable to computer processing, there arises the crucially important problem of determining whether the discretization of the model faithfully represents the actual behaviour of solutions to the model. In order to address this question one requires a formal approach that enables the proof of powerful results about the ability of numerics to reproduce the behaviour of the model. One fruitful approach is to treat the numerics themselves as discrete dynamical systems so that the theory of dynamical systems can be brought to bear on the problem. This paper is concerned with an exploration of the basics of this approach.

There are two general kinds of results that are proved concerning the ability of numerics to faithfully reproduce features of the model. One kind of question concerns the closeness of phase trajectories of the numerical simulation to the actual trajectories of the dynamical system. This becomes analyzed as a *convergence* question. The results addressing this question concern when and whether solutions generated by numerics converge to the true solution in the limit as the time-step of the numerical method goes to zero, and what is the rate of the convergence. Convergence results can be proved for finite-time evolution, where one is concerned with the *global error*, *i.e.* the overall distance between the computed trajectory and the actual trajectory, going to zero as the time-step Δt goes to zero. Convergence results can also be proved for infinite-time evolution, where one is concerned with whether trajectories of the numerical solution converge on ω-limit sets of the dynamical system in the limit where $\Delta t \to 0$. This is one area where the dynamical systems approach to numerics is useful. The focus of this paper, however, is on the second general kind of question.

The other kind of question concerns whether the invariant sets of the numerical method, *e.g.* equilibria, steady state solutions, attractors, *etc.*, actually correspond to the invariant sets of the dynamical system. The kind of results that can be proved are results concerning the structure preserving properties of particular classes of numerical methods under certain conditions, and the structure preserving properties of general numerical methods under mild or no restrictions on the time-step. It is very difficult to prove results on very weak assumptions, so assumptions are usually made about the properties of the numerical method or the structure of the vector field of the dynamical system or both. The dynamical systems approach to numerics is also useful for proving stability results. This paper is primarily an examination of the basics of this kind of numerical analysis.

The breakdown of the sections of the paper are as follows. Section 2 just reviews some basic definitions from the theory of dynamical systems and establishes notation. Section 3 discusses the particular numerical methods considered in this paper, *viz.* the forward and backward Euler methods, the one- and two-stage theta methods and the classic RK4 method, as well as the MATLAB code for these methods written for use with this paper. Section 4 is concerned with a brief consideration of the issue of establishing conditions under which a numerical method defines a dynamical system. Section 5, the main section of the paper, considers some global stability properties. The first part

is a detailed examination and proof of results concerning the linear decay test problem. This is to motivate and establish a basis for stability concepts for Runge-Kutta methods. The second part is a consideration of the preservation of the dissipative structure of a dynamical system. The results of theorems for the numerical methods discussed in section 3 are explored and the consequences of the theorems explored with simulations. The final section is a brief consideration of the deep and difficult question of when and whether attractors of dynamical systems are preserved under discretization. The limited conclusion of the paper is that simulations may provide a useful guide for the discovery of new theorems in numerical analysis.

2 Dynamical Systems

There are a number of concepts from the theory of dynamical systems that will be required for what follows. Many of the concepts from dynamical systems theory are assumed, but ones that require careful or clear definition will be provided in this section. The most basic distinction is between what I am calling discrete and continuous dynamical systems. A *discrete dynamical system* on a subset $E \subseteq \mathbb{R}^p$ is constituted by a map

$$U_{n+1} = f(U_n),$$

where f is a function from $\mathbb{R}^p$ to $\mathbb{R}^p$, for which there exists a unique solution sequence $\{U_n\}_{n=0}^{\infty}$ for all $U \in E$ that remains in E for all $n \geq 0$. This definition can be generalized to include non-unique solution sequences. Such a system is called a *generalized discrete dynamical system.* Each dynamical system generates an operator S^n: $E \to E$ defined by $U_n = S^n U_0$ called the *evolution semigroup* of the system. The collection of semigroups S^n for all $n \geq 0$ form a commutative monoid with identity S^0. The *action* of a semigroup S^n on a set of initial data B is defined by

$$S^n B = \bigcup_{U \in B} S^n U.$$

For a generalized discrete dynamical system the *generalized evolution semigroup* T^n can also be defined.

A *continuous dynamical system* on a subset $E \subseteq \mathbb{R}^p$ is constituted by a system of ordinary differential equations

$$\dot{u} = f(u), \tag{2.1}$$

where $f(u)$ is a function from $\mathbb{R}^p$ to $\mathbb{R}^p$, such that for all $U \in E$ there exists a unique solution that is defined for all $t \in [0, \infty)$ and remains inside E for all $t \in [0, \infty)$. f is written as an autonomous function of u for simplicity, but there is no loss of generality here since a non-autonomous system may always be converted into an autonomous one. Each continuous dynamical system generates an *evolution semigroup* $S(t)$: $E \to E$ defined by $u(t) = S(t)u(0)$ for all $t \geq 0$. The collection of semigroups $S(t)$ for all $t \geq 0$ forms a commutative monoid with identity $S(0)$.

A possible ambiguity with the distinction between discrete and continuous dynamical systems arises because some authors reserve the term 'continuous dynamical system' for a dynamical system for which the evolution semigroup is a continuous function of the initial data. I will use the term

well-posed to describe the case where the evolution semigroup is continuous, for both discrete and continuous dynamical systems.

An *invariant set* for a dynamical system is a set that is invariant under the action of the evolution semigroup S^n for all $n \geq 0$, or $S(t)$ for all $t \geq 0$, as the case may be. A set A *attracts* a set B under $S(t)$ (resp., S^n) if for any $\varepsilon > 0$ there exists a $t^* = t^*(\varepsilon, B, A)$ (resp., $n^* = n^*(\varepsilon, B, A)$) such that $S(t) \subset \mathcal{N}(A, \varepsilon)$ (resp., $S^n \subset \mathcal{N}(A, \varepsilon)$) for all $t \geq t^*$ (resp. $n \geq n^*$).[1] A compact invariant set is an *attractor* if it attracts an open neighbourhood of itself. A *global attractor* is an attractor that attracts every bounded set in $\mathbb{R}^p$.

A compact set Λ is *uniformly stable* if for every $\varepsilon > 0$ there exists a $\delta = \delta(\varepsilon) > 0$ such that if $\text{dist}(U, A) < \delta$ then $\text{dist}(S(t)U, \Lambda) < \varepsilon$ for all $t \geq 0$.[2] A compact set is *uniformly asymptotically stable* if it is both asymptotically stable and uniformly stable. Similar definitions apply for discrete dynamical systems. An important result that applies in both the discrete and continuous cases is that an attractor is uniformly asymptotically stable and if Λ is uniformly asymptotically stable then $\mathcal{A} = \omega(\Lambda) \subseteq \Lambda$ is an attractor.

We will be interested in treating the discretization of (2.1) by a fixed time-step numerical method as a discrete dynamical system. In such a case an evolution semigroup of the numerical method is written as $S^n_{\Delta t}$, where Δt is the time-step. Attracting sets are similarly indexed.

We now move on to a consideration of the fixed time-step Runge-Kutta methods considered in this paper and the MATLAB code written for them.

3 Numerical Methods

For simplicity, the numerical methods used in this paper are all fixed time-step Runge-Kutta methods. The MATLAB file `ode23tx`, a textbook version of MATLAB built-in `ode23` solver available from the Mathworks website for Cleve Moler's book Numerical Computing with MATLAB, was used as a code base for the numerical methods. All of the methods may be called without output arguments and plot an emerging solution using `odeplot` or with arguments `[t, y]`, which returns a vector `t` of times and an array `y` where `y(:,k)` is the solution at `t(k)`. The code for each of the methods written for this paper are available in appendix A.

Two of the methods are explicit Runge-Kutta methods. The first, and simplest, is `euler`, which uses the forward Euler method. Calls to `euler` are of the form

```
euler(F, tspan, y0, h)
```

The arguments of this method are common to all of the numerical methods written for this paper: `F` is a function handle or anonymous function defining the right hand side $\mathbf{f}(t, \mathbf{y})$ of a system of differential equations

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y});$$

[1] $\mathcal{N}(A, \varepsilon)$ denotes an ε-neighourhood of the set A.

[2] The distance from an element a to a set B is defined as $\text{dist}(a, B) = \inf_{b \in B} \|a - b\|$.

`tspan` is a vector `[t0 tf]`, where `t0` is the start time of the integration and `tf` is the final time of the integration and `tf` can be less that `t0`, so the integration can process backward in time; `y0` is a vector of initial values; and `h` is the length of the time-step to be used in the integration.

The second explicit Runge-Kutta method is `rk4`, which uses the classic fourth-order Runge-Kutta method with Butcher tableau

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

Calls to `rk4` have the form `rk4(F, tspan, y0, h)`. The arguments are the same as for `euler` above.

The rest of the methods used are implicit Runge-Kutta methods. The simplest is `beuler`, which uses the backward Euler method. Calls to `beuler` have the form

```
beuler(F, DF, tspan, y0, h, tol, Nmax)
```

The inputs `F`, `tspan`, `y0` and `h` have the same function as the previous methods. `DF` is the Jacobian of `F`, which is used to solve the, in general, nonlinear implicit equation at each step. The nonlinear equation is solved at each step using Newton's method. For the backward Euler method, the implicit equation that must be solved is

$$U_{n+1} = U_n + \Delta t\, \mathbf{f}(U_{n+1}).$$

The approach is to use Netwon's method to find the zero of the function

$$\mathcal{F}(x) = (U_n - x) + \Delta t\, \mathbf{f}(x). \tag{3.1}$$

The Jacobian of this function is

$$D\mathcal{F}(x) = -I + \Delta t\, D\mathbf{f}(x).$$

Expanding $\mathcal{F}(x)$ about its zero x^* to first order we obtain

$$\mathcal{F}(x) = D\mathcal{F}(x)(x - x^*).$$

So by letting $x = x_n$ and $x^* = x_{n+1}$ we obtain

$$D\mathcal{F}(x_n)(x_n - x_{n+1}) = \mathcal{F}(x_n). \tag{3.2}$$

and inverting the system we obtain

$$x_{n+1} = x_n - (D\mathcal{F}(x_n))^{-1}\mathcal{F}(x_n). \tag{3.3}$$

This is the recurrence used to find the root of (3.1). `beuler` uses a different method depending on whether the system of equations is a scalar or a vector system. The one dimensional case runs Newton's method using ordinary arithmetic. For the vector case, where matrix operations must be used, the inverse of the Jacobian $D\mathcal{F}$ is not computed at each step of the iteration, rather

the MATLAB backslash operator is used to solve the system 3.2. `tol` is the target tolerance of the Newton iteration and `Nmax` is an adjustable maximum number of iterations for the Newton iteration.

The remaining methods are theta methods, the code for which is based on that of `beuler`. First we have `theta1`, which uses the one-stage theta method, a general one-stage Runge-Kutta method. The Butcher tableau for this method is

$$\begin{array}{c|c} \theta & \theta \\ \hline & 1 \end{array}$$

with $\theta \in [0, 1]$. So the implicit equation that must be solved at each step is

$$U_{n+1} = U_n + \Delta t\, \mathbf{f}\left((1-\theta)U_n + \theta U_{n+1}\right). \tag{3.4}$$

The approach used is to find the zero of the function

$$\mathcal{F}(x) = (U_n - x) + \Delta t\, \mathbf{f}\left((1-\theta)U_n + \theta x\right),$$

which has Jacobian

$$D\mathcal{F}(x) = -I + \theta \Delta t\, D\mathbf{f}\left((1-\theta)U_n + \theta x\right),$$

using Newton's method. Calls to `theta1` have the form

```
theta1(F, DF, tspan, y0, h, theta, tol, Nmax)
```

The inputs are identical to those of `beuler` except for `theta`, which is the value of θ for the method. For $\theta = 0$ the method becomes identical for forward Euler and for $\theta = 1$ the method becomes identical to backward Euler. For $\theta = 0.5$ the method becomes the implicit midpoint rule. This is the main use of `theta1` and the one-stage theta method in this paper.

The final method is `theta2`, which uses the two-stage theta method. The Butcher tableau for this method is

$$\begin{array}{c|cc} 0 & & \\ 1 & 1-\theta & \theta \\ \hline & 1-\theta & \theta \end{array}$$

with $\theta \in [0, 1]$. So the implicit equation that must be solved at each step is

$$U_{n+1} = U_n + \Delta t\left[(1-\theta)\mathbf{f}(U_n) + \theta \mathbf{f}(U_{n+1})\right]. \tag{3.5}$$

The approach used is to find the zero of the function

$$\mathcal{F}(x) = (U_n - x) + \Delta t\left[(1-\theta)\mathbf{f}(U_n) + \theta \mathbf{f}(x)\right],$$

which has Jacobian

$$D\mathcal{F}(x) = -I + \theta \Delta t\, D\mathbf{f}\left(x\right),$$

using Newton's method. Calls to `theta2` have the form

```
theta2(F, DF, tspan, y0, h, theta, tol, Nmax)
```

The inputs are identical to those of `theta1`. Just like `theta1`, for $\theta = 0$ the method becomes identical for forward Euler and for $\theta = 1$ the method becomes identical to backward Euler. For $\theta = 0.5$ the method becomes the trapezoidal rule. This is the main use of `theta2` and the two-stage theta method in this paper.

4 Numerical Methods as Dynamical Systems

The focus of this paper is on the sort of results that are available concerning the stability of numerical methods that can be treated as dynamical systems. Hence, it is an important question to determine the conditions under which the discretization of a system of differential equations used by a numerical method defines a discrete dynamical system. This section is devoted to a brief consideration of results in this direction.

Many existence and uniqueness results for systems of differential equations, including (2.1), rely on the function f satisfying some kind of Lipschitz condition. Lipschitz conditions can also be used to determine conditions such that under discretization, the semigroup $S(t)$ defined by (2.1) defines a semigroup $S^n_{\Delta t}$ for the numerical method dependent on the choice of time-step Δt. Due to the importance of well-posedness for physical applications we are interested in conditions under which the semigroup $S^n_{\Delta t}$ is continuous, or well-posed, when $S(t)$ is. This section is just a few basic results along these lines.

A general s-stage fixed time-step Runge-Kutta method for the solution of (2.1) has the form:

$$Y_i = U_n + \Delta t \sum_{j=1}^{s} a_{ij} f(Y_j), \qquad i = 1, \ldots, s, \tag{4.1}$$

$$U_{n+1} = U_n + \Delta t \sum_{i=1}^{s} b_i f(Y_i), \qquad U_0 = U, \tag{4.2}$$

where U_n is the approximation of $u(t_n)$ with $t_n = n\Delta t$, Δt is the fixed step-size and the Y_i are the *stage values*. The b_i are called the *weights* of the method and for consistent methods satisfy the condition $\sum_{i=1}^{s} b_i = 1$. The b_i determine a vector b and the values a_{ij} determine a matrix A. If the entries of A satisfy the condition

$$a_{ij} = 0 \qquad \forall 1 \leq i \leq j \leq s,$$

then the Runge-Kutta method is *explicit* and time-steps are directly calculable. Otherwise the method is *implicit* and an indirect method of calculation of the state values must be used.

The theorems cited in this section are from Stuart & Humphries (1996) and proofs are available there. For explicit Runge-Kutta methods the following result is available:

Theorem 4.1 (Explicit Runge-Kutta Methods as Dynamical Methods: Locally Lipschitz Problems) Let f be locally Lipschitz. Then, if the Runge-Kutta system (4.1), (4.2) is explicit, it generates a well-posed dynamical system.

Thus, under conditions such that local solutions exist for (2.1) and are unique, the discretization using an explicit Runge-Kutta method generates a well-posed dynamical system. This is interesting since for $f = u^2$, which is locally Lipschitz, the system (2.1) does not define a dynamical system since solutions blow-up, even though its discretization with an explicit Runge-Kutta method does.

This is because iterations of a continuous map cannot blow-up in a finite number of iterations. Taking the scalar case and the Euler method as an example, the solution of (2.1) for this f is

$$u(t) = \frac{1}{1/u_0 - t},$$

which blows-up in finite time, but the discretization $U_{n+1} = U_n + \Delta t U_n^2$ does not since it can only increase a finite amount at each step.

So, explicit methods are always solvable but for implicit methods we must worry about whether the method is solvable. We have the following result:

Theorem 4.2 (Solvability of Runge-Kutta Equations: Globally Lipschitz Problems) If f is globally Lipschitz with Lipschitz constant L and

$$\Delta t < \frac{1}{L\|A\|_\infty},$$

where $\|A\|_\infty$ is the induced ∞-norm of A, then the equations (4.1) are uniquely solvable.

Stuart & Humphries (1996) also provide a fixed point method to find the stage values. This result yields the following

Corollary 4.3 (Runge-Kutta Methods as Dynamical Systems: Globally Lipschitz Problems) If f is globally Lipschitz with Lipschitz constant L and if

$$\Delta t < \frac{1}{L\|A\|_\infty},$$

then equations (4.1), (4.2) define a well-posed dynamical system.

If f is globally Lipschitz then (2.1) defines a well-posed dynamical system and this result shows that the discretization by an implicit Runge-Kutta method does also.

Although this last result has a nice strong conclusion, the assumption of a globally Lipschitz f is too strong for the majority of cases—nonlinear problems will generally not be globally Lipschitz. So the ideal kind of result that we desire is that the discretization defines a well-posed dynamical system whenever (2.1) does. We may obtain something along these lines. For systems of ODE, provided that f is locally Lipschitz and there is an *a priori* condition on f that ensures boundedness of solutions, then (2.1) defines a well-posed dynamical system. The following result establishes a similar result for Runge-Kutta methods.

Theorem 4.4 (Runge-Kutta Methods on Locally Lipschitz Problems) Let f be locally Lipschitz and assume that there is an *a priori* bound that implies that for some bounded set B, if $U_n \in B$ then any solution of (4.1), (4.2) satisfies $U_{n+1} \in B$. Then it follows that there exists $\Delta t_c = \Delta t_c(B) > 0$ such that for $\Delta t \in (0, \Delta t_c)$ the Runge-Kutta method (4.1), (4.2) defines a well-posed dynamical system on B.

Again, Stuart & Humphries (1996) also provide a fixed point method to find the stage values.

Stuart & Humphries (1996) also provide a number of results that give conditions under which Runge-Kutta methods define dynamical systems or generalized dynamical systems when the vector field f satisfies some kind of structural property, including the structural properties to be considered in the next section, *viz.* linear decay and dissipative systems. We are not concerned with detailing these results here and as required we will assume the the solvability conditions are met.

5 Global Stability Properties

In this section we consider the stability properties of Runge-Kutta methods applied to problems where the vector field f in (2.1) has a certain kind of global structural property that ensures that solutions do not diverge to infinity as $t \to \infty$. This is an important class to consider since such systems arise commonly in practice. The question we are interested in here is whether the ω-limit sets $\omega(U)$ of the dynamical system $S(t)$ agree with the ω-limit sets $\omega_{\Delta t}(U)$ of the discretized dynamical system $S^n_{\Delta t}$.

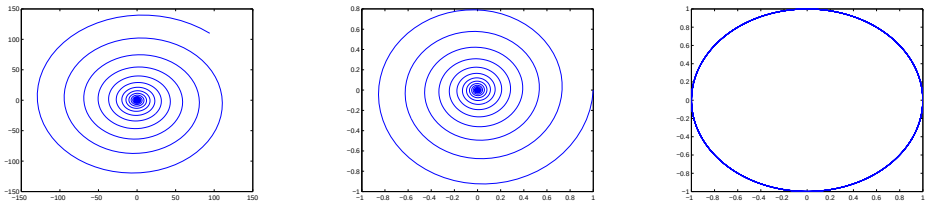
As an example of the sort of thing that can go wrong consider a simple problem, the simple harmonic oscillator:

$$\dot{u} = Au, \quad u(0) = (U, V)^T \qquad A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.$$

The equilibrium point at the origin is a centre, so the invariant sets of the semigroup $S(t)$ are circles centred at the origin. Thus, if we are using a numerical method to compute the solution we would want its semigroup $S^n_{\Delta t}$ to share this property. But not all numerical methods do. We see in figure 1 that different numerical methods have quite different results when applied to this problem. We see from figure 1(a) that the forward Euler method diverges to infinity as $t \to \infty$, and figure 1(b) shows that the backward Euler method converges in on the origin. Thus, both forward and backward Euler do a very poor job of discretizing the simple harmonic oscillator. One might think that these results are of little practical interest, since we are just considering simple fixed time-step methods. From figure 2, however, we see that running the simulation for longer times, $t \in [0, 5000]$, both the implicit midpoint rule and the classic Runge-Kutta method do quite well, but one of MATLAB's best adaptive time-step ordinary differential equation solvers, `ode45`, actually slowly converges to the origin. This shows the practical importance of stability considerations, even on simple problems. The simple harmonic oscillator is a conservative system, which raises special stability issues. In this section, however, we are only interested in dissipative systems.

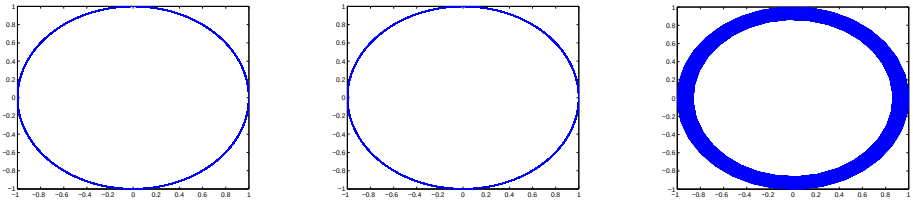
Another thing that can go wrong with discretization is the introduction of spurious invariant sets by the discretization, including spurious equilibria and periodic solutions. There are a great many results that can be proved about this phenomenon, but consideration of spurious invariant sets is beyond the scope of this paper.

We now turn to consider some of the theoretical results that are available for vector fields with certain particular structural properties, *viz.*, linear decay systems and dissipative systems.



(a) `euler` (b) `beuler` (c) `theta1`, $\theta = 0.5$

Figure 1: Computation of the simple harmonic oscillator, with $u(0) = (1, 0)$, using different numerical methods. For each method, $t \in [0, 100]$ and $\Delta t = 0.1$.



(a) `theta1`, $\theta = 0.5$ (b) `rk4` (c) `ode45`

Figure 2: Computation of the simple harmonic oscillator, with $u(0) = (1, 0)$, using different numerical methods. For each of the simulations $t \in [0, 5000]$ and for the first two methods $\Delta t = 0.1$.

5.1 Linear Decay Systems

The importance of the eigenvalues of the Jacobian Df of the vector field f in stability analysis makes linear test problems useful to consider. And since we here are interested in the replication of ω-limit sets by numerical methods, the problem of *linear decay*

$$\dot{u} = \lambda u, \qquad \lambda \in \mathbb{C}, \qquad Re(\lambda) \leq 0, \tag{5.1}$$

is a useful place to start. Since the solution to this problem is $u(t) = Ue^{\lambda t}$, it is easily seen that for for two solutions $u(t)$ and $v(t)$

$$\|u(t) - v(t)\| \leq \|U - V\|$$

for all $t \geq 0$, since $\|u(t) - v(t)\| = Re(\lambda)\|U - V\|$. In case that the inequality in (5.1) is strict, it is therefore seen that this system has the property that $\omega(U) = 0$ for any initial value $U \in \mathbb{C}$. Thus, we are interested in the conditions under which numerical methods replicate this behaviour.

Applied to this problem, Runge-Kutta methods can be written in the form

$$U_{n+1} = R(\lambda \Delta t)U_n, \qquad R(z) = 1 + zb^T(1 - zA)^{-1}\mathbb{1}_s, \tag{5.2}$$

where b is the vector of weights in equation (4.2) and A is the matrix of multipliers appearing in (4.1) (b and A are determined by the Butcher tableau of the method as explained above), s is the order of the method and $\mathbb{1}_s = (1, 1, \ldots, 1)^T \in \mathbb{R}^s$. This result follows easily from the linearity of the vector field f in (5.1). It follows from this that $\|U_n\| = |R(\lambda \Delta t)|^n \|U_0\|$, so for the method to be stable on this problem we require that $|R(\lambda \Delta t)| \leq 1$. The region $\mathcal{S}$ of the complex plane satisfying

this condition is called the *region of absolute stability* of the method. The desire for a Runge-Kutta method that is stable when (5.1) is motivates the following

Definition 5.1 (A-Stability) A Runge-Kutta method is *A-stable* iff the region of absolute stability $\mathcal{S}$ satisfies the condition

$$\{z \in \mathbb{C} \,|\, Re(z) \leq 0\} \subseteq \mathcal{S}.$$

From the characterization of the region of absolute stability above we have the following

Proposition 5.2 A Runge-Kutta method is A-stable iff $R(z)$ satisfies the condition $|R(z)| \leq 1$ for all z such that $Re(z) \leq 0$.

For an explicit Runge-Kutta method applied to (5.1) the Y_i in (4.1) can be computed directly in order of increasing i, starting with $Y_1 = U_n$. Then $Y_2 = (1 + a_{21}\lambda\Delta t)U_n$ and it is easy to see that in general $Y_i = p(\lambda\Delta t)U_n$, where $p(z)$ is a polynomial of degree less than or equal to $i-1$, since the formula (4.1) and the linearity of f ensure that each subsequent Y_i gets multiplied by a factor of $\lambda\Delta t$. It therefore follows from (4.2) that $U_{n+1} = q(\lambda\Delta t)U_n$, where $q(z)$ is a polynomial of degree less than or equal to s. Now, for any consistent method $\sum b_i = 1$, so the formula (4.2) together with the linearity of f ensures that each Y_i gets multiplied by a factor of $\lambda\Delta t$, so $q(z)$ is at least a linear function of z. Now, we observe that $R(z) = q(z)$, so this shows that $|R(z)| \to \infty$ as $z \to \infty$. Therefore, no explicit Runge-Kutta methods are A-stable.

Now, consider the one-stage theta method. Applied to the problem (5.1) it follows from (3.4) that

$$U_{n+1} = U_n + \Delta t\lambda\left((1-\theta)U_n + \theta U_{n+1}\right), \tag{5.3}$$

so that $U_{n+1}(1 - \lambda\Delta t\theta) = (1 + \lambda\Delta t(1-\theta))U_n$ and thus

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

If $\theta = 0$, then $R(z) = 1 + z$ and so the method is not A-stable, which we already knew since this corresponds to forward Euler, which is explicit. Suppose that $\theta \neq 0$. Then, letting $z = re^{i\phi}$, it follows that $R(z) = (e^{i\phi} + (1-\theta)r)/(e^{i\phi} - \theta r)$. Thus,

$$|R(z)| = \frac{(1-\theta)^2}{\theta^2} = \frac{1 - 2\theta + \theta^2}{\theta^2} = \theta^{-2} - 2\theta^{-1} + 1.$$

Thus, for $|R(z)| \leq 1$ we must have $\theta^{-2} - 2\theta^{-1} < 0$, which means that $-2\theta \leq -1$, or $\theta \geq \frac{1}{2}$. Thus, the one-stage theta method is A-stable provided that $\theta \in \left[\frac{1}{2}, 1\right]$. Considering the two-stage theta method, the linearity of the f in this case causes (3.5) to give rise to the same equation as (5.3). Therefore, the two-stage theta method is also A-stable provided that $\theta \in \left[\frac{1}{2}, 1\right]$.

We need the following

Lemma 5.3 Let $\mathcal{S}$ be the region of absolute stability of a consistent Runge-Kutta method. If $z \in \text{int}(\mathcal{S})$ then $|R(z)| < 1$.

Proof. (Adapted from Stuart & Humphries (1996)) Consider the expression for $R(z)$ in (5.2). We have that $(I - zA)^{-1} = \text{adj}(I - zA)/\det(I - zA)$. Since $\det(I - zA)$ is a polynomial in z and each entry in $\text{adj}(I - zA)$ is a polynomial in z, the entries of $(I - zA)^{-1}$ are rational functions of z. Thus, from (5.2), $R(z)$ is a rational function of z. Now, since by definition $|R(z)| \leq 1$ on $\mathcal{S}$, R cannot have a pole in $\mathcal{S}$, so R is analytic on $\mathcal{S}$. Thus, it follows from the maximum modulus principle that either $R(z)$ is constant or $|R(z)| < 1$ in $\text{int}(\mathcal{S})$. If R was constant and equal to 1 then it would follow from the expression for R in equation (5.2) that $b = 0$, but this contradicts the consistency of the method and so if $z \in \text{int}(\mathcal{S})$, $|R(z)| < 1$. □

Now, the following result establishes that A-stable methods succeed in replicating the asymptotic behaviour of the system (5.1).

Theorem 5.4 (Decay Preserving Runge-Kutta Methods) Any two solution sequences $\{U_n\}_{n=0}^{\infty}$, $\{V_n\}_{n=0}^{\infty}$ of an A-stable Runge-Kutta method applied to the problem (5.1) satisfy

$$\|U_{n+1} - V_{n+1}\| \leq \|U_n - V_n\| \tag{5.4}$$

for all $n \geq 0$, and if the inequality in (5.1) is strict then for all $\Delta t > 0$ and all $U \in \mathbb{C}$ then $\omega_{\Delta t}(U) = 0$.

Proof. We know that $\|U_{n+1} - V_{n+1}\| = \|R(\lambda\Delta t)(U_n - V_n)\| = |R(\lambda\Delta t)|\|U_n - V_n\|$. The condition that $Re(\lambda) \leq 0$ in (5.1) implies that $Re(\lambda\Delta t) \leq 0$ for all $\Delta t > 0$. Since the region of stability of A-stable methods includes the closure of the left half-plane, it follows from proposition 5.2 that $R(\lambda\Delta t) \leq 1$. The first part follows.
Now, we have that $\|U_{n+1}\| = |R(z)|\|U_n\|$ for the problem (5.1). Thus, for $U_0 = U \in \mathbb{C}$,

$$\|U_n\| = |R(z)|^n\|U\|. \tag{5.5}$$

If the inequality in (5.1) is strict, $Re(\lambda\Delta t) < 0$ for all $\Delta t > 0$. Thus, since $\mathcal{S}$ contains the closure of the left half-plane, $\lambda\Delta t$ is in the interior of $\mathcal{S}$. Thus, $|R(z)| < 1$ by lemma 5.3. Therefore, the result follows from (5.5). □

That A-stability is precisely the right condition to require in order for a Runge-Kutta method to stably discretize (5.1) follows from the following

Theorem 5.5 (Blow-up for Non-A-Stable Methods) For any non-A-stable Runge-Kutta method there exists a $z \in \mathbb{C}$ with $Re(z) < 0$ such that if $\lambda\Delta t = z$ then the numerical solution of (5.1) satisfies

$$\|U_n\| \to \infty \quad \text{as} \quad n \to \infty$$

for any initial condition $U \in \mathbb{C}$.

Proof. (Adapted from Stuart & Humphries (1996)) For the problem (5.1), given $U_0 = U \in \mathbb{C}$, $\|U_n\| = |R(z)|^n\|U\|$. For a non-A-stable Runge-Kutta method it follows from proposition 5.2 that there is a $z \in \mathbb{C}$ with $Re(z) \leq 0$ such that $|R(z)| > 1$. In the proof of lemma 5.3 we showed that $R(z)$ is a rational function of z, so $R(z)$ is continuous except at its poles. Thus, if z' such that $|R(z')| > 1$ has $Re(z') = 0$, there will exist a z in its neighbourhood with $Re(z) < 0$ such that $|R(z)| > 1$. Therefore, if for such a z we let $\lambda\Delta t = z$, then $|R(\lambda\Delta t)|^n \to \infty$ as $n \to \infty$. The result follows. □

Now, although we have shown that A-stability is the correct condition to require of a Runge-Kutta method in order for the method to correctly replicate the asymptotic behaviour of the system (5.1) for any time-step size Δt, this does not necessarily entail that non-A-stable methods are entirely without value. Indeed, they are not as the following result shows.

Theorem 5.6 (Conditional Decay of Runge-Kutta Methods) Let $\mathcal{S}$ be the region of stability of a consistent Runge-Kutta method. If $z = \lambda\Delta t \in \mathcal{S}$ then any two solution sequences $\{U_n\}_{n=0}^{\infty}$, $\{V_n\}_{n=0}^{\infty}$ of a Runge-Kutta method applied to the problem (5.1) satisfy (5.4), and if $z \in \text{int}(\mathcal{S})$ then $\omega_{\Delta t}(U) = 0$ for all $U \in \mathbb{C}$.

Proof. As we saw in the proof of theorem 5.4, $\|U_{n+1} - V_{n+1}\| = |R(\lambda\Delta t)|\|U_n - V_n\|$, so since $|R(z)| \leq 1$ inside the region of stability of the method the first part follows.
Lemma 5.3 ensures that for any $z \in \text{int}(\mathcal{S})$, $|R(z)| < 1$ for any consistent Runge-Kutta method. Since (5.5) applies for any Runge-Kutta method applied to the problem (5.1), provided that $\lambda\Delta t \in \text{int}(\mathcal{S})$, the second part follows. □

Thus, even non-A-stable methods can stably discretize the linear decay problem provided we can choose Δt such that $\lambda\Delta t \in \text{int}(\mathcal{S})$. Depending on how large $|\lambda|$ is this may not be a severe restriction, but for large $|\lambda|$ we would require an A-stable method.

This concludes our detailed consideration of the linear decay problem. This case was treated in detail because it is the simplest to analyze and it provides a conceptual basis for the treatment of more complex structural conditions on the vector field f. The treatment to follow will be less theoretical and more focused on the results available and on simulations to explore their consequences.

5.2 Dissipative Systems

A property satisfied by the vector field f of many discrete and continuous dynamical systems is that all bounded sets of initial data end up in some bounded set in phase space for sufficiently large t. Such a bounded set is called an *absorbing* set and a system that possesses an absorbing set is called *dissipative*. Part of the interest in this property is that the dynamics inside the absorbing set is not specified and so could include complicated invariant set, including chaotic attractors. At this point we are only concerned with the preservation of the absorbing set under discretization. We will consider what can be shown about the preservation of the attracting sets within it in the next section.

Humphries & Stuart (1994) proved a number of results for Runge-Kutta methods applied to dissipative problems (2.1) that satisfy the additional constraint that f is locally Lipschitz and that there are $\alpha, \beta \geq 0$ such that

$$\langle f(u), u\rangle \leq \alpha - \beta\|u\|^2 \tag{5.6}$$

for all $u \in \mathbb{R}^p$. They prove there that such a system is dissipative, with the open ball $B = B(0, \sqrt{\alpha/\beta} + \varepsilon)$ for any $\varepsilon > 0$ as an absorbing set, and that the system has a global attractor $\mathcal{A}$ given by $\mathcal{A} = \omega(B)$. We will now consider some of these results.

Given the importance of A-stable methods for the linear decay problem, the following result from

Stuart & Humphries (1996, 400) is of interest. For Runge-Kutta methods applied to any problem of the form (2.1),(5.6), they show that in order for the numerical solution to be dissipative for all $\Delta t > 0$, the method must be A-stable. It should be noted that dissipativity is understood in a generalized sense here to include the action of a generalized semigroup T^n of a generalized dynamical system.

Stating one of the main results from Humphries & Stuart (1994) requires a few additional definitions. For any Runge-Kutta method (4.1),(4.2) we can define two matrices,

$$B = \text{diag}(b_1, b_2, \ldots, b_s), \tag{5.7}$$

and

$$M = BA + A^T B - bb^T. \tag{5.8}$$

With these in hand we have the following

Definition 5.7 A Runge-Kutta method is called *algebraically stable* if B and M are both positive-semi definite.

An additional notion we need is that of a DJ-irreducible Runge-Kutta method, where 'DJ' is for 'Dahlquist-Jeltsch' who introduced the definition. It is defined negatively.

Definition 5.8 A Runge-Kutta method is called *DJ-reducible* if for some non-empty index set $T \subset \{1, \ldots, s\}$,

$$b_j = 0, j \in T, \qquad a_{ij} = 0, i \notin T, j \in T,$$

and is called *DJ-irreducible* otherwise.

We now have the following

Theorem 5.9 (Dissipative Algebraically Stable Runge-Kutta Methods, (Humphries & Stuart, 1994)) Suppose that (2.1),(5.6) is discretized using a DJ-irreducible, algebraically stable Runge-Kutta method. Then for any fixed step size $\Delta t > 0$ the map generated by the numerical method is dissipative (in the generalized sense) and the open ball $B(0, R)$ is an absorbing set for any $R > \sqrt{\alpha/\beta + \epsilon + \Delta t C(0, \Delta t)}$.[3]

The condition that the method be DJ-irreducible is not restrictive since, as they point out, in practice methods are usually DJ-irreducible. Moreover they prove in (Stuart & Humphries, 1996, 265) that every DJ-reducible method is reducible to a DJ-irreducible one. Thus, this result shows that the condition of algebraic stability is sufficient for the discretization by a Runge-Kutta method to preserve the dissipativity of the system (2.1),(5.6). The Runge-Kutta methods considered in this paper are all DJ-irreducible since $b_j \neq 0$ for all $j = 1, \ldots, s$, except for the two-stage theta method with θ equal to 0 or 1, but these cases are equivalent to the forward and backward Euler methods, which are DJ-irreducible.

[3]The expression for $C(\varepsilon, \Delta t)$ is quite complicated and it is not necessary to reproduce it here, but is provided in (Humphries & Stuart, 1994, 1466).

Consider the one-stage theta method. From the definition in section 3 $A = \theta$ and $b = 1$. $B = 1$, so it is positive definite. and since $M = \theta + \theta - 1 = 2\theta - 1$, for M to be positive semi-definite we require that $\theta \in \left[\frac{1}{2}, 1\right]$. Now consider the two-stage theta method. From the definition in section 3 and using (5.7), we have

$$A = \begin{pmatrix} 0 & 0 \\ 1-\theta & \theta \end{pmatrix}, \quad b = (1-\theta, \theta)^T, \quad B = \begin{pmatrix} 1-\theta & 0 \\ 0 & \theta \end{pmatrix}.$$

For $\theta \in [0, 1]$, B is positive semi-definite, so we consider M. From (5.8) we have that

$$\begin{aligned} M &= \begin{pmatrix} 1-\theta & 0 \\ 0 & \theta \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1-\theta & \theta \end{pmatrix} + \begin{pmatrix} 0 & 1-\theta \\ 0 & \theta \end{pmatrix} \begin{pmatrix} 1-\theta & 0 \\ 0 & \theta \end{pmatrix} - \begin{pmatrix} (1-\theta)^2 & \theta(1-\theta) \\ \theta(1-\theta) & \theta^2 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 \\ \theta(1-\theta) & \theta^2 \end{pmatrix} + \begin{pmatrix} 0 & \theta(1-\theta) \\ 0 & \theta^2 \end{pmatrix} - \begin{pmatrix} (1-\theta)^2 & \theta(1-\theta) \\ \theta(1-\theta) & \theta^2 \end{pmatrix} \\ &= \begin{pmatrix} -(1-\theta)^2 & 0 \\ 0 & \theta^2 \end{pmatrix}. \end{aligned} \tag{5.9}$$

For M to be positive semi-definite the diagonal terms must be non-zero, so this entails that $\theta = 1$. Therefore, the two-stage theta method is only algebraically stable if $\theta = 1$.

These results show that the backward Euler method and the implicit midpoint rule (1-stage theta, $\theta = \frac{1}{2}$) are algebraically stable, but that the forward Euler method and the trapezoidal rule (2-stage theta, $\theta = \frac{1}{2}$) are not. Of the methods considered in this paper this only leaves RK4. Since we know that RK4 is not A-stable, we expect that it is not algebraically stable, but it is not difficult to show that it is not using (5.8). From the definition is section 3 and from (5.7) and (5.8) we have that

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} \frac{1}{6} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & \frac{1}{6} \end{pmatrix} \quad bb^T = \begin{pmatrix} \frac{1}{36} & \frac{1}{18} & \frac{1}{18} & \frac{1}{36} \\ \frac{1}{18} & \frac{1}{9} & \frac{1}{9} & \frac{1}{18} \\ \frac{1}{18} & \frac{1}{9} & \frac{1}{9} & \frac{1}{18} \\ \frac{1}{36} & \frac{1}{18} & \frac{1}{18} & \frac{1}{36} \end{pmatrix}.$$

A bit of algebra then shows that

$$M = \begin{pmatrix} -\frac{1}{36} & \frac{1}{9} & -\frac{1}{18} & -\frac{1}{36} \\ \frac{1}{9} & -\frac{1}{9} & \frac{1}{18} & -\frac{1}{18} \\ -\frac{1}{18} & \frac{1}{18} & -\frac{1}{9} & -\frac{1}{9} \\ -\frac{1}{36} & -\frac{1}{18} & \frac{1}{9} & -\frac{1}{36} \end{pmatrix},$$

which is not positive semi-definite since it has negative diagonal entries. Therefore, RK4 is not algebraically stable.

Although we are primarily concerned with stability properties in this section, Humphries & Stuart (1994) also prove an important convergence result for dissipative systems. As well as showing that DJ-irreducible, algebraically stable Runge-Kutta method preserves dissipativity, they also prove the following

Theorem 5.10 (Convergence of Dissipative Algebraically Stable RK Methods(Humphries & Stuart, 1994)) Suppose that (2.1),(5.6) is approximated numerically using a DJ-irreducible, algebraically stable Runge-Kutta method. Then there exists $\Delta t_c > 0$ such that for $\Delta t < \Delta t_c$ the numerical solution possesses a global attractor $\mathcal{A}_{\Delta t}$ that satisfies[4]

$$\text{dist}(\mathcal{A}_{\Delta t}, \mathcal{A}) \to 0 \text{ as } \Delta t \to 0,$$

where $\mathcal{A}$ is the global attractor of (2.1),(5.6).

Thus, not only does the condition of algebraic stability ensure the preservation of the dissipativity of the dynamical system, it also preserves the ω-limit sets in the limited sense detailed by the theorem. It is limited for the following reason. This convergence condition is called *upper semicontinuity*, and it means that every point on the numerical attractor is close to a point on the true attractor. However, this does not in general entail the converse, *i.e.* the condition of *lower semicontinuity*:

$$\text{dist}(\mathcal{A}, \mathcal{A}_{\Delta t}) \to 0 \text{ as } \Delta t \to 0.$$

We require both upper and lower semi-continuity for true preservation of the dynamical attractor $\mathcal{A}$. Nevertheless, theorem 5.10 shows that algebraically stable Runge-Kutta methods do a very good job at approximating the attractor of a dissipative system.

To explore the results of theorems 5.9 and 5.10 let us consider the following test problem in $\mathbb{R}^2$:

$$\dot{r} = r - ar^3, \qquad \dot{\theta} = 1, \qquad a > 0. \tag{5.10}$$

For steady-state solutions $\dot{r} = 0$, which implies that $r(1 - ar^2) = 0$, *i.e.* $r = 0$ or $ar^2 - 1 = 0$. Thus, the system has an equilibrium point at $r = 0$ and a limit cycle solution $r = a^{-1/2}$. Since $f_r(r) = 1 - ar^2$, the equilibrium point at the origin is unstable. The stability of the limit cycle cannot be determined by a linear analysis but nonlinear stability analysis determines the stability by the sign of the coefficient $-a$ of r^3, which is negative, so the limit cycle is stable. Since the limit cycle is stable we have that for all initial points $U = (r, \theta)^T$ away from the origin, $\omega(U) = \{r \,|\, r = a^{-1/2}\}$. So given any bounded set E of initial data and any $\varepsilon > 0$, there is a $t^* > 0$ such that for all $t > t^*$, $S(t)E$ is inside the set $B_\varepsilon = \{(r, \theta) \,|\, r < a^{-1/2} + \epsilon\}$. Thus, for any $\varepsilon > 0$, B_ε is an absorbing set, and the system is dissipative. To prove that the system is dissipative in the sense required for theorem 5.9 we must show that the condition (5.6) holds. $f(u) = (r - ar^3, 1)^T$, with $u = (r, \theta)^T$, so $\langle f(u), u) \rangle = r^2 - ar^4 + \theta$ and $\|u\|^2 = r^2 + \theta^2$. Thus, we require that there are $\alpha, \beta \geq 0$ such that

$$r^2 - ar^4 + \theta \leq \alpha - \beta(r^2 + \theta^2).$$

Rearranging we obtain the equivalent condition

$$(1 + \beta)r^2 - ar^4 + \theta(1 + \beta\theta) \leq \alpha.$$

[4]The notion of set-distance used in this theorem is the asymmetric Hausdorff semi-distance, *viz.* for two sets A, B,

$$\text{dist}(A, B) = \sup_{a \in A} \text{dist}(a, A).$$

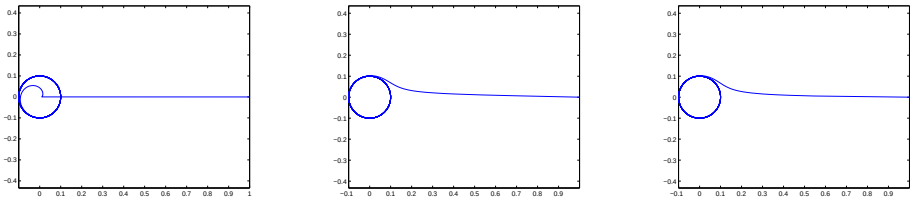
The left hand side splits into a function of r and a function of θ. The function of r is a quartic opening downwards and so is bounded above. The function of θ is a parabola opening upwards for $\beta \neq 0$, but since the system is confined to $\mathbb{R}^2$, there is an implicit periodic boundary condition on θ and θ only takes values in some finite set, *e.g.* $[0, 2\pi]$. Therefore the left hand side is bounded for any β so it is always possible to find an α to satisfy (5.6). Therefore, theorem 5.9 can apply to this system.

We now consider simulations of (5.10) in MATLAB. Consider the case where $a = 100$, so that the stable limit cycle has radius $r = 0.1$. The symmetry of the system (5.10) under θ-translations shows that the solutions are independent of θ so we need not worry about dynamical behaviour peculiar to our initial choice of θ. Thus, we only need to consider varying the initial value of r, and we may leave $\theta = 0$. Let us take $r = 1$. If we work with a very small time-step $\Delta t = 0.001$, all of the methods do quite well, both in the sense of getting the phase trajectory approximately right as well as the asymptotic behaviour.[5] Increasing to $\Delta t = 0.01$ the trajectory of the forward Euler method no longer matches the true trajectory, which is closely approximated by `ode45` (see figure 3(a) and compare to 3(f)). Increasing to $\Delta t = 0.1$, RK4, the trapezoidal rule (2-stage theta, $\theta = 0.5$) and the implicit midpoint rule (1-stage theta, $\theta = 0.5$) all fail to match the correct trajectory and both RK4 and forward Euler blow-up (see figure 4). It is seen from figure 4 that the implicit midpoint rule and the trapezoidal rule are beginning to show signs of near-term instability, in that they no longer closely approximate the correct trajectory. Nevertheless, we notice that the get the asymptotic behaviour right. Backward Euler, however, still approximates the true trajectory closely (see figure 4(b)). We see at this stage that all of the algebraically stable methods still get the asymptotic behaviour correct, as we expect, even if the global error is large. The only non-algebraically stable method still functioning at this point, the trapezoidal rule, also gets the asymptotic behaviour correct. Leaving aside forward Euler and RK4 since they are unstable, and increasing to $\Delta t = 1$, none of the fixed time-step methods approximates the phase trajectory accurately, although backward Euler still does quite well (see figure 5(a) and compare to 5(d)), and the trapezoidal rule has become quite unstable in terms of its phase trajectory (see figure 5(c)). But the algebraically stable methods and the trapezoidal rule still get the asymptotic behaviour correct (see figure 5). Of course this does not show that the trapezoidal rule gets the asymptotics correct for arbitrary inital data and Δt, but it seems likely that the increased stability of the trapezoidal rule over RK4 and forward Euler has to do with the implicitness of the method.

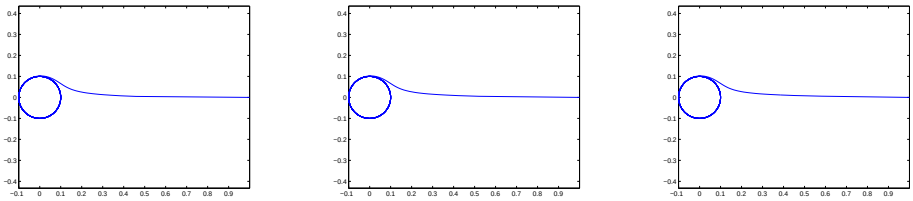
To explore the limits of the stability of the trapezoidal rule let us consider larger initial values of r. Increasing to $r(0) = 10$ and running the simulation for longer times, we see from figure 6(a) that by $t = 3000$ the phase trajectory still has not converged on the limit cycle set, but we see from figure 6(b) that it has by $t = 6000$. This is more clearly seen from figure 6(c), where it can be seen that the method takes valules approximately on the circle $r = 0.1$.[6] Thus, even for Δt as large as 10, the trapezoidal rule still reaches the limit cycle. Unfortunately the long computation time makes exploring much further quite difficult, but it can be seen by running the simulation for larger radii for long times that the radius of the orbit decreases very slowly, but steadily. This is of course not conclusive, but it is suggestive that a weaker condition than algebraic stability is

[5] Since all of the trajectories are visibly similar plots of phase trajectories have not been included.

[6] This is also a very good illustration of the difference between upper and lower semi-continuity, since if it were the case that the method replicated this 'star' pattern for all time then clearly each point of $\mathcal{A}_{\Delta t}$ would be close to $\mathcal{A}$, the limit cycle, but there would be many points on $\mathcal{A}$ quite far from any point of $\mathcal{A}_{\Delta t}$.

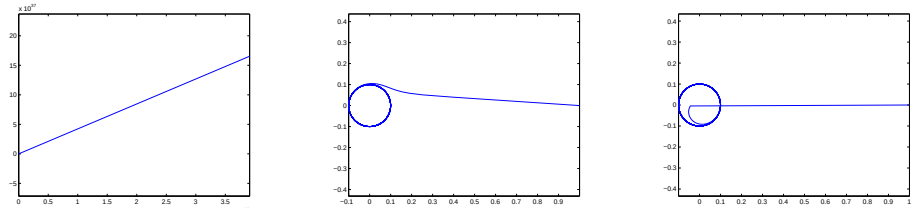


(a) `euler` (b) `beuler` (c) `theta1`, $\theta = 0.5$

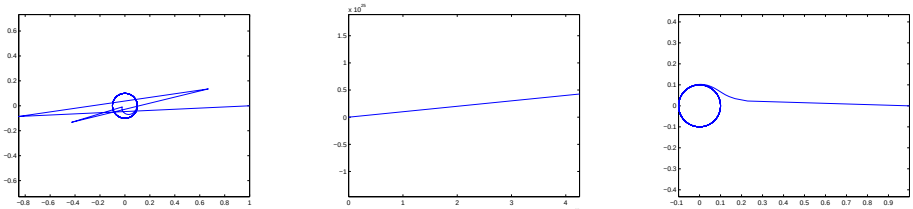


(d) `theta2`, $\theta = 0.5$ (e) `rk4` (f) `ode45`

Figure 3: Computation of the limit cycle test problem (5.10), with $u(0) = (1, 0)$, using different numerical methods. For each method, $t \in [0, 100]$ and $\Delta t = 0.01$ (except `ode45` is evaluated at 0.01 intervals).

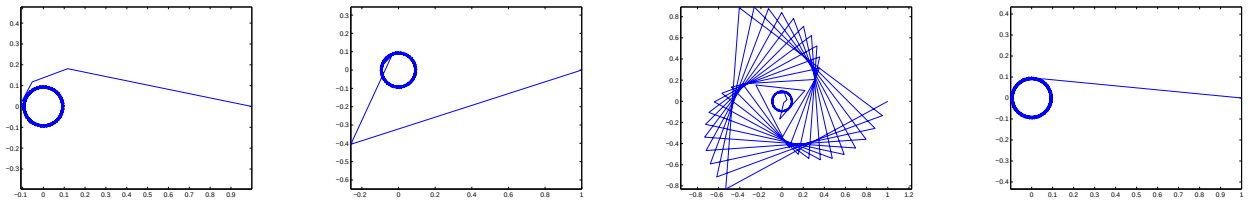


(a) `euler` (b) `beuler` (c) `theta1`, $\theta = 0.5$



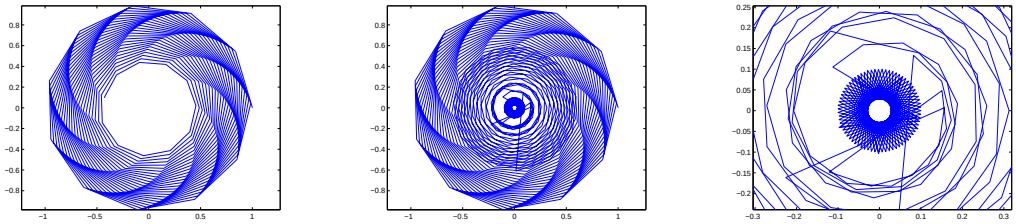
(d) `theta2`, $\theta = 0.5$ (e) `rk4` (f) `ode45`

Figure 4: Computation of the limit cycle test problem (5.10), with $u(0) = (1, 0)$, using different numerical methods. For each method, $t \in [0, 100]$ and $\Delta t = 0.1$ (except `ode45` is evaluated at 0.1 intervals and for `euler` $t \in [0, 0.4]$).



(a) `beuler` (b) `theta1`, $\theta = 0.5$ (c) `theta2`, $\theta = 0.5$ (d) `ode45`

Figure 5: Computation of the limit cycle test problem (5.10), with $u(0) = (1, 0)$, using different numerical methods. For each method, $t \in [0, 100]$ and $\Delta t = 1$ (except `ode45` is evaluated at 0.1 intervals).



(a) $t \in (0, 3000)$ (b) $t \in (0, 6000)$ (c) zoom in of 6(b)

Figure 6: Computation of the limit cycle test problem (5.10), with $u(0) = (1, 0)$, using the trapezoidal rule (`theta2`, $\theta = 0.5$) with $\Delta t = 10$.

sufficient to preserve dissipativity, perhaps with some restriction on the time-step. This is question appears only to be of theoretical interest, however, since the algebraically stable methods clearly perform better than the trapezoidal rule.

6 Discretization of Arbitrary Attractors

We now turn to a very brief consideration of the issue of the discretization of arbitrary attractors. Attractors play an important role in determining the overall dynamics of a dynamical system (2.1), so gaining insight into their structure is important. But for nonlinear vector fields f, even if they are only moderately complex, obtaining analytical results about the attractors is an extremely difficult matter. Thus, numerical methods are a key part of the study of dynamical systems. It is therefore an extremely important question what effect the discretization has on the attracting sets of the dynamical system.

A basic result concerning this question was proved by Kloeden & Lorenz (1986) which says that, under a fairly strong global Lipschitz condition on f, if the dynamical system (2.1) possesses a compact, uniformly asymptotically stable set Λ then there is a $\Delta t_c > 0$ such that for $\Delta t \in (0, \Delta t_c)$ a one-step numerical method has a compact, uniformly asymptotically stable absorbing set $\Lambda(\Delta t) \supseteq \Lambda$ that converges to Λ as $\Delta t \to 0$. Thus, this establishes that for an attractor $\mathcal{A} = \omega(\Lambda)$ of the dynamical system, the numerical method has attractors $\mathcal{A}_{\Delta t} = \omega(\Lambda(\Delta t))$ that converge to $\mathcal{A}$ as $\Delta t \to 0$, where the convergence is upper semi-continuous. Although this is a nice result, it requires a quite strong smoothness assumption on the vector field f and it falls short of the ideal of Hausdorff convergence,

i.e. convergence that is both upper and lower semi-continuous.

Much time has passed since this paper was written, and there are a number of ways in which better results can be obtained. Grüne (2003) describes the three main approaches that have been taken to this problem. One approach is to impose certain conditions on a general numerical method such that Hausdorff convergence can be proved. Stuart & Humphries (1996, 555) prove such a result for attractors that satisfy a particular structural condition that includes gradient systems with a finite number of equilibria. They also prove more generally that, under suitable conditions on the numerical method, if an attracting set $\mathcal{A}_{\Delta t}$ is invariant for all Δt and is Hausdorff convergent to a compact set $\mathcal{A}_0$ than that set is an invariant set of the dynamical system. A second approach is to devise specific algorithms for which it is possible to prove convergence to the correct sets under mild or no restrictions on the discretized system. Such results include the ability to compute an approximation of a global attractor of an evolution semigroup. Dellnitz & Hohmann (1997) have developed such a method. The third approach is to formulate "conditions on the behavior of the numerical systems under which [one] can ensure convergence of the respective sets or the existence of respective nearby sets for the approximated system." (Grüne, 2003, 2097) This is the approach Grüne follows in the cited paper, where it is proved, *e.g.*, that a sequence of numerical attractors converges to a real attractor if and only if the numerical attractors are attracting with uniformly bounded attraction rates.

Thus, there are number of promising approaches to the problem of the discretization of the attractors of a dynamical system. It is unclear how general a characterization of the numerical methods that preserve invariant sets it will be possible to obtain. And aside from the question of discretization error for finite dimensional systems (2.1), there is the more difficult question of the preservation of invariant sets for infinite dimensional systems, including numerical methods for the computation of delay differential equations and partial differential equations. Nevertheless, it is clear distinct progress is being made in this interesting and important field.

7 Conclusion

The main, but very limited, conclusion that can be drawn from this project is that numerical simulations provide a useful complement to an understanding of results in numerical analysis. On the one hand it is useful to illustrate the results of theorems by seeing how they apply to particular numerical methods and with large variations of the time-step. And on the other hand it is interesting to explore beyond what is established by theorems by running simulations under conditions or with methods to which the theorems do not apply. The question raised concerning the limitation of the result concerning algebraically stable Runge-Kutta methods applied to dissipative systems, not of any significance in itself, points to the potential use of numerical simulation in the search for new theorems that can be proved. This is becomes of increasing interest as the computational power available on personal computers continues to grow in approximate accordance with Moore's law.

A Numerical Methods (in alphabetical order)

A.1 beuler

```
function [tout,yout] = beuler(F, DF, tspan, y0, h, tol, Nmax)

%BEULER - Fixed time step backward Euler method for solving single or
%   systems of first-order ODE;  Newton's method is used to solve the
%   implicit equation which arises at each time step
%
%   BEULER(F,DF,TSPAN,Y0,H,TOL,NMAX) with TSPAN = [T0 TFINAL] integrates
%   the system of differential equations dy/dt = f(t,y) from t = T0 to
%   t = TFINAL. The initial condition is y(T0) = Y0.
%
%   The first argument, F, is a function handle or an anonymous function
%   that defines f(t,y).  This function must have two input arguments,
%   t and y, and must return a column vector of the derivatives, dy/dt.
%
%   The second argument, DF, is a function handle or an anonymous function
%   that defines the Jacobian Df(t,y) of f(t,y).  This function must have
%   two input arguments, t and y, and must return a matrix with the
%   derivatives, (df_i/dy_j)(t), as elements.
%
%   The fifth argument, H, is the fixed time step of the method
%
%   The sixth argument, TOL, is convergence tolerance applied to Newton's
%   method at each time step
%
%   The last argument, NMAX, is the maximum number of iterations of
%   Newton's method to be performed at each time step
%
%   With two output arguments, [T,Y] = BEULER(...) returns a column
%   vector T and an array Y where Y(:,k) is the solution at T(k).
%
%   With no output arguments, BEULER plots the emerging solution.
%
%   Dependencies:
%
%   when applied to a system of equations, this routine makes use of
%   MATLAB's backslash operator to solve a linear system

plotfun = @odeplot;
t0 = tspan(1);
tfinal = tspan(2);
tdir = sign(tfinal - t0);
```

```
h=tdir*abs(h);
plotit = (nargout == 0)
t = t0;
y = y0(:);

% Initialize output.

if plotit
   plotfun(tspan,y,'init');
else
   tout = t;
   yout = y.';
end

% number of equations determines the method used

neqn = length(y0)

if (neqn == 1)
    while t ~= tfinal

       % Stretch the step if t is close to tfinal.

       if abs(h) >= abs(tfinal - t)
          h = tfinal - t;
       end

       % Compute a step

       x = y;
   for j = 1:Nmax
           top = (x - y) - h * F(t+h, x);
   bot = 1 - h * DF(t+h, x);
   dx = top / bot;
   x = x - dx;
   if (abs(dx) < tol)
               break
           end
       end

       % Take a step.

   y = x;
   t = t + h;
```

```
       % Update output

        if plotit
          if plotfun(t,y,'');
             break
          end
        else
          tout(end+1,1) = t;
          yout(end+1,:) = y.';
        end
    end
else
    while t ~= tfinal

       % Stretch the step if t is close to tfinal.

       if abs(h) >= abs(tfinal - t)
          h = tfinal - t;
       end

       % Compute a step

       x = y;
       %w0 = y0;
   for j = 1:Nmax
            Fx = (x - y) - h * F(t+h, x);
   DFx = eye(neqn) - h * DF(t+h, x);
            dx = -DFx\Fx;
   x = x + dx;
   if (max(abs(dx)) < tol)
               break
            end
       end

       % Take a step

   y = x;
   t = t + h;

       % Update output

        if plotit
          if plotfun(t,y,'');
             break
          end
```

```
        else
          tout(end+1,1) = t;
          yout(end+1,:) = y.';
        end
    end
end

if plotit
   plotfun([],[],'done');
end
```

A.2 euler

```
function [tout,yout] = euler(F, tspan, y0, h);
% EULER - Fixed step euler method for solving systems of ODE
%
%   EULER(F,TSPAN,Y0,H) with TSPAN = [T0 TFINAL] integrates the system
%   of differential equations dy/dt = f(t,y) from t = T0 to t = TFINAL.
%   The initial condition is y(T0) = Y0.
%
%   The first argument, F, is a function handle or an anonymous function
%   that defines f(t,y).  This function must have two input arguments,
%   t and y, and must return a column vector of the derivatives, dy/dt.
%
%   The last argument, H, is the fixed time step of the method
%
%   With two output arguments, [T,Y] = EULER(...) returns a column
%   vector T and an array Y where Y(:,k) is the solution at T(k).
%
%   With no output arguments, EULER plots the emerging solution.

plotfun = @odeplot;
t0 = tspan(1);
tfinal = tspan(2);
tdir = sign(tfinal - t0);
h=tdir*abs(h);
plotit = (nargout == 0)
t = t0;
y = y0(:);

% Initialize output.

if plotit
   plotfun(tspan,y,'init');
else
```

```
   tout = t;
   yout = y.';
end

% The main loop.

while t ~= tfinal

   % Stretch the step if t is close to tfinal.

   if abs(h) >= abs(tfinal - t)
      h = tdir*(tfinal - t);
   end

   % Take a step.

   y = y + h*F(t, y);
   t = t + h;

   % Update output

   if plotit
      if plotfun(t,y,'');
         break
      end
   else
      tout(end+1,1) = t;
      yout(end+1,:) = y.';
   end

end

if plotit
   plotfun([],[],'done');
end
```

A.3 rk4

```
function [tout,yout] = rk4(F, tspan, y0,h)
%RK4  Solve non-stiff differential equations.
%
%   RK3(F,TSPAN,Y0,H) with TSPAN = [T0 TFINAL] integrates the system
%   of differential equations dy/dt = f(t,y) from t = T0 to t = TFINAL.
%   The initial condition is y(T0) = Y0.
%
```

```
%   The first argument, F, is a function handle or an anonymous function
%   that defines f(t,y).  This function must have two input arguments,
%   t and y, and must return a column vector of the derivatives, dy/dt.
%
%   The last argument, H, is the fixed time step of the method
%
%   With two output arguments, [T,Y] = RK4(...) returns a column
%   vector T and an array Y where Y(:,k) is the solution at T(k).
%
%   With no output arguments, RK4 plots the emerging solution.

% Initialize variables.

plotfun = @odeplot;
t0 = tspan(1);
tfinal = tspan(2);
tdir = sign(tfinal - t0);
plotit = (nargout == 0);
t = t0;
y = y0(:);

% Initialize output.

if plotit
   plotfun(tspan,y,'init');
else
   tout = t;
   yout = y.';
end

% The main loop.

while t ~= tfinal

   % Stretch the step if t is close to tfinal.

   if abs(h) >= abs(tfinal - t)
      h = tfinal - t;
   end

   % Take a step.

   s1 = F(t, y);
   s2 = F(t+h/2, y+h/2*s1);
   s3 = F(t+h/2, y+h/2*s2);
```

```
   s4 = F(t+h, y+h*s3);
   t = t + h;
   y = y + h*(s1 + 2*s2 + 2*s3 + s4)/6;

   % Update output

   if plotit
     if plotfun(t,y,'');
        break
     end
   else
     tout(end+1,1) = t;
     yout(end+1,:) = y.';
   end

end

if plotit
   plotfun([],[],'done');
end
```

A.4 theta1

```
function [tout,yout] = theta1(F, DF, tspan, y0, h, theta, tol, Nmax)

%THETA2 - Fixed time one stage theta method for solving single or
%   systems of first-order ODE;  Newton's method is used to solve the
%   implicit equation which arises at each time step - For theta = 0 the
%   method is equivalent to forward Euler and for theta = 1 the method is
%   equivalent to backward Euler
%
%   THETA2(F,DF,TSPAN,Y0,H,TOL,NMAX) with TSPAN = [T0 TFINAL] integrates
%   the system of differential equations dy/dt = f(t,y) from t = T0 to
%   t = TFINAL. The initial condition is y(T0) = Y0.
%
%   The first argument, F, is a function handle or an anonymous function
%   that defines f(t,y).  This function must have two input arguments,
%   t and y, and must return a column vector of the derivatives, dy/dt.
%
%   The second argument, DF, is a function handle or an anonymous function
%   that defines the Jacobian Df(t,y) of f(t,y).  This function must have
%   two input arguments, t and y, and must return a matrix with the
%   derivatives, (df_i/dy_j)(t), as elements.
%
%   The fifth argument, H, is the fixed time step of the method
```

```
%
%   The sixth argument, TOL, is convergence tolerance applied to Newton's
%   method at each time step
%
%   The last argument, NMAX, is the maximum number of iterations of
%   Newton's method to be performed at each time step
%
%   With two output arguments, [T,Y] = THETA1(...) returns a column
%   vector T and an array Y where Y(:,k) is the solution at T(k).
%
%   With no output arguments, THETA1 plots the emerging solution.
%
%   Dependencies:
%
%   when applied to a system of equations, this routine makes use of
%   MATLAB's backslash operator to solve a linear system

plotfun = @odeplot;
t0 = tspan(1);
tfinal = tspan(2);
tdir = sign(tfinal - t0);
h=tdir*abs(h);
plotit = (nargout == 0)
t = t0;
y = y0(:);

% Initialize output.

if plotit
   plotfun(tspan,y,'init');
else
   tout = t;
   yout = y.';
end

% number of equations determines the method used

neqn = length(y0)

if (neqn == 1)
    while t ~= tfinal

       % Stretch the step if t is close to tfinal.

       if abs(h) >= abs(tfinal - t)
```

```
          h = tfinal - t;
       end

       % Compute a step

       x = y;
   for j = 1:Nmax
        top = (x - y) - h * F(t+h, (1 - theta)*y + theta*x);
bot = 1 - h * theta * DF(t+h, (1 - theta)*y + theta*x);
dx = top / bot;
x = x - dx;
if (abs(dx) < tol)
           break
        end
       end

       % Take a step.

   y = x;
   t = t + h;

       % Update output

        if plotit
          if plotfun(t,y,'');
             break
          end
        else
          tout(end+1,1) = t;
          yout(end+1,:) = y.';
        end
    end
else
    while t ~= tfinal

       % Stretch the step if t is close to tfinal.

       if abs(h) >= abs(tfinal - t)
          h = tfinal - t;
       end

       % Compute a step

       x = y;
       %w0 = y0;
```

```
   for j = 1:Nmax
           Fx = (x - y) - h * F(t+h, (1 - theta)*y + theta*x);
   DFx = eye(neqn)-h*theta*DF(t+h, (1 - theta)*y + theta*x);
           dx = -DFx\Fx;
   x = x + dx;
   if (max(abs(dx)) < tol)
               break
           end
       end

       % Take a step

   y = x;
   t = t + h;

       % Update output

        if plotit
          if plotfun(t,y,'');
             break
          end
        else
          tout(end+1,1) = t;
          yout(end+1,:) = y.';
        end
    end
end

if plotit
   plotfun([],[],'done');
end
```

A.5 theta2

```
function [tout,yout] = theta2(F, DF, tspan, y0, h, theta, tol, Nmax)

%THETA2 - Fixed time two stage theta method for solving single or
%   systems of first-order ODE;  Newton's method is used to solve the
%   implicit equation which arises at each time step - For theta = 0 the
%   method is equivalent to forward Euler and for theta = 1 the method is
%   equivalent to backward Euler
%
%   THETA2(F,DF,TSPAN,Y0,H,TOL,NMAX) with TSPAN = [T0 TFINAL] integrates
%   the system of differential equations dy/dt = f(t,y) from t = T0 to
%   t = TFINAL. The initial condition is y(T0) = Y0.
```

```
%
%   The first argument, F, is a function handle or an anonymous function
%   that defines f(t,y).  This function must have two input arguments,
%   t and y, and must return a column vector of the derivatives, dy/dt.
%
%   The second argument, DF, is a function handle or an anonymous function
%   that defines the Jacobian Df(t,y) of f(t,y).  This function must have
%   two input arguments, t and y, and must return a matrix with the
%   derivatives, (df_i/dy_j)(t), as elements.
%
%   The fifth argument, H, is the fixed time step of the method
%
%   The sixth argument, TOL, is convergence tolerance applied to Newton's
%   method at each time step
%
%   The last argument, NMAX, is the maximum number of iterations of
%   Newton's method to be performed at each time step
%
%   With two output arguments, [T,Y] = THETA2(...) returns a column
%   vector T and an array Y where Y(:,k) is the solution at T(k).
%
%   With no output arguments, THETA2 plots the emerging solution.
%
%   Dependencies:
%
%   when applied to a system of equations, this routine makes use of
%   MATLAB's backslash operator to solve a linear system

plotfun = @odeplot;
t0 = tspan(1);
tfinal = tspan(2);
tdir = sign(tfinal - t0);
h=tdir*abs(h);
plotit = (nargout == 0)
t = t0;
y = y0(:);

% Initialize output.

if plotit
   plotfun(tspan,y,'init');
else
   tout = t;
   yout = y.';
end
```

```
% number of equations determines the method used

neqn = length(y0)

if (neqn == 1)
    while t ~= tfinal

       % Stretch the step if t is close to tfinal.

       if abs(h) >= abs(tfinal - t)
          h = tfinal - t;
       end

       % Compute a step

       x = y;
   for j = 1:Nmax
          top = (x - y) - h * ((1 - theta)*F(t, y) + theta*F(t+h, x));
   bot = 1 - h * theta * DF(t+h, x);
   dx = top / bot;
   x = x - dx;
   if (abs(dx) < tol)
              break
          end
       end

       % Take a step.

   y = x;
   t = t + h;

       % Update output

        if plotit
          if plotfun(t,y,'');
             break
          end
        else
          tout(end+1,1) = t;
          yout(end+1,:) = y.';
        end
    end
else
    while t ~= tfinal
```

```
      % Stretch the step if t is close to tfinal.

      if abs(h) >= abs(tfinal - t)
         h = tfinal - t;
      end

      % Compute a step

      x = y;
      %w0 = y0;
   for j = 1:Nmax
         Fx = (x - y) - h * ((1 - theta)*F(t, y) + theta*F(t+h, x));
   DFx = eye(neqn) - h * theta * DF(t+h, x);
         dx = -DFx\Fx;
   x = x + dx;
   if (max(abs(dx)) < tol)
             break
         end
      end

      % Take a step

   y = x;
   t = t + h;

      % Update output

       if plotit
         if plotfun(t,y,'');
            break
         end
       else
         tout(end+1,1) = t;
         yout(end+1,:) = y.';
       end
    end
end

if plotit
   plotfun([],[],'done');
end
```

References

Dellnitz, Michael, & Hohmann, Andreas. (1997). A Subdivision Algorithm for the Computation of Unstable Manifolds and Global Attractors. *Numerical Mathematics*, **75**, 293–317.

Grüne, Lars. (2003). Attraction Rates, Robustness, and Discretization of Attractors. *SIAM Journal on Numerical Analysis*, **41**(6), 2096–2113.

Humphries, A. R., & Stuart, A. M. (1994). Runge-Kutta Methods for Dissipative and Gradient Dynamical Systems. *SIAM Journal on Numerical Analysis*, **31**(5), 1452–1485.

Kloeden, P. E., & Lorenz, J. (1986). Stable Attracting Sets in Dynamical Systems and in their One-Step Discretizations. *SIAM Journal on Numerical Analysis*, **23**(5), 986–995.

Stuart, A. M., & Humphries, A. R. (1996). *Dynamical Systems and Numerical Analysis*. Cambridge University Press.