Heidelberg University

Department of Physics and Astronomy

Seminar report

Classical Numerical Approaches for Solving (Stiff) Differential Equations

Leonard Storcks and Daniel Richter

Seminar by Dr. Tobias Buck

AstroAI-Lab

August 30, 2023

Abstract

This report explores various numerical methods for solving differential equations, focusing on the challenges of computationally preserving physically conserved quantities and addressing the concept of stiffness. Symplectic integrators, like the leapfrog method, are shown to be advantageous for Hamiltonian systems, mitigating long-term energy drift. Implicit integrators and techniques such as automatic differentiation and graph coloring are presented for efficiently solving stiff differential equations.

Contents

1 Introduction **1**

2 A Brief Seminar Report **2**
2.1 Some Classical Methods for Solving ODEs Numerically 2
2.2 The Problem of Conserved Quantities 3
2.3 The Problem of Stiffness 5
2.4 Efficiently Solving Stiff ODEs 6

3 Some Classical Methods for Solving ODEs **8**
3.1 Explicit Euler Method 8
3.2 Going higher order: Runge-Kutta methods 10

4 The Problem of Conserved Quantities **13**
4.1 Runge-Kutta methods are unfavorable for simulating Hamiltonian systems . 13
4.2 Symplectic integrators to the help 15

5 The Problem of Stiffness **18**
5.1 Introducing stiffness at the hand of a simple example 18
5.2 A *definition* of stiffness 20

6 Efficiently Solving Stiff ODEs **21**
6.1 Implicit Euler to the help 21
6.2 Solving larger systems is difficult 25
6.3 Outlook on more intricate methods and refinements 37

7 Breakdown of Classical Methods and Outlook **38**

References **40**

1 Introduction

Numerical methods for solving differential equations and the rise of computers have allowed for accurate modeling of complex dynamical systems that could hardly be approached by analytical means even under the usage of perturbation theory (compare Moser, 1978).

The application of numerical methods, however, is not without caveats. One can quickly run into large temporal drifts in quantities that should physically be conserved (as illustrated in Hairer, Wanner, and Lubich, 2006, chapter 1) or end up having to use excessively small steps in the numerical scheme compared to the smoothness of the true solution (*stiffness*) (Lambert, 1991, chapter 6).

After a short introduction to basic numerical methods, we will discuss the problem of conserved quantities and the problem of stiffness. For the simulation of Hamiltonian systems, for which we want our numerical methods to have good conservation properties, symplectic integrators like the leapfrog method are favorable (Hairer, Wanner, and Lubich, 2006, chapters 1, 6, 10 – 12). Efficiently solving stiff differential equations will then be the central aspect covered. Here, implicit integrators (Press et al., 2007, chapter 17.5) and efficient methods like automatic differentiation and graph coloring will be applied (Rackauckas, Sciemon, et al., 2022; Gebremedhin et al., 2005).

While numerically solving differential equations is an old and well researched field[1] simulations containing strong turbulence or high-dimensional partial differential equations are highly problematic and may require new approaches incorporating insights from machine learning (Han et al., 2018; Kochkov et al., 2021). The breakdown of classical methods is discussed in the outlook.

As we felt that a short report could hardly encompass easily approachable and sufficiently broad coverage of the topic, we decided to extend it by adding more in-depth chapters. Implementations of the methods discussed can be found under is.gd/stiff.

[1] The well known Verlet-Method for instance was already used in Newtons *Principia* from 1687.

2 A Brief Seminar Report

2.1 Some Classical Methods for Solving ODEs Numerically

Explicit Euler and Runge Kutta methods

The simplest method for solving an ordinary differential equation (ODE) $\partial_t \underline{y} = \underline{f}\left(\underline{y}\right)$ with initial values $\underline{y}(t = t_0) = \underline{y}_0$ numerically, is the Explicit Euler Method which is of first order accuracy (so cutting the time-step in half will *cut in half* the error accumulated over a certain timespan, shown in section 3.1).

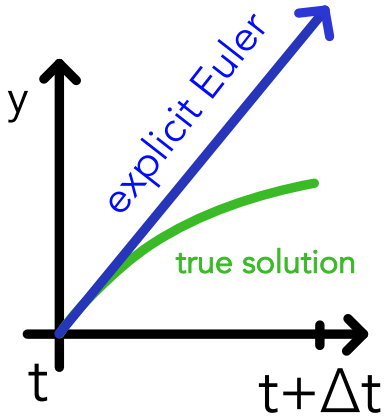


Figure 1: Explicit Euler illustration

$$\underline{y}^{(n+1)} = \underline{y}^{(n)} + \underline{f}\left(\underline{y}^{(n)}\right)\Delta t,$$
$$\text{where} \quad \underline{y}^{(0)} = \underline{y}_0$$

We can, for instance, go to higher order accuracies than Explicit Euler by using specific weighted combinations of first order derivatives. These methods are called *Runge-Kutta methods* (for a general derivation see Springel et al., 2023, chapter 2.4) with the simplest of

them being the second order *Runge-Kutta 2* (RK2) method.

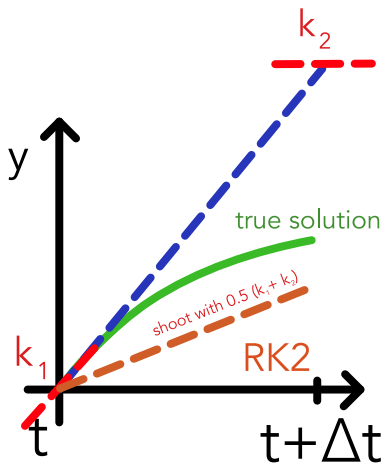


Figure 2: Runge-Kutta 2 illustration

$$k_1 = f\left(y^{(n)}, t_n\right)$$
$$k_2 = f\left(y^{(n)} + \Delta t k_1, t_n + \Delta t\right)$$
$$y^{(n+1)} = y^{(n)} + \frac{\Delta t}{2}\left(k_1 + k_2\right) + \mathcal{O}\left(\Delta t^3\right)$$

Depending on the step-size Explicit Euler as well as (explicit) RK methods can show divergent behavior for problems with well-behaved analytical solutions (see e.g. in section 3.1; both have finite regions of absolute stability; for Explicit Euler see figure 9), details on stability analysis are presented in Hairer and Wanner, 1996, chapter IV.3.

2.2 The Problem of Conserved Quantities

Runge-Kutta methods do not conserve energy and are not symplectic

Consider the two-body-problem reduced to a dimensionless one-body-problem as described by:

$$\frac{d\underline{s}}{d\tau} = \underline{w}, \quad \frac{d\underline{w}}{d\tau} = -\frac{\underline{s}}{s^3}$$

where $\underline{s}$ is the position and $\underline{w}$ the velocity. The numerical solution obtained using RK2 can be seen in figure 3. It is evident that energy, angular momentum and the Runge-Lenz vector are not conserved numerically, the numerical orbit deviates from the true orbit.

By using symplectic integrators

- symplecticity, most intuitively understood as preservation of area in the phase-space flow of Hamiltonian systems (Hairer, Wanner, and Lubich, 2006, sections 6.2, 6.3) (see also figure 14), is maintained
- long-term energy drift can be mitigated (the general idea is that geometric properties of the integrator translate into structure preservation on the level of modified equations) (Hairer, Wanner, and Lubich, 2006, chapters X - XII)

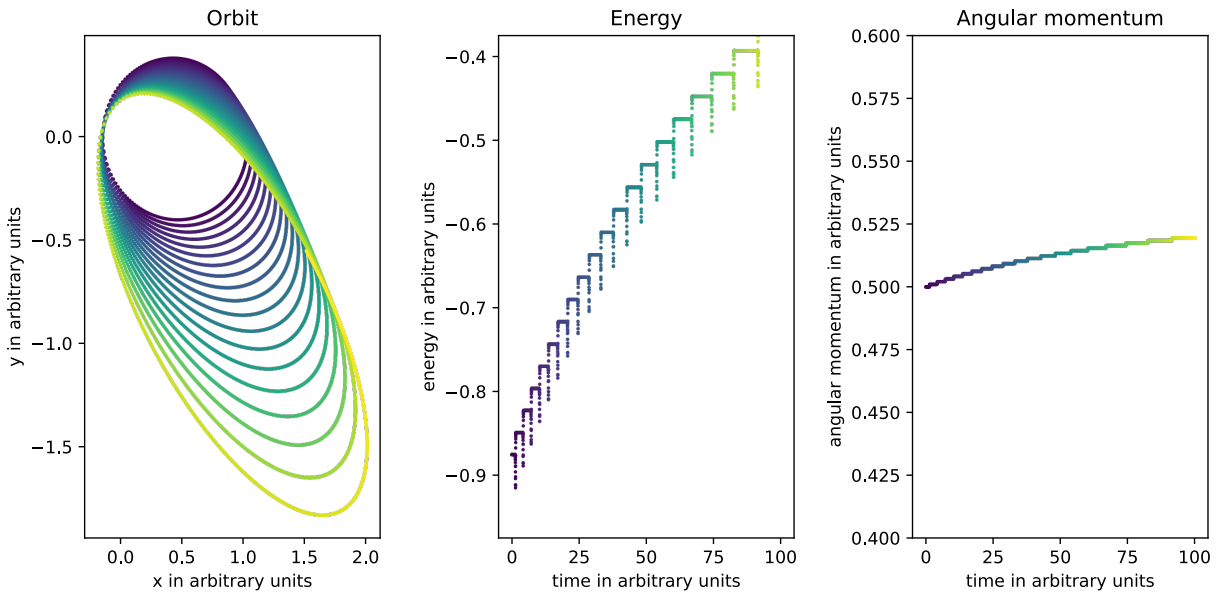
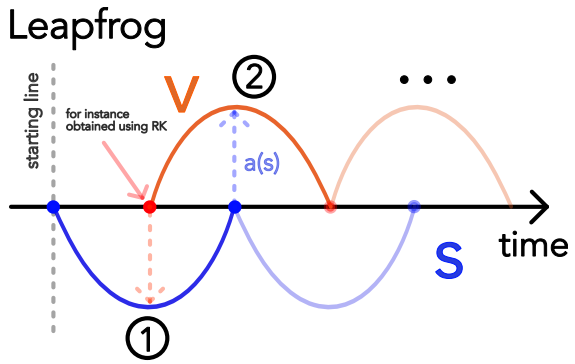


Figure 3: Numerical solution of the two-body-problem using RK2. The left panel shows the orbit, the central one the energy and the right one the angular momentum. Time is also encoded in the form of color, going from a dark blue to yellow.

- angular momentum is usually exactly conserved (compare Hairer, Lubich, et al., 2003, theorem 3.5 for the leapfrog method)

The Leapfrog Method

Let us illustrate the effect of using a symplectic method - the leapfrog method - on long-term energy conservation. Consider a classical physical problem with a particle at position $\underline{s}$ with velocity $\underline{v}$ and an acceleration $\underline{a}$ that depends only on the position. The leapfrog method can then be described by:



$$\underline{s}(t+\frac{1}{2}\Delta t) = \underline{s}(t-\frac{1}{2}\Delta t) + \underline{v}(t)\Delta t$$
$$\underline{v}(t+\Delta t) = \underline{v}(t) + \underline{a}(t+\frac{1}{2}\Delta t)\Delta t$$

Figure 4: Illustration of the leapfrog scheme

This integration method can be applied to the same orbital problem. From the results, depicted in figure 5, angular momentum conservation and good energy conservation properties (no overall drift) can be observed.

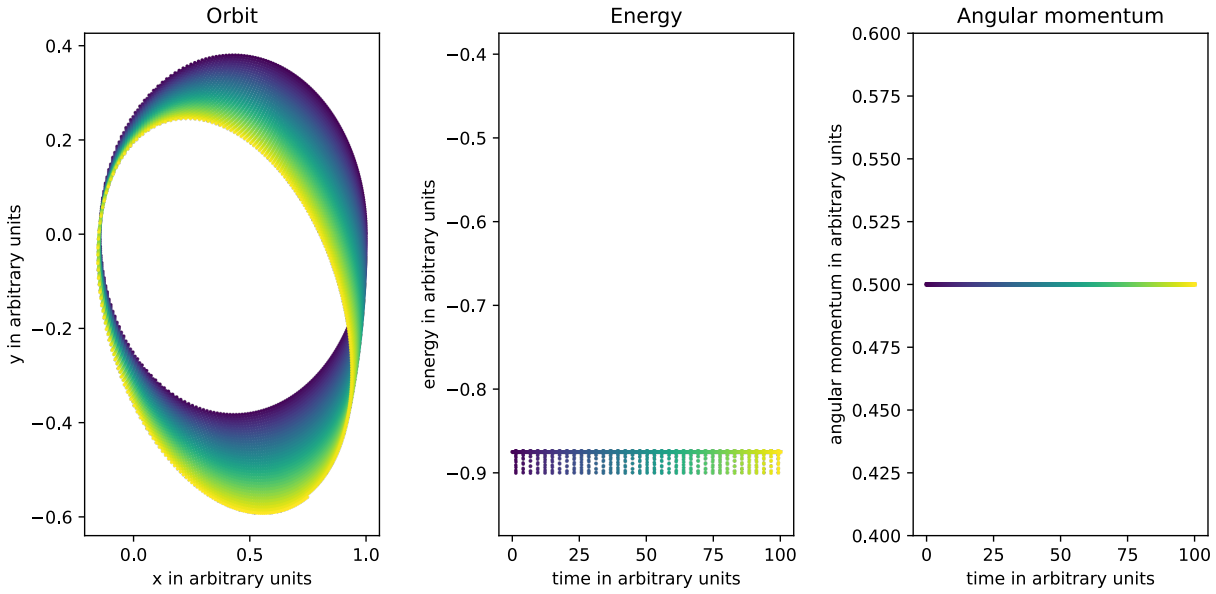


Figure 5: Numerical solution of the two-body-problem using the leapfrog method in the kick-drift-kick scheme (see section 4.2). The left panel shows the orbit, the central one the energy and the right one the angular momentum. Time is also encoded in the form of color, going from a dark blue to yellow.

2.3 The Problem of Stiffness

When using an explicit method like Explicit Euler, we often run into a situation where to guarantee numerical stability, we have to resolve the smallest timescale of the problem at hand, which is usually infeasible from an efficiency standpoint (see section 5.1 for a simple example).

More specifically, as Lambert, 1991 shows, a hard mathematical definition of stiffness is difficult and we therefore resort to the broad practical definition (Lambert, 1991, chapter 6)

> »If a numerical method with a finite region of absolute stability, applied to a system with any initial conditions, is forced to use in a certain interval of integration a step length which is excessively small in relation to the smoothness of the exact solution in that interval, then the system is said to be stiff in that interval.«

Applied examples can be found in chapters 5 and 6.

2.4 Efficiently Solving Stiff ODEs

Based on the stability advantages of implicit methods (see section 6.1) we can avoid those *excessively small time-steps*. This, however, comes with the caveat that implicit steps usually involve solving a root-finding problem which is computationally expensive. Therefore, we have to use smart approaches detailed in chapter 6 and illustrated in figure 6.

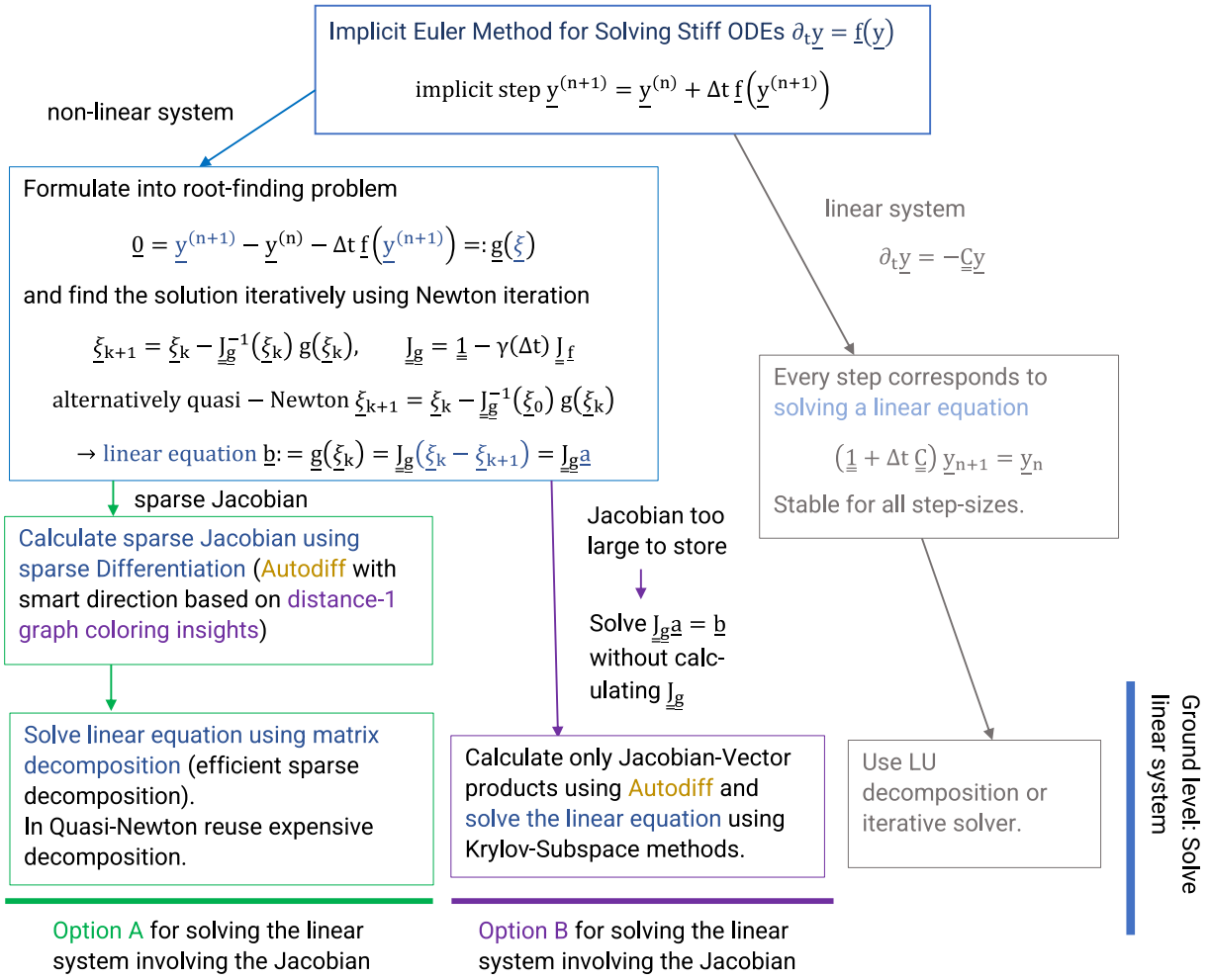


Figure 6: Flowchart of smart approaches to Implicit Euler

At the hand of the *Brusselator* - a non-linear chemical-reaction-diffusion partial differential equation - we studied the performance of Explicit Euler and Implicit Euler at different levels of sophistication as shown in table 1. The more sophisticated implicit method using *Quasi-Newton* as well as smart *graph-coloring* and *sparse factorization* making use of the sparsity of the system significantly outperforms Explicit Euler in runtime and necessary allocations.

Method	Δt	Performance
Explicit Euler	0.0001	1.848 s (1447185 allocations: 5.95 GiB)
Implicit Euler; Newton	0.1	6.737 s (396055 allocations: 11.48 GiB)
Implicit Euler; Quasi-Newton	0.1	3.144 s (197980 allocations: 5.57 GiB)
Implicit Euler; Quasi-Newton, coloring	0.1	877.994 ms (17404 allocations: 1.51 GiB)
Implicit Euler; Quasi-Newton, coloring, sparse factorization	0.1	88.977 ms (25192 allocations: 200.19 MiB)

Table 1: Performance of Explicit and Implicit Euler at different stages of sophistication on a simplified Brusselator in one dimension. For $N = 400$ grid points the simplified Brusselator is numerically solved for a timespan of 10.

The ideas behind these methods are presented in chapter 6 and the implementation can be found under is.gd/stiff.

In conclusion, it is important to use an integrator suited to the problem at hand (e.g. one with favorable geometric or stability properties) and to smartly use sparsity in large systems of differential equations.

We now proceed with a more detailed discussion of the topics introduced and discuss breakdown points of classical methods in chapter 7.

3 Some Classical Methods for Solving ODEs

Our aim is solving an ordinary differential equation (ODE) $\partial_t \underline{y} = \underline{f}\left(\underline{y}\right)$ with initial values $\underline{y}(t = t_0) = \underline{y}_0$. Notice that $\underline{f} = \underline{f}(\underline{y}, t)$ can be handled by augmenting $\tilde{\underline{y}} = \begin{pmatrix} \underline{y} \\ t \end{pmatrix}$ and $\tilde{\underline{f}}(\tilde{\underline{y}}) = \begin{pmatrix} \underline{f}(\tilde{\underline{y}}) \\ 1 \end{pmatrix}$.

3.1 Explicit Euler Method

The simplest method for solving an ODE is the Explicit Euler method

$$\underline{y}^{(n+1)} = \underline{y}^{(n)} + \underline{f}\left(\underline{y}^{(n)}\right)\Delta t, \quad \text{where} \quad \underline{y}^{(0)} = \underline{y}_0$$

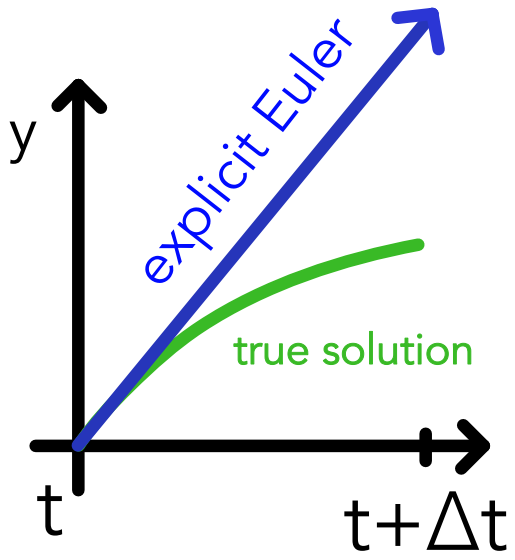


Figure 7: Illustration of one time step in the Explicit Euler scheme.

As illustrated in Figure 7 in every step we step forward along the current derivative $\underline{f}\left(\underline{y}^{(n)}\right)$. A simple error approximation follows from Taylor expansion

$$\underline{y}(t + \Delta t) = \underline{y}(t) + \Delta t \underline{f}(t) + \mathcal{O}_s(\Delta t^2)$$

In each step we make an error $\mathcal{O}_s(\Delta t^2)$ so over some timespan T where we need $N_S = \frac{T}{\Delta t}$ steps we accumulate the error $N_S \mathcal{O}_S(\Delta t^2) = \mathcal{O}_T(\Delta t)$. We therefore call Explicit Euler first order accurate.

Stability issues of Explicit Euler

Stability analysis is a broad field, and the interested reader can find details in chapter IV.3 of Hairer and Wanner, 1996. For now, consider the ODE $\partial_t y = \alpha y, Re(\alpha) < 0, y(0) = y_0$ with the solution $y(t) = y_0 e^{\alpha y}$.

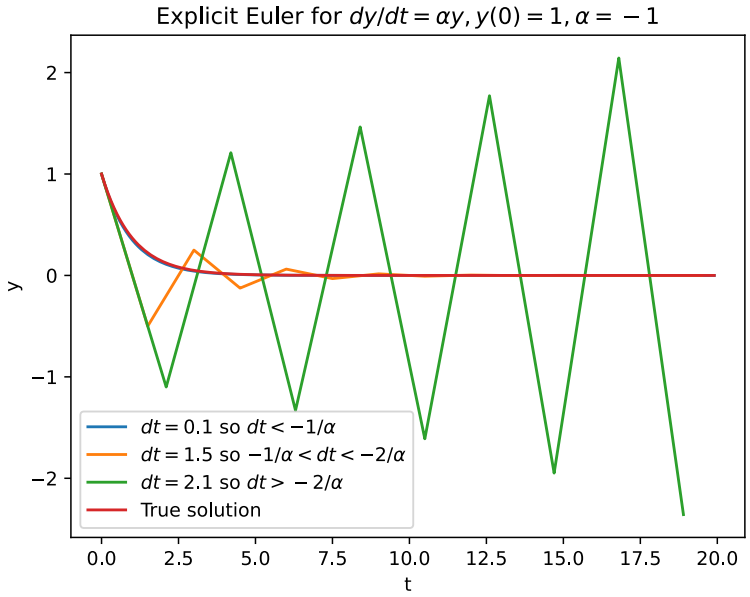


Figure 8: Linear stability of the Explicit Euler scheme.

The results of applying Explicit Euler for different step sizes Δt are shown in figure 8. At a small step size the correct solution is obtained, for a larger step size the numerical solution becomes oscillatory and for even larger step sizes it diverges. We can quantitatively explain this behavior by looking at the Euler steps

$$\begin{aligned} y^{(n+1)} &= y^{(n)} + \alpha y^{(n)} \Delta t \\ &= y^{(n)}(1 + \alpha \Delta t) \\ &= y^{(0)}(1 + \alpha \Delta t)^{n+1} \end{aligned}$$

- $\Delta t < -\frac{1}{\alpha} \rightarrow$ we observe monotonous decrease (ok)
- $-\frac{1}{\alpha} < \Delta t < -\frac{2}{\alpha} \rightarrow$ oscillation (regarding the sign) but still decrease in the absolute value (problematic)
- $-\frac{2}{\alpha} < \Delta t \rightarrow$ an increasing, oscillating solution (very bad)

The growth factor $R(\alpha \Delta t) = 1 + \alpha \Delta t$ in $y^{(n+1)} = R(\alpha \Delta t) y^{(n)}$ is called stability function and

$$\mathcal{D} := \{z \in \mathcal{C} : |R(z)| \leq 1\} \quad \text{so} \quad D_{Euler} = \{z = \alpha\Delta t \in \mathcal{C} : |1 + z| \leq 1\}$$

is called region of absolute stability or linear stability domain. D_{Euler} is a finite region of absolute stability in form of a circle on the left of the complex plane (see figure 9).

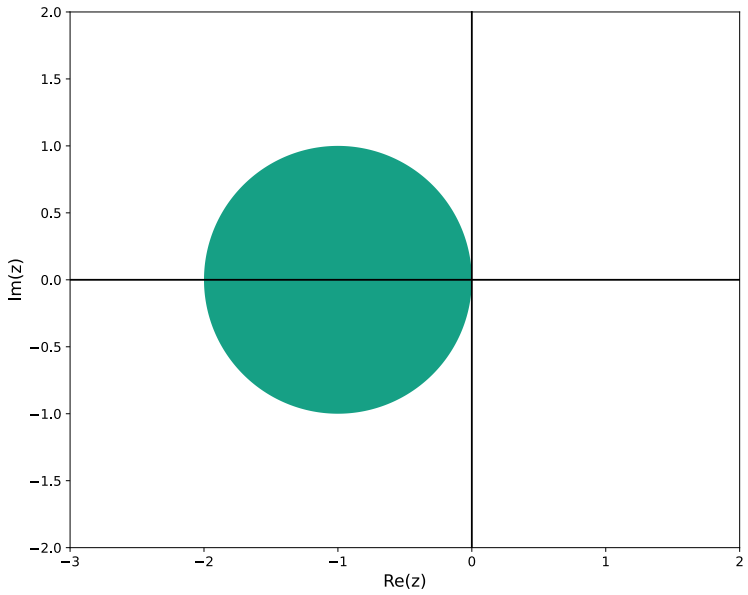


Figure 9: Region of absolute stability of the Explicit Euler method.

3.2 Going higher order: Runge-Kutta methods

We have seen that the Explicit Euler method is of first order, so $O_T(\Delta t)$ (i.e. the error accumulated over integrating over a timespan T is of order Δt so if we cut in half the time-step Δt we also *cut in half* the error accumulated over the timespan). We are now interested in higher order methods. One natural idea is to do the time steps based on higher order Taylor expansions.

As the higher derivatives needed in higher order Taylor expansion can, however, be problematic we use a scheme with the weighted combination of first order derivatives at the heart - Runge-Kutta schemes which can generally be written as (see Chapter 2.4 in Springel et al., 2023 for a derivation)

$$\underline{y}^{(n+1)} = \underline{y}^{(n)} + \Delta t \sum_{i=1}^{m} \beta_i \underline{k}_i$$

$$\underline{k}_i = f\left(\left(\underline{y}^{(n)} + \Delta t \sum_{l=1}^{m-1} \alpha_{i,l} \underline{k}_l\right), t_n + \gamma_i \Delta t\right), \quad i = 1, \ldots, m$$

$$\sum_{l=1}^{m} \alpha_{i,l} = \gamma_i, \quad \alpha_{i,l} = 0 \text{ for } l \geq i \rightarrow \text{ explicit method}$$

Specific schemes like RK2 or RK4 can then be derived by coefficient comparison to a Taylor expansion. To illustrate the Runge-Kutta principle, we introduce the RK2 scheme. To solve the ODE $\partial_t y = f(y)$ with initial condition $y(t = t_0) = y_0$ in the RK2 scheme, we perform steps of the following form

$$k_1 = f\left(y^{(n)}, t_n\right)$$
$$k_2 = f\left(y^{(n)} + \Delta t k_1, t_n + \Delta t\right)$$
$$y^{(n+1)} = y^{(n)} + \frac{\Delta t}{2}\left(k_1 + k_2\right) + \mathcal{O}\left(\Delta t^3\right)$$

which is illustrated in figure 10. By smartly combining the slope at the beginning and end of what would be an Euler step we have yielded a second order scheme.

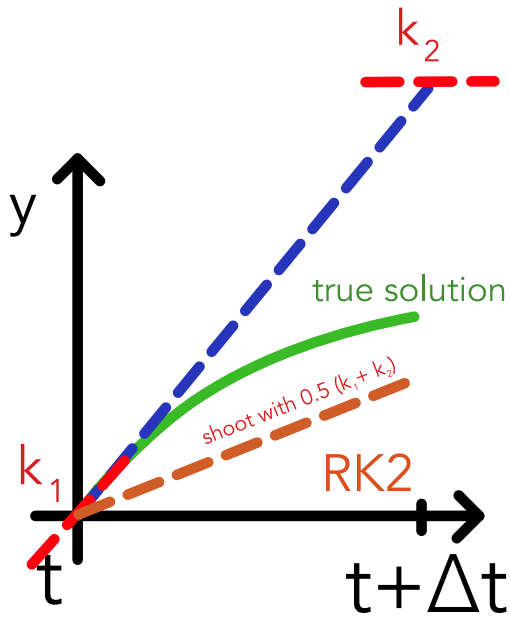


Figure 10: Illustration of the RK2 scheme.

Runge-Kutta schemes are workhorses in the field of solving ODE problems. Popular schemes like the Tsitouras 5/4 Runge-Kutta method (Tsitouras, 2011) make two approximations for the step based on different weighting coefficients (one 4th and one 5th order approximation)

from which an error estimate of the step can be calculated. This is more efficient than the typical time step halving and doubling method, where error estimates are calculated from comparing the results obtained from crossing a timespan with one or alternatively two steps of half the size. The error estimate is, for instance, used for choosing an appropriate adaptive time step Δt.

4 The Problem of Conserved Quantities

4.1 Runge-Kutta methods are unfavorable for simulating Hamiltonian systems

Hamiltonian Systems and Symplecticity

Evolutions of classical physical systems can very generally be stated in terms of equations of motion derived from a real-valued, smooth Hamiltonian $H(\underline{p}, \underline{q})$

$$\begin{aligned} \partial_t \underline{p} &= -\underline{\nabla}_{\underline{q}} H(\underline{p}, \underline{q}) \\ \partial_t \underline{q} &= \underline{\nabla}_{\underline{p}} H(\underline{p}, \underline{q}) \end{aligned} \quad \text{or} \quad \partial_t \underline{y} = \underline{\underline{J}}^{-1} \underline{\nabla} H(\underline{y}), \underline{y} = \begin{pmatrix} \underline{p} \\ \underline{q} \end{pmatrix}, \quad \underline{\underline{J}} = \begin{pmatrix} 0 & \underline{\underline{1}} \\ -\underline{\underline{1}} & 0 \end{pmatrix} \in \mathbb{R}^{2d}$$

where $\underline{p} \in \mathbb{R}^d$ is the generalized momentum and $\underline{q} \in \mathbb{R}^d$ are the generalized coordinates.

Hamiltonian systems

- conserve energy, i. e. $H(\underline{p}, \underline{q})$ is conserved along a trajectory if the Hamiltonian does not explicitly depend on time
- show *symplecticity*, which is most intuitively understood as area conservation in phase space

The mathematical details of symplecticity can be found in chapter VI of Hairer, Wanner, and Lubich, 2006. For our purposes it is sufficient to understand it as preservation of phase space area which will be illustrated in the following examples.

Runge-Kutta methods do not conserve energy and are not symplectic

Let us now apply RK2 to the two-body-problem reduced to a dimensionless one-body-problem as described by

$$\frac{d\underline{s}}{d\tau} = \underline{w}, \quad \frac{d\underline{w}}{d\tau} = -\frac{\underline{s}}{s^3}$$

where $\underline{s}$ is the position and $\underline{w}$ the velocity. The numerical solution obtained using RK2 can be seen in figure 11. One can see that neither the energy nor the angular momentum nor the Runge-Lenz vector are conserved. Every step in the scheme is erroneous, and those errors add up leading to our quantities of interest not being conserved.

Let us look at another physical problem, the pendulum, as it lends itself well to phase space visualization. We see in figure 12 that the phase space area, the area spanned by the test

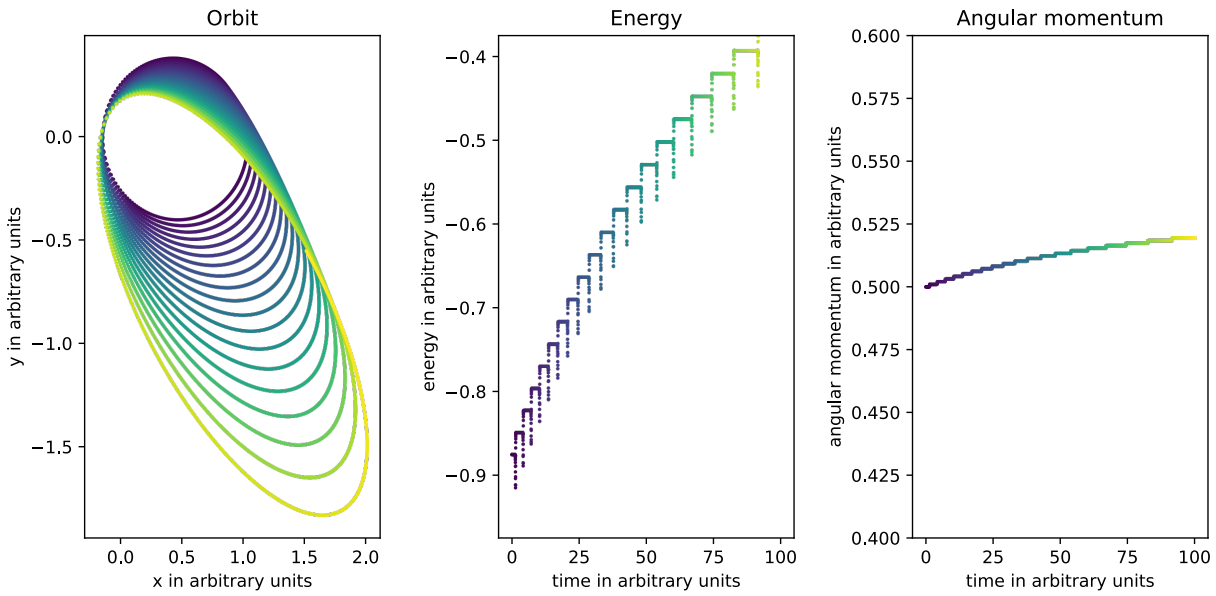


Figure 11: Numerical solution of the two-body-problem using RK2. The left panel shows the orbit, the central one the energy and the right one the angular momentum. Time is also encoded in the form of color, going from a dark blue to yellow.

points as they propagate in time, is not preserved as it should be for Hamiltonian flow.

We need to switch to a whole other class of integrators.

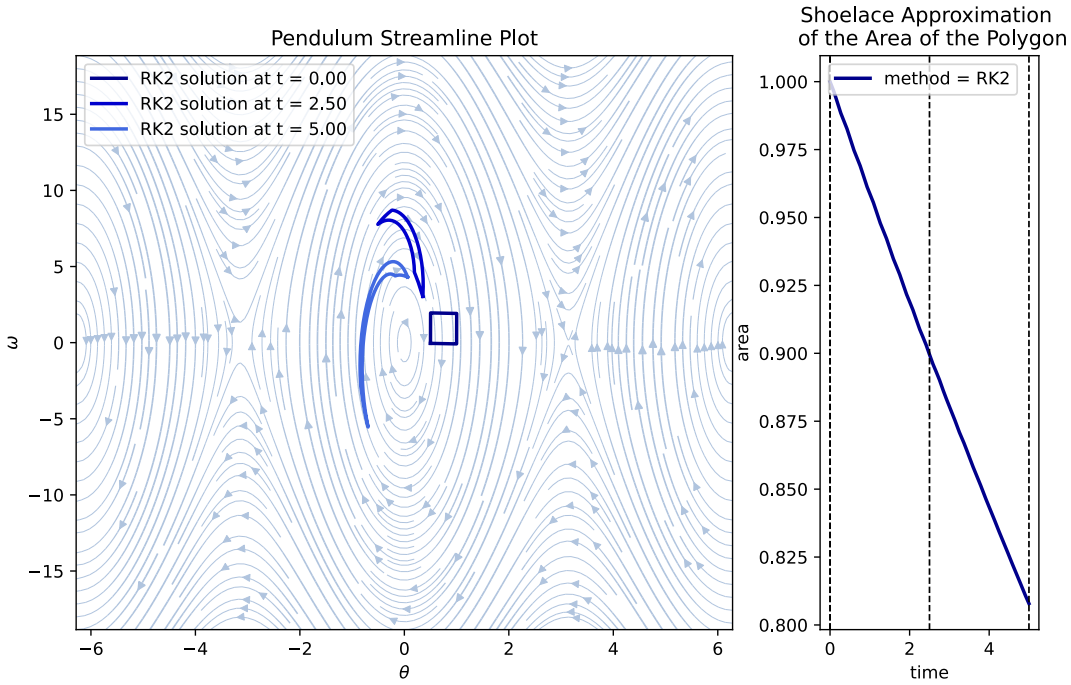


Figure 12: The left panel shows solutions to the pendulum problem at different points in time for different initial values as obtained by the RK2 scheme. The initial values are chosen so that they initially span a square in phase space. The right panel shows the phase space area spanned by the solutions as a function of time.

4.2 Symplectic integrators to the help

This leads us to geometric integrators which are numerical methods preserving geometric properties of the exact flow of ODEs (Hairer, Wanner, and Lubich, 2006). More specifically for Hamiltonian systems we use **Symplectic Integrators** which produce a flow in phase space that is symplectic just as the flow of the exact solution (Hairer, Wanner, and Lubich, 2006, chapter VI).

It turns out that preserving symplecticity and energy at the same time is very difficult (Hairer, 2006). However, symplectic integrators still have good energy conservation properties without much long-term drifts. The general idea behind the connection between symplecticity and energy conservation is that geometric properties of the integrator translate into structure preservation on the level of modified equations (Hairer, Wanner, and Lubich, 2006, preface and chapters X through XII).

The Leapfrog Method

Let us illustrate the effect of using a symplectic method - the leapfrog method - on energy and phase space conservation. Consider a classical physical problem with a particle at

position $\underline{s}$ with velocity $\underline{v}$ and an acceleration $\underline{a}$ that depends only on the position.

The leapfrog scheme can then be written as

$$\underline{s}(t+\frac{1}{2}\Delta t) = \underline{s}(t-\frac{1}{2}\Delta t) + \underline{v}(t)\Delta t$$
$$\underline{v}(t+\Delta t) = \underline{v}(t) + \underline{a}(t+\frac{1}{2}\Delta t)\Delta t$$

The position and velocity are updated at alternating half time steps as illustrated in figure 13.

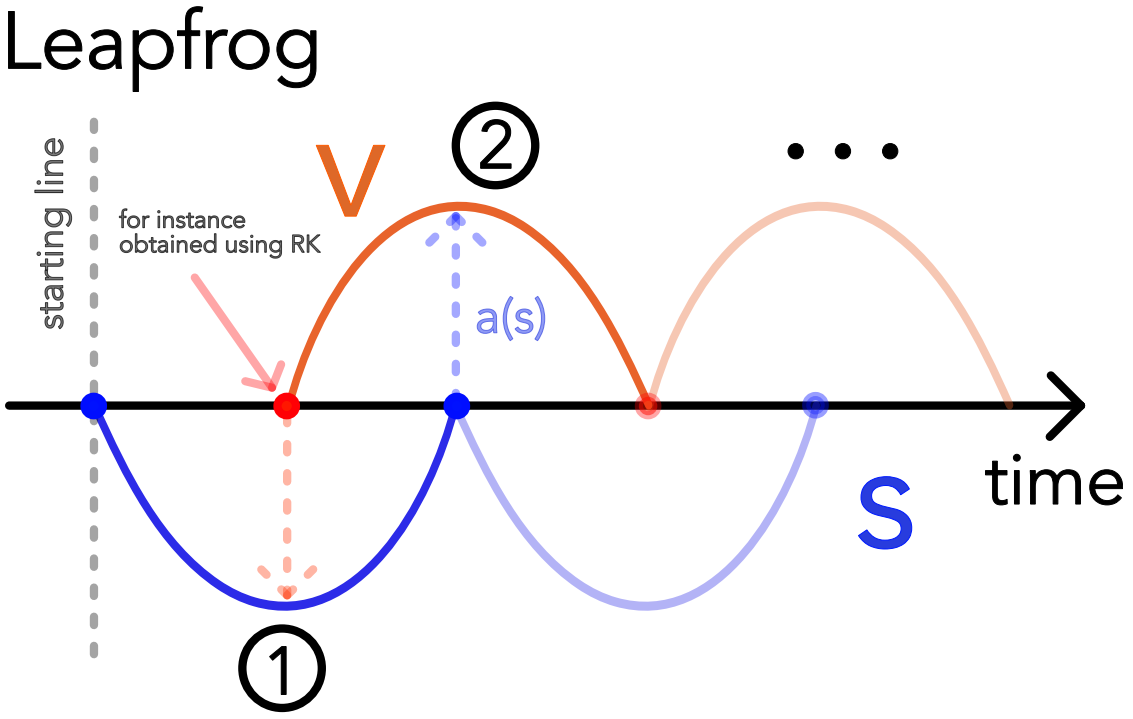


Figure 13: Illustration of the leapfrog scheme.

One problem of the Leapfrog scheme in this form is that velocity and position information are not available at the same time. This can be circumvented by using the kick-drift-kick formulation of the scheme:

$$\underline{v}(t+\frac{1}{2}\Delta t) = \underline{v}(t) + \underline{a}(t)\frac{\Delta t}{2} \quad \text{kick}$$
$$\underline{s}(t+\Delta t) = \underline{s}(t) + \underline{v}(t+\frac{1}{2}\Delta t)\Delta t \quad \text{drift}$$
$$\underline{v}(t+\Delta t) = \underline{v}(t+\frac{1}{2}\Delta t) + \underline{a}(t+\Delta t)\frac{\Delta t}{2} \quad \text{kick}$$

where a drift is a change of the position with constant velocity and a kick is a change of the velocity with constant acceleration.

The leapfrog scheme is symplectic, has good energy conservation properties and is time reversible (proofs in Springel et al., 2023, chapter 2.8).

Symplecticity (so area conservation in phase space) is illustrated at the hand of the pendulum in figure 14 and energy conservation at the hand of the two-body-problem in figure 15. The area in phase space does not change at all while the energy shows small fluctuations but no overall drift (compare Hairer, Lubich, et al., 2003, theorem 5.5). The angular momentum is exactly conserved in the leapfrog solution to the two-body-problem (details on the conservation of specific *quadratic first integrals* can be found in Hairer, Lubich, et al., 2003, theorem 3.5). As visible in the changing orientation of the orbit in figure 15 the leapfrog scheme does not preserve the orientation of the Runge-Lenz vector.

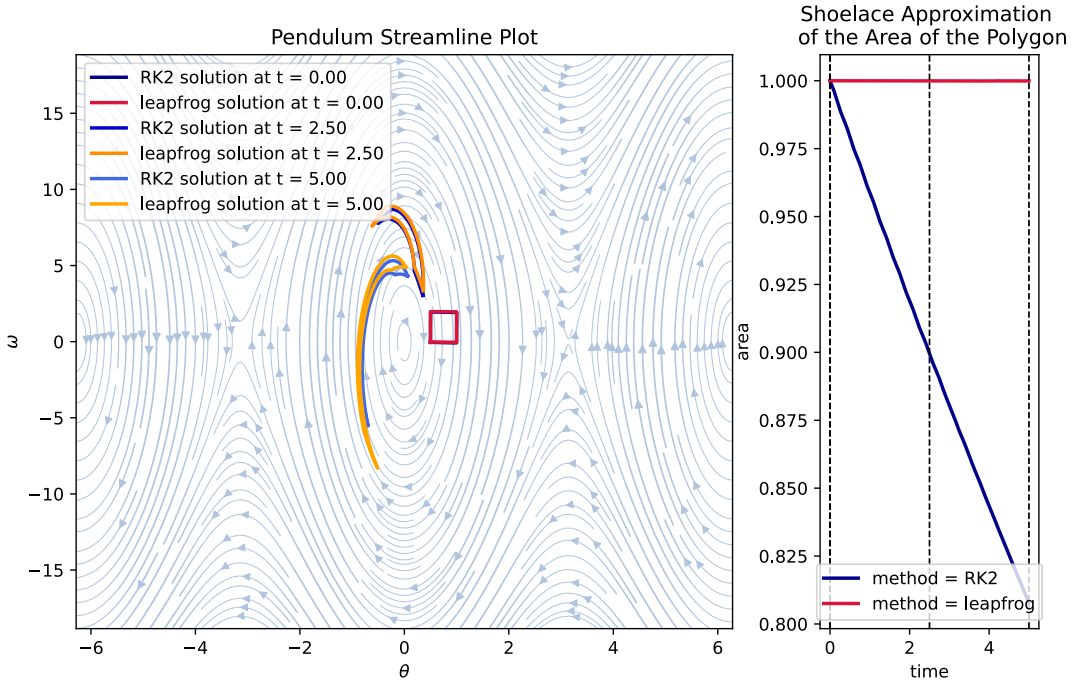


Figure 14: The same situation as in figure 12 is shown but now with the result of the leapfrog method added. The leapfrog scheme preserves the area in phase space while the RK2 scheme does not.

Indeed, it turns out that the leapfrog scheme exactly solves the modified Hamiltonian

$$H_{leap} = H + H_{err}, \quad H_{err} \propto \frac{\Delta t^2}{12} \left\{ \{H_{kin}, H_{pot}\}, H_{kin} + \frac{1}{2} H_{pot} \right\} + \mathcal{O}(\Delta t^3)$$

where H_{kin} and H_{pot} are the kinetic and potential part of the original Hamiltonian (Springel et al., 2023, chapter 2.8). The curly brackets denote the Poisson bracket.

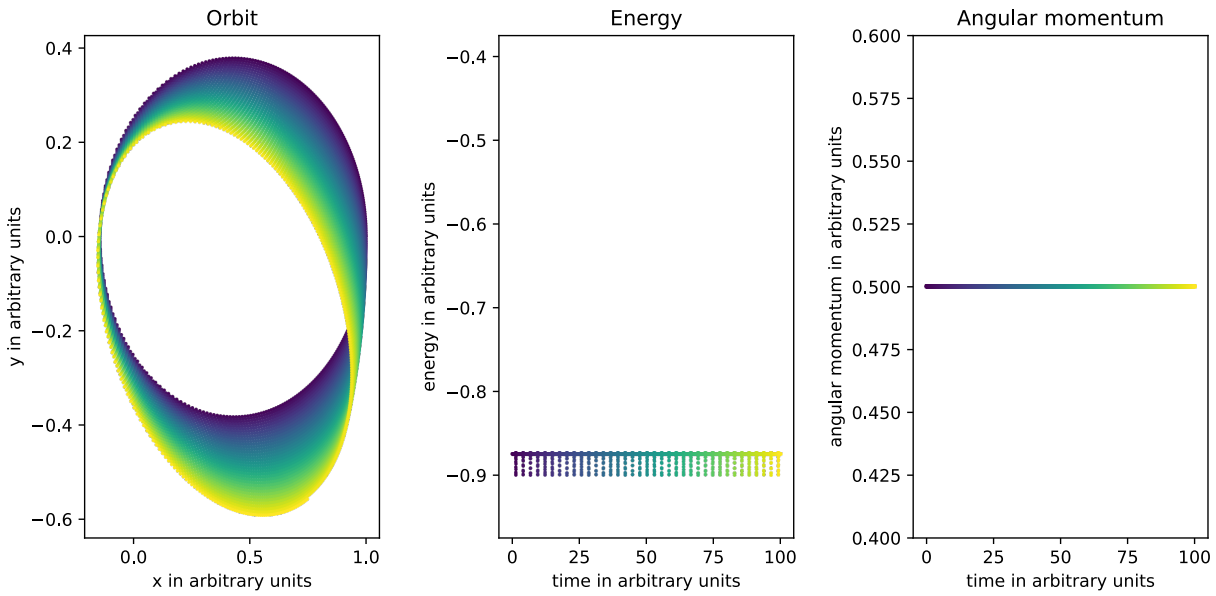


Figure 15: Numerical solution of the two-body-problem using the leapfrog method in the kick-drift-kick scheme. The left panel shows the orbit, the central one the energy and the right one the angular momentum. Time is also encoded in the form of color, going from a dark blue to yellow.

5 The Problem of Stiffness

5.1 Introducing stiffness at the hand of a simple example

Consider the following ODE system (following Press et al., 2007, chapter 17.5)

$$\begin{aligned}\partial_t y_1 &= 998 y_1 + 1998 y_2 \\ \partial_t y_2 &= -999 y_1 - 1999 y_2\end{aligned}$$

with initial conditions $y_1(0) = 1$ and $y_2(0) = 0$. The system can be represented in matrix form as

$$\partial_t \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \underline{\underline{A}} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \quad \underline{\underline{A}} = \begin{pmatrix} 998 & 1998 \\ -999 & -1999 \end{pmatrix}$$

The eigenvalues of $\underline{\underline{A}}$ are $\lambda_1 = -1$ and $\lambda_2 = -1000$. The eigenvectors are $\underline{e}_1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ and $\underline{e}_2 = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$. The solution of the system is then

$$\begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \exp\left(\underline{\underline{A}}t\right) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \end{pmatrix} \exp\left(-1t\right) + \begin{pmatrix} -1 \\ 1 \end{pmatrix} \exp\left(-1000t\right)$$

Let us now apply the Explicit Euler method to this system for different time-steps Δt. The result is shown in figure 16.

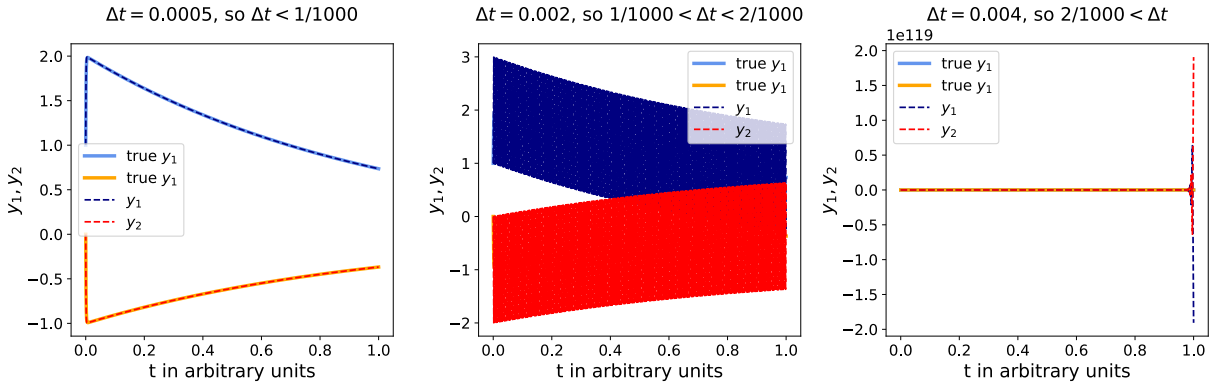


Figure 16: Numerical solution to the linear system $\partial_t \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \underline{\underline{A}} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ with $\underline{\underline{A}} = \begin{pmatrix} 998 & 1998 \\ -999 & -1999 \end{pmatrix}$ and $y_1(0) = 1$, $y_2(0) = 0$ using the Explicit Euler method for different time-steps Δt. The left panel shows the solution for $\Delta t = 0.0005$, the central one for $\Delta t = 0.002$ and the right one for $\Delta t = 0.004$.

Let us think back to the linear stability analysis of the Explicit Euler scheme for $\partial_t y = \alpha y, Re(\alpha) < 0, y(0) = y_0$. We had obtained

- $\Delta t < -\frac{1}{\alpha} \rightarrow$ we observe monotonous decrease (ok)
- $-\frac{1}{\alpha} < \Delta t < -\frac{2}{\alpha} \rightarrow$ oscillation (regarding the sign) but still decrease in the absolute value (problematic)
- $-\frac{2}{\alpha} < \Delta t \rightarrow$ an increasing, oscillating solution (very bad)

The same result holds in principle for our linear system - but with α replaced by the eigenvalue of largest magnitude of $\underline{\underline{A}}$, here $\lambda_2 = -1000$ (for the proof see Press et al., 2007, chapter 17.5).

As we move away from the origin, the fastest decreasing term $\propto \exp\left(-\lambda_2 t\right)$ in the true solution is completely negligible. However, in the explicit scheme it still sets the timescale that has to be resolved for a stable solution.

In the setting of $\partial_t y = \alpha y, Re(\alpha) < 0$ the stability constraint for Δt is not too problematic because the resulting step-size is reasonable compared to the timescale of the problem. In

the case of an ODE with different timescales in the solution, however, we are often interested in the timescale of the slowest processes but in the explicit scheme we still need to resolve the fastest timescale which quickly becomes infeasable. This is the problem of stiffness and can - in such a linear setting with all negative eigenvalues of $\underline{\underline{A}}$ - be characterized by the stiffness ratio

$$\text{stiffness ratio} := \frac{\max_{\text{eigenvalues } \lambda_i \text{ of } \underline{\underline{A}}} |\text{Re } \lambda_i|}{\min_{\text{eigenvalues } \lambda_i \text{ of } \underline{\underline{A}}} |\text{Re } \lambda_i|} = \frac{\lambda_2}{\lambda_1} = 1000$$

A large stiffness ratio indicates that an explicit scheme like the Explicit Euler method would be very inefficient for following the slowest process.

5.2 A *definition* of stiffness

As discussed in Lambert, 1991 a hard mathematical definition of stiffness is difficult and we therefore resort to the broad practical definition (Lambert, 1991, chapter 6)

> »If a numerical method with a finite region of absolute stability, applied to a system with any initial conditions, is forced to use in a certain interval of integration a step length which is excessively small in relation to the smoothness of the exact solution in that interval, then the system is said to be stiff in that interval.«

An example for a numerical method with a finite region of absolute stability is the Explicit Euler method (see figure 9). In the example above, in spite of the fact that the solution is very smooth and the term $\propto \exp(-\lambda_2 t)$ is quickly negligible, we have to use excessively small steps.

6 Efficiently Solving Stiff ODEs

6.1 Implicit Euler to the help

At the core of dealing with stiffness are implicit methods, the simplest representative being Implicit Euler.

An Implicit Euler step for solving $\partial_t \underline{y} = \underline{f}\left(\underline{y}\right)$ is given by

$$\underline{y}^{(n+1)} = \underline{y}^{(n)} + \underline{f}\left(\underline{y}^{(n+1)}\right)\Delta t \quad \text{where} \quad \underline{y}^{(0)} = \underline{y}_0$$

which is an implicit equation as $\underline{f}$ is evaluated at the new time step $\underline{y}^{(n+1)}$.

Region of absolute stability of the Implicit Euler method

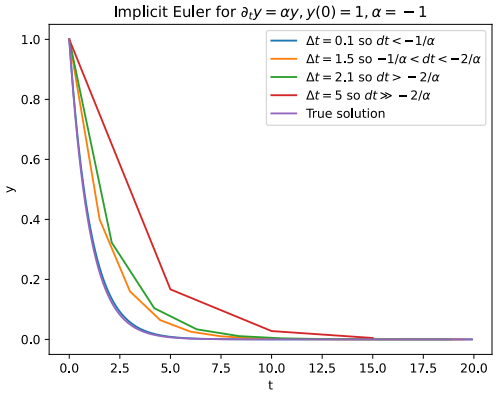
As for the Explicit Euler method, we perform a linear stability analysis of the Implicit Euler method for $\partial_t y = \alpha y, Re(\alpha) < 0, y(0) = y_0$. We obtain

$$y^{(n+1)} = y^{(n)} + \alpha y^{(n+1)}\Delta t \quad \Rightarrow \quad y^{(n+1)} = \frac{1}{1-\alpha\Delta t} y^{(n)}$$

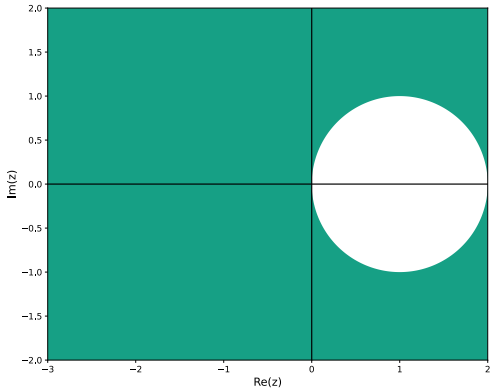
which decreases for any $\Delta t > 0$ (illustrated in figure 17a). For large time-steps, the result is inaccurate (Implicit Euler is a first order scheme) but the solution remains stable. As of the stability function $R(z) = \frac{1}{1-z}$ the region of absolute stability is given by

$$\mathcal{D}_{\text{implicit euler}} = \{z \in \mathbb{C} \mid |R(z)| < 1\} = \{z \in \mathbb{C} \mid |1-z| > 1\}$$

which is illustrated in figure 17b. The whole left half plane is included in the region of absolute stability and the method is therefore unconditionally stable.



(a) Linear stability of Implicit Euler.



(b) Region of absolute stability of the Implicit Euler method (shaded in green).

Figure 17: Stability of the Implicit Euler scheme.

Implicit Euler for stiff linear ODEs

As Implicit Euler is unconditionally stable, the fast oscillating terms resulting from

$$\partial_t \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \underline{\underline{A}} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \quad \underline{\underline{A}} = \begin{pmatrix} 998 & 1998 \\ -999 & -1999 \end{pmatrix}$$

with initial conditions $y_1(0) = 1$ and $y_2(0) = 0$ are no problem as illustrated in figure 18, where in spite of the relatively large time-step a good approximation of the solution is obtained.

The implicit step for such a linear system $\partial_t \underline{y} = \underline{\underline{A}}\underline{y}$ is

$$\underline{y}^{(n+1)} = \underline{y}^{(n)} + \underline{\underline{A}}\underline{y}^{(n+1)}\Delta t \quad \Rightarrow \quad \left(\underline{\underline{1}} - \underline{\underline{A}}\Delta t\right) \underline{y}^{(n+1)} = \underline{y}^{(n)}$$

which means that to make a step we have to solve a linear system which is usually done by matrix decomposition (like LU decomposition).

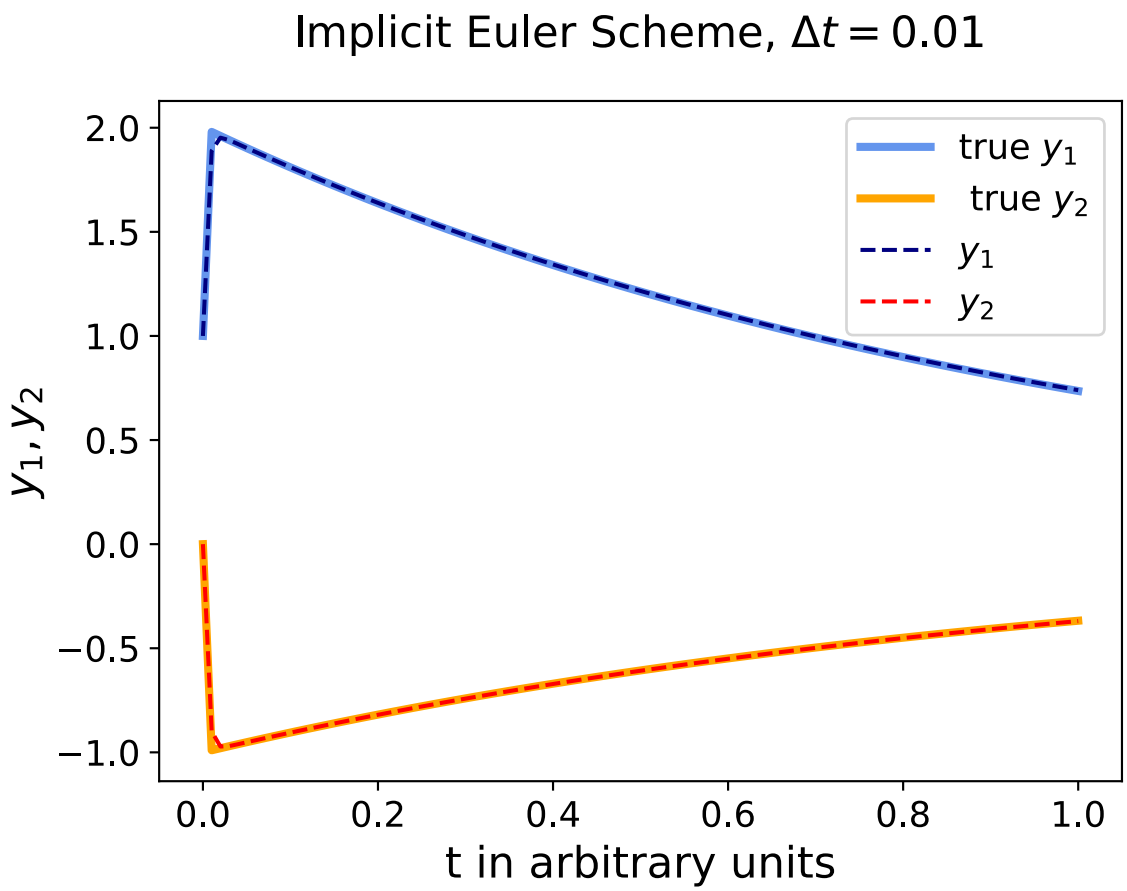


Figure 18: The same problem as in figure 16 is now approached using the Implicit Euler method and a relatively large time-step of $\Delta t = 0.01$.

But how can we approach non-linear ODEs using the Implicit Euler method?

To perform an implicit step

$$\underline{y}^{(n+1)} = \underline{y}^{(n)} + \underline{f}\left(\underline{y}^{(n+1)}\right) \Delta t$$

for a non-linear system $\partial_t \underline{y} = \underline{f}\left(\underline{y}\right)$ like the Davis-Skodje equation

$$\begin{aligned} \dot{y}_1(t) &= -y_1(t) \\ \dot{y}_2(t) &= -\gamma y_2(t) + \frac{(\gamma - 1) y_1(t) + \gamma y_1^2(t)}{(1 + y_1(t))^2} \end{aligned}$$

where γ is a measure for the stiffness (see Heiter, 2012, chapter 2.4) we reformulate the implicit step as a root-finding problem

$$\underline{0} = \underline{y}^{(n+1)} - \underline{y}^{(n)} - \Delta t \underline{f}\left(\underline{y}^{(n+1)}\right), \quad \underline{g}\left(\underline{\xi}\right) := \underline{\xi} - \underline{y}^{(n)} - \Delta t \underline{f}\left(\underline{\xi}\right)$$
$$\rightarrow \underline{0} = \underline{g}\left(\underline{\xi}\right) \Leftrightarrow \underline{\xi} = \underline{y}^{(n+1)}$$

where each of those time-steps is solved using Newton's method (or quasi-Newton)

$$\underline{\xi}_{k+1} = \underline{\xi}_k - \underline{\underline{J}}_{\underline{g}}^{-1}\left(\underline{\xi}_k\right) \underline{g}\left(\underline{\xi}_k\right), \quad \underline{\underline{J}}_{\underline{g}} = \underline{\underline{1}} - \Delta t \gamma \left(\underline{\xi}_k\right) \underline{\underline{J}}_{\underline{f}}$$

$$\underline{\xi}_0 = \underline{y}^{(n)}, \quad \underline{\xi}_m \rightarrow \underline{y}^{(n+1)} \quad \text{for} \quad m \rightarrow \infty$$

where $\underline{\underline{J}}_{\underline{f}}$ is the Jacobian of $\underline{f}$. In Quasi-Newton the Jacobian is only recalculated once per time-step in the Euler method

$$\underline{\xi}_{k+1} = \underline{\xi}_k - \underline{\underline{J}}_{\underline{g}}^{-1}\left(\underline{\xi}_0\right) \underline{g}\left(\underline{\xi}_k\right)$$

For the Davis-Skodje problem mentioned above some Implicit Euler steps are drawn into the stream plot of the equation in figure 19. Here, one can also see the intuition behind Implicit Euler steps: One searches a point where the derivative is such that shooting back with this slope leads back to the point we are coming from, as

$$\underline{y}^{(n)} = \underline{y}^{(n+1)} - \underline{f}\left(\underline{y}^{(n+1)}\right) \Delta t$$

The steps of the Newton iteration done for each Implicit Euler step can most intuitively be understood in the formulation as the linear equation

$$\underline{b} := \underline{g}\left(\underline{\xi}_k\right) = \underline{\underline{J}}_{\underline{g}}\left(\underline{\xi}_k - \underline{\xi}_{k+1}\right) = \underline{\underline{J}}_{\underline{g}} \underline{a}, \quad \underline{a} := \underline{\xi}_k - \underline{\xi}_{k+1}$$

which is also the equation solved on the computer using matrix decomposition. $\underline{\underline{J}}_{\underline{g}} \underline{a}$ is the directional derivative of $\underline{g}$ in the direction of $\underline{a}$ and in a step of the Newton iteration we search for a step $\underline{a}$ that gets us from $\underline{0}$ to $\underline{b}$ in other words $\underline{\xi}_{k+1} = \underline{\xi}_k - \underline{a}$.

Efficiency considerations were intentionally left out here so that the general idea becomes clearer but are at the center of the next and main section.

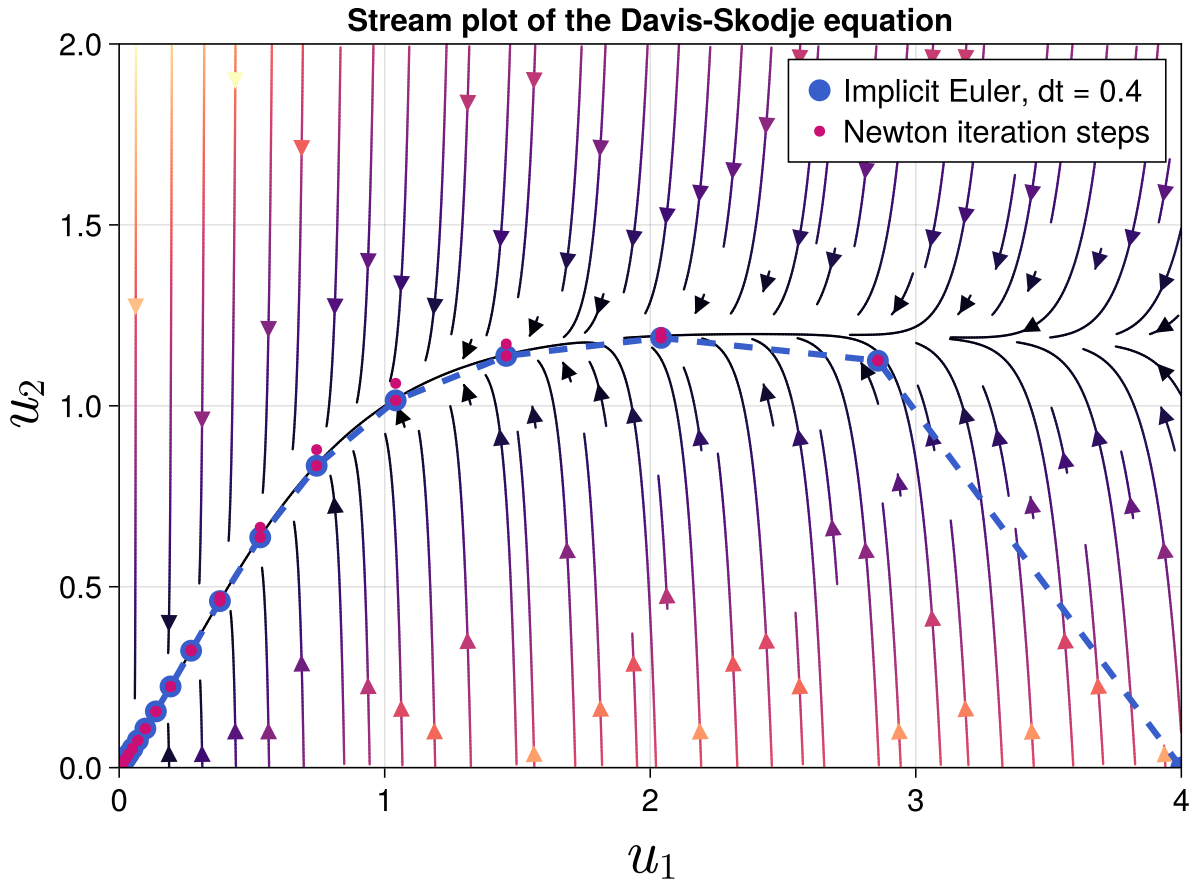


Figure 19: Stream plot of the Davis-Skodje equation with some Implicit Euler steps also drawn. The direction of the Implicit Euler steps is from right (starting at (4, 0)) to left.

6.2 Solving larger systems is difficult

Introduction to the example problem - the Brusselator

As our main example, we use a simplified *Brusselator* as introduced in Hairer, Wanner, and Nørsett, 1993, chapter I.16 and further discussed in Hairer and Wanner, 1996, chapter IV.1 (originally introduced in Lefever and Nicolis, 1971).

Some details on the Brusselator and all of our implementations regarding numerical methods for solving it can be found in the accompanying Julia notebook.

Here, it is sufficient to know that we consider a non-linear chemical-reaction-diffusion partial differential equation in one dimension of the form

$$\frac{\partial u}{\partial t} = A + u^2 v - (B+1)u + \alpha \frac{\partial^2 u}{\partial x^2}$$
$$\frac{\partial v}{\partial t} = Bu - u^2 v + \alpha \frac{\partial^2 v}{\partial x^2}$$

where $u(x,t)$ and $v(x,t)$ are the concentrations of chemical substances, α is a diffusion constant and A and B are fixed concentrations of other substances.

From discretizing the differentiation in space (i.e. using the method of lines for approaching this partial differential equations) we follow (with $x_i = \frac{i}{N+1}(1 \leq i \leq N), \Delta x = \frac{1}{N+1}$, $A = 1$, $B = 3$, $\alpha = \frac{1}{50}$)

$$\begin{aligned} u_i' &= 1 + u_i^2 v_i - 4u_i + \frac{\alpha}{(\Delta x)^2}\left(u_{i-1} - 2u_i + u_{i+1}\right), \\ v_i' &= 3u_i - u_i^2 v_i + \frac{\alpha}{(\Delta x)^2}\left(v_{i-1} - 2v_i + v_{i+1}\right) \\ u_0(t) &= u_{N+1}(t) = 1, \quad v_0(t) = v_{N+1}(t) = 3 \\ u_i(0) &= 1 + \sin\left(2\pi x_i\right), \quad v_i(0) = 3, \quad i = 1, \ldots, N. \end{aligned} \tag{1}$$

where some boundary conditions and initial conditions have been chosen. The constant boundary values are enforced using so-called ghost-cells in the implementation.

We compactly write the differential equation system as

$$\underline{y} = \underline{f}\left(\underline{y}\right), \quad \underline{y} = \begin{pmatrix} u_0 \\ \vdots \\ u_{N+1} \\ v_0 \\ \vdots \\ v_{N+1} \end{pmatrix}$$

where $\underline{f} : \mathbb{R}^{2N} \to \mathbb{R}^{2N}$ follows from equation 1.

The occurrence of stiffness

Let us start by applying the Explicit Euler method to the simplified Brusselator with $N = 40$ grid points and a step size $dt = 0.01$. The result is shown in figure 20a and is in agreement with the literature results (Hairer and Wanner, 1996, chapter IV.1).

But if we increase N to $N = 400$, i.e. we decrease the spacing between the grid points $\Delta x = \frac{1}{N+1}$, the Explicit Euler scheme yields a diverging result (see figure 20b).

We can get back to a stable solution by decreasing the step size but notice that we have to use a much smaller step size, e.g. $dt = 0.0001$ (see figure 20c), in spite of the solution still being very smooth.

In fact even a more sophisticated explicit method like the Tsitouras 5/4 Runge-Kutta method

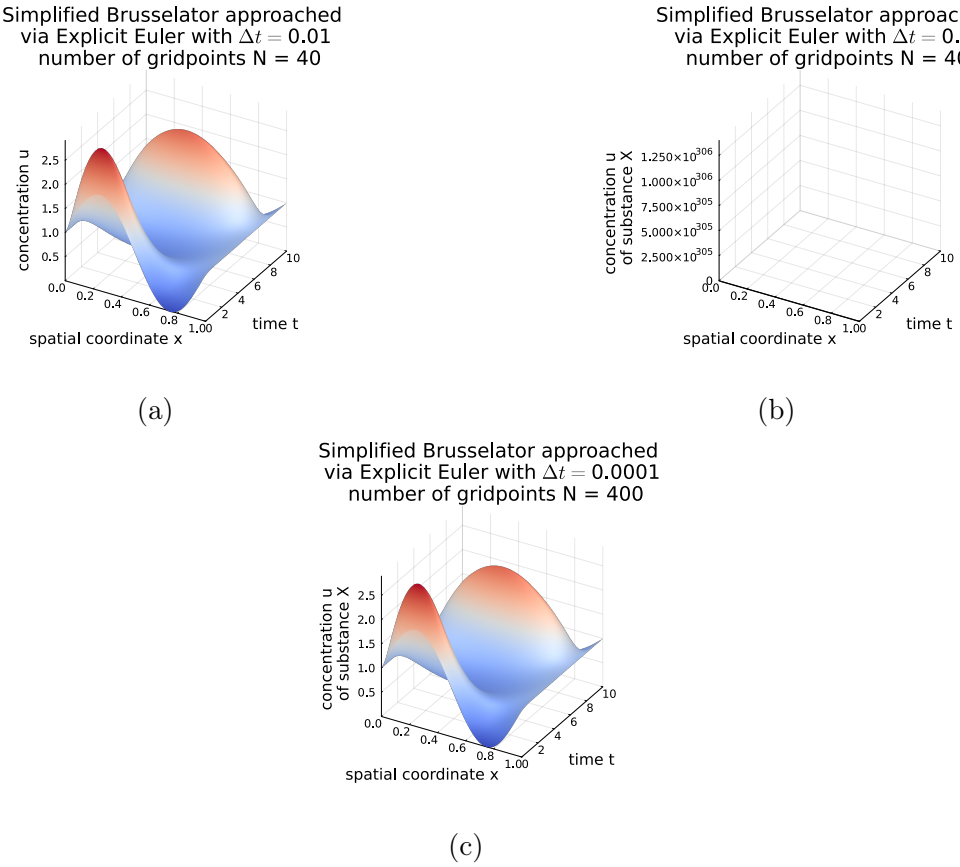


Figure 20: Numerical solutions to a simplified Brusselator using the Explicit Euler method with different numbers N of grid points and time-steps Δt.

from the DifferentialEquations.jl package (Rackauckas and Nie, 2017) will use excessively many steps (e.g. to cover a time interval of length 10 (dimensionless as of our problem formulation) using the default settings 220047 evaluations of $\underline{f}$ are used). The problem of stiffness as described in section 5.2 has occurred.

Understanding stiffness in a diffusive context

The transport process at play is diffusion. For a diffusive process we know the spreading of some concentration to follow $\sigma = \sqrt{2\alpha t}$. Now in each Euler step we do, only neighboring cells have an effect on each other (compare the discretized ODE we introduced in the beginning). Therefore - in the style of a Courant-Friedrichs-Levy criterion (Courant et al., 1928) - we can propose the stability constraint

$$\Delta x > \sigma(\Delta t) = \sqrt{2\alpha t}$$

so $\Delta t < \frac{\Delta x^2}{2\alpha}$. If we want to double N (cut in half Δx) we need $\mathcal{O}(N^2)$ more time-steps with the complexity of a function evaluation scaling with $\mathcal{O}(N)$ resulting in a $\mathcal{O}(N^3)$ scaling -

calculations quickly become unfeasible.

Let us note that in the simplified Brusselator at hand it is the diffusive term causing stiffness, but in a more complex model the chemical reactions could be an additional factor of stiffness (see e.g. Chou et al., 2007).

A naive implementation of Implicit Euler and its limitations

In the Implicit Euler method each implicit step to get from $\underline{y}^{(n)}$ to $\underline{y}^{(n+1)}$ is reformulated as a root-finding problem of a function $\underline{g}$ which we approach using Newton iteration. In every Newton step

$$\underline{b} := \underline{g}\left(\underline{\xi}_k\right) = \underline{\underline{J}}_{\underline{g}}\left(\underline{\xi}_k - \underline{\xi}_{k+1}\right) = \underline{\underline{J}}_{\underline{g}}\underline{a}$$

must be solved where the Jacobian is evaluated at $\underline{y}^{(n)}$ in the quasi-Newton method and at $\underline{\xi}_n$ in the Newton method.

The above linear equations leads to the following questions

- How to construct the Jacobian?
- How to solve the linear system?

Constructing the Jacobian

Regarding the construction of the Jacobian let us remember that we can calculate a Jacobian vector product (directional derivative)

$$\underline{\underline{J}}_{\underline{g}}\underline{v} = \left(\nabla \cdot \underline{g}\right)\underline{v}, \quad \underline{\underline{J}}_{\underline{g}}\underline{e}_j \text{ calculates } j\text{-th column of Jacobian}$$

to machine precision using one forward mode AD calculation

$$\underline{\underline{J}}_{\underline{g}}\left(\underline{\xi} + \epsilon\underline{v}\right) = \underline{\underline{J}}_{\underline{g}}\underline{\xi} + \epsilon\underline{\underline{J}}_{\underline{g}}\underline{v}, \quad \text{dual number } \underline{\xi} + \epsilon\underline{v}, \quad \epsilon^2 = 0$$

Now, in the simplest approach, we construct the Jacobian column by column $\underline{\underline{J}}_{\underline{g}}\hat{\underline{e}}_j$ by

$$\underline{g}(\underline{\xi} + \epsilon\hat{\underline{e}}_j) = \underline{g}(\underline{\xi}) + \epsilon\underline{\underline{J}}_{\underline{g}}\hat{\underline{e}}_j, \quad j = 1, \ldots, 2(N+2)$$

which takes $\mathcal{O}(N)$ evaluations of $\underline{f}$, where each of those evaluations is in our example $\mathcal{O}(N)$ itself.

Method	Δt	Performance
Explicit Euler	0.0001	1.848 s (1447185 allocations: 5.95 GiB)
Implicit Euler; Newton	0.1	6.737 s (396055 allocations: 11.48 GiB)
Implicit Euler; Quasi-Newton	0.1	3.144 s (197980 allocations: 5.57 GiB)

Table 2: Performance of Explicit and naively implemented Implicit Euler on a simplified Brusselator. For $N = 400$ the simplified Brusselator is numerically solved for a timespan of 10.

Solving the linear systems

A linear system $\underline{\underline{J_g}}\,\underline{a} = \underline{b}$ is usually solved by matrix decomposition. While there are sophisticated techniques for inverting a matrix, the inverse can potentially be very dense even if the matrix itself is not, which would infer with smart memory optimizations.

A typical decomposition is the LU decomposition where we decompose the matrix, here $\underline{\underline{J_g}}$, into a lower-left triangular matrix $\underline{\underline{L}}$ and an upper-right triangular matrix $\underline{\underline{U}}$ such that $\underline{\underline{P}}\,\underline{\underline{J_g}} = \underline{\underline{L}}\,\underline{\underline{U}}$ ($\underline{\underline{P}}$ is a permutation matrix that might be necessary). The decomposition can be done in $\mathcal{O}(N^3)$ and solving the linear system then consists of

1. Solving $\underline{\underline{L}}\,\underline{y} = \underline{b}$ for $\underline{y}$ (forward substitution)
2. Solving $\underline{\underline{U}}\,\underline{a} = \underline{y}$ for $\underline{a}$ (backward substitution)

and is done in $\mathcal{O}(N^2)$.

While Newtons method has the advantage of quadratic convergence over quasi-Newton, the expensive decomposition $\mathcal{O}(N^3)$ has to only be done once per Implicit Euler step not once per Newton iteration. In quasi-Newton we need more Newton steps until convergence but in each step we only have to do the $\mathcal{O}(N^2)$ solve which is advantageous for sufficiently large systems.

Results and limitations of the naive approach

For $N = 400$ even at a large step-size of $\Delta t = 0.1$ we obtain the correct solution using the implicit scheme. The performance, however, is even worse than the one of Explicit Euler with small time-steps, see table 2.

The bad performance has two reasons

1. The Jacobian is constructed naively column by column which takes $\mathcal{O}(N)$ function evaluations of $\underline{f}$
2. The linear system is solved using a dense LU decomposition which is $\mathcal{O}(N^3)$ (but still a highly optimized problem)

We therefore end up with the same scaling as in the explicit scheme ($\mathcal{O}(N^2)$ steps with one evaluation of $\underline{f}$ per step which is $\mathcal{O}(N)$).

Overview on smarter approaches to implicit steps in large systems

Note that for large systems of differential equation the Jacobian is usually sparse as variables in the equation system oftentimes only depend on a few others. In our case, both concentrations we consider per grid point in fact only depend on the concentrations of the neighboring grid points leading to the sparsity pattern shown in figure 21 for $N = 10$.

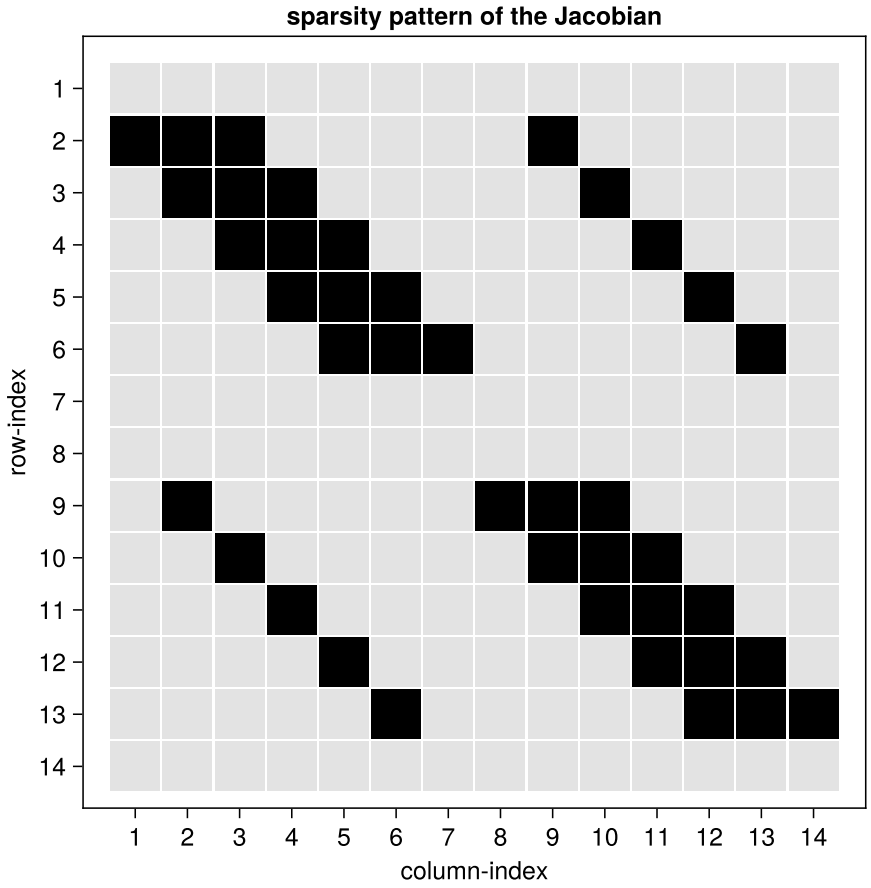


Figure 21: Sparsity pattern of the Jacobian of the simplified Brusselator for $N = 10$. In light gray, entries of the Jacobian that are definitely zero are indicated, the rest is black. The center, upper and lower diagonal stem from the interaction of the same substances (u-u, v-v) and the side-bands of the interaction of different substances (u-v). The empty rows 1, 7, 8, 14 surrounding each of the two blocks are due to the implementation of the ghost cells that enforce constant boundary conditions.

For $N = 400$ the ratio of non-zero to all elements in the Jacobian of our simplified Brusselator is $\approx 6 \cdot 10^{-5}$, the larger our diffusive system, the sparser the Jacobian.

The central idea for efficient improvements is to use the sparsity of the Jacobian.

1. To combat the problem of the Jacobian being constructed column by column we can opt for a construction of the Jacobian where less Jacobian vector products have to be calculated
2. To combat the problem of the expensive LU decomposition, sparse decomposition techniques can be used (see Scott and Tůma, 2023, the explanation of those exceeds the scope of this work)

Let us note that there is a completely different method for solving the linear system $\underline{\underline{J}}_g\,\underline{a} = \underline{b}$ where the Jacobian is not constructed, and no matrix decomposition is done. These are Krylow subspace methods that solve the linear system iteratively based on Jacobian vector products. Notes on these can be found in the Julia notebook or in more depth in Gutknecht, 2007, and references therein.

An overview on the more efficient approaches is given in figure 22.

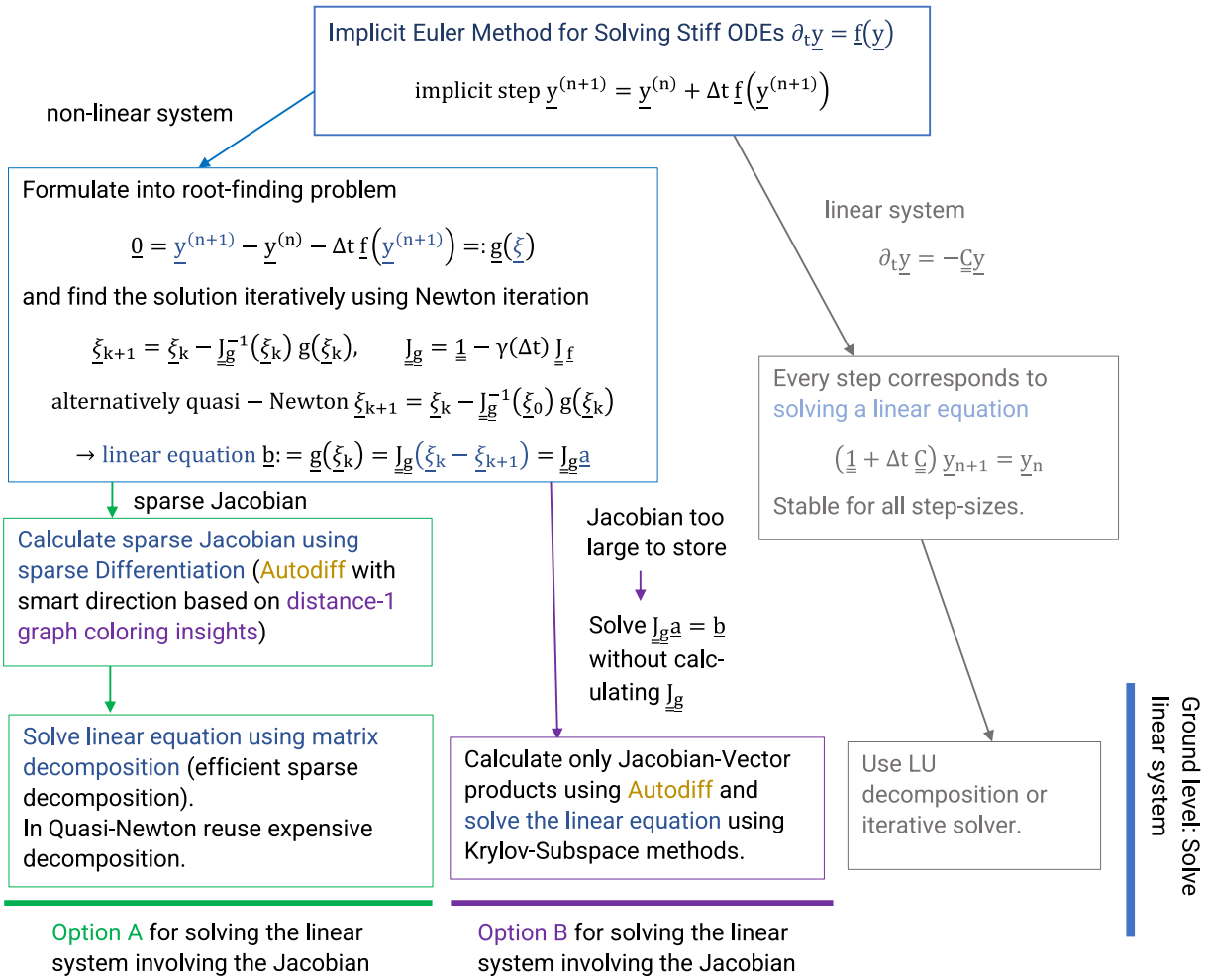


Figure 22: Flowchart of smart approaches to Implicit Euler

Smart construction of the Jacobian

Let us start by making the principle we use clear at a simple example following Rackauckas, Sciemon, et al., 2022, chapter 9.

The basic idea at the hand of a simple example

Consider the function

$$\underline{f}(\underline{x}) = \begin{pmatrix} x_1 + x_3 \\ x_2 x_3 \\ x_1 \end{pmatrix}$$

and notice that in no row of $\underline{f}$ both x_1 and x_2 appear, so there are definitely no cross effects of x_1 and x_2. Now consider the general form of the Jacobian

$$\underline{\underline{J}}_{\underline{f}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{pmatrix}$$

Only from knowing which variables occur in what entry of $\underline{f}$ we can deduce the sparsity pattern

$$\underline{\underline{S}} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

where the first and second column *do not overlap*, i.e. they share no 1 in the same row in the sparsity pattern which is due to x_1 and x_2 not appearing in the same row of $\underline{f}$.

Let us now consider the directional derivatives of $\underline{f}$ along $\underline{\hat{e}}_1$ and $\underline{\hat{e}}_2$.

$$\frac{\underline{f}(\underline{x} + \underline{\hat{e}}_1)}{h} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \quad \frac{\underline{f}(\underline{x} + \underline{\hat{e}}_2)}{h} = \begin{pmatrix} 0 \\ x_3 \\ 0 \end{pmatrix}$$

When taking the derivative along $\underline{\hat{e}}_1 + \underline{\hat{e}}_2$ we can see that the result is the sum of the two directional derivatives along $\underline{\hat{e}}_1$ and $\underline{\hat{e}}_2$.

$$\frac{\underline{f}(\underline{x} + \underline{\hat{e}}_1 + \underline{\hat{e}}_2)}{h} = \begin{pmatrix} 1 \\ x_3 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ x_3 \\ 0 \end{pmatrix}$$

Combined with the sparsity pattern from this one directional derivative we can calculate

two columns of the Jacobian.

$$\underline{\underline{J}}_{\underline{f}} = \begin{pmatrix} 1 & 0 & \vdots \\ 0 & x_3 & \vdots \\ 1 & 0 & \vdots \end{pmatrix}$$

From this example we can understand the more general scheme: Non-overlapping columns of the Jacobian can be calculated using one directional derivative, so only one forward automatic differentiation pass.

Application to the simplified Brusselator

For the simplified Brusselator it turns out that the whole Jacobian can be calculated in only four forward automatic differentiation passes. In figure 23 we can see the colored sparsity pattern of the Jacobian of the simplified Brusselator for $N = 10$ where columns of the same color do not overlap and can therefore be calculated simultaneously.

How can the coloring of the sparsity pattern be calculated?

It turns out that the coloring problem of our sparsity patterns is the distance-1 coloring problem of graph theory. We can turn the sparsity pattern into a graph by identifying the columns with nodes where two nodes are connected if the corresponding columns in the sparsity pattern overlap. For the simplified Brusselator, the graph is shown in figure 24.

The distance-1 graph coloring problem is to color the nodes of the graph so that no adjacent (directly by an edge connected) nodes are of the same color and a minimum number of colors is used. For a simple example graph this is shown in figure 25.

In general, distance-1 graph coloring is an NP-complete, i.e. not in polynomial time solvable, problem. There are, however, greedy algorithms that can find decent colorings, like

1. Generate a reverse-ordered list of the nodes by the node-degree, i.e. number of edges incident on a node
2. Go through this list (i.e. the highest degree first) and choose the smallest color index possible for every node

The resulting colored graph is shown in figure 26.

Also note that the coloring has to be calculated only once for a given problem, e. g. we can simulate our simplified Brusselator with thousands of initial conditions and still only have to calculate the coloring once.

Figure 23: Colored sparsity pattern of the Jacobian of the simplified Brusselator for $N = 10$. Colors indicate non-overlapping columns in the sparsity patterns, i.e. columns that do not have a 1 in the same row.

Improvements of the efficiency of the Implicit Euler scheme based on coloring the Jacobian and sparse factorization

Based on the concepts introduced above, the efficiency can be drastically improved as one can see in table 3.

Giving a differential equation solver more information is crucial

In many scenarios using the default automatic ODE solver from a package like DifferentialEquations.jl (Rackauckas and Nie, 2017) will also work nicely for stiff problems (automatic stiffness detection is used). However, giving something like the sparsity pattern to the solver beforehand can vastly benefit efficiency.

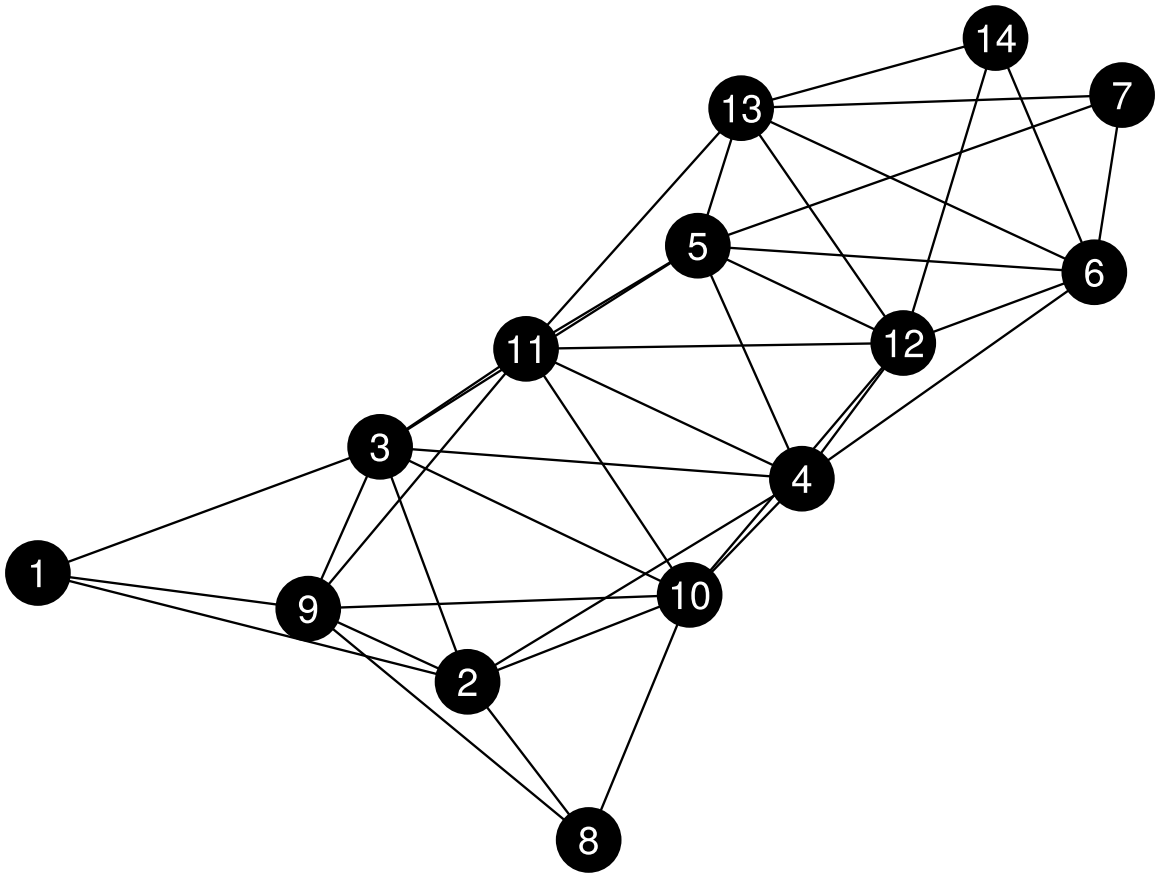


Figure 24: Graph representation of the sparsity pattern of the Jacobian of the simplified Brusselator for $N = 10$.

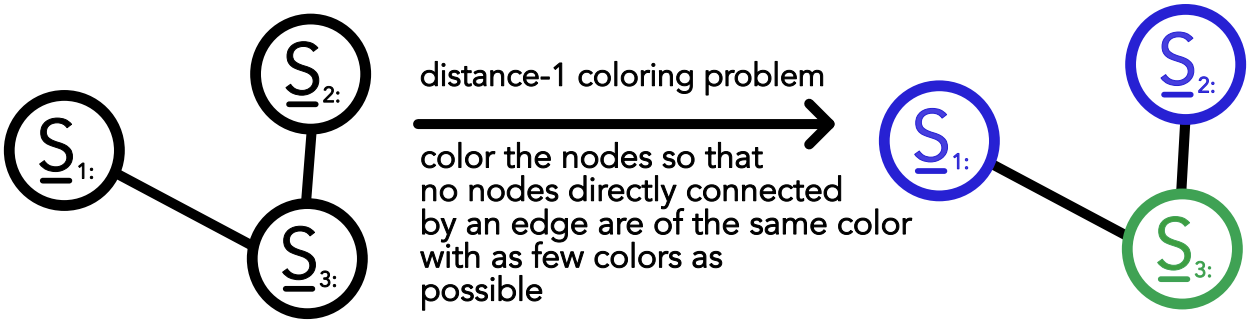


Figure 25: Simple distance-1 graph coloring example.

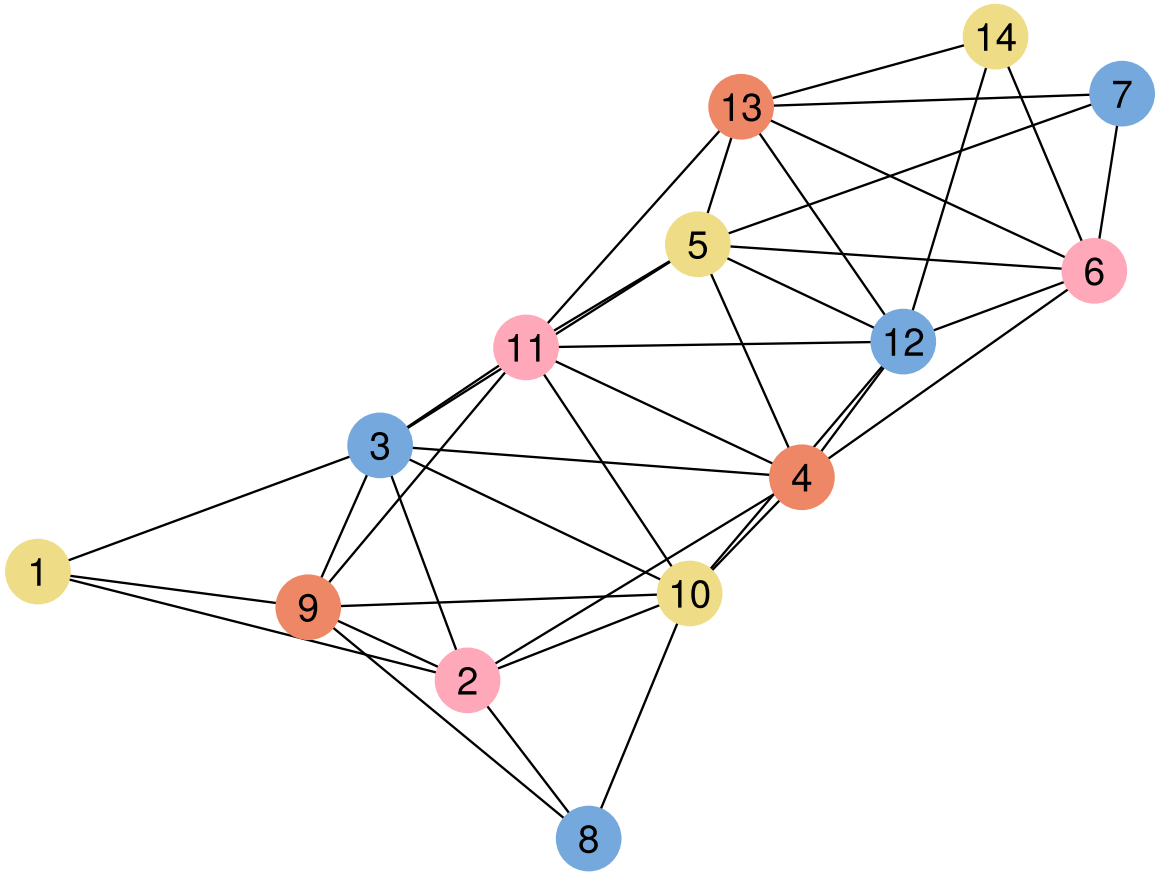


Figure 26: Graph as in figure 24 colored using a greedy distance-1 coloring.

Method	Δt	Performance
Implicit Euler; Quasi-Newton	0.1	3.144 s (197980 allocations: 5.57 GiB)
Implicit Euler; Quasi-Newton, coloring	0.1	877.994 ms (17404 allocations: 1.51 GiB)
Implicit Euler; Quasi-Newton, coloring, sparse factorization	0.1	88.977 ms (25192 allocations: 200.19 MiB)

Table 3: Performance of more efficient implementations of the Implicit Euler scheme. For $N = 400$ the simplified Brusselator is numerically solved for a timespan of 10.

Method	Performance
DifferentialEquations.jl solve method without sparsity pattern given	251.900 ms (63311 allocations: 211.75 MiB)
DifferentialEquations.jl solve method with sparsity pattern given	14.961 ms (33879 allocations: 27.41 MiB)

Table 4: For $N = 400$ the simplified Brusselator is numerically solved for a timespan of 10 using DifferentialEquations.jl

6.3 Outlook on more intricate methods and refinements

We have now finished our extensive discussion of the Implicit Euler method for solving stiff differential equations. In practical applications one would resort to higher order methods like Rosenbrock or semi-implicit extrapolation methods (Press et al., 2007, chapter 17.5.1f) and also use adaptive time-stepping (see e.g. Rackauckas, Sciemon, et al., 2022, chapter 9).

Solving the linear system can be additionally sped up using the technique of preconditioning as done in Delis et al., 2022 and also described in Rackauckas, Sciemon, et al., 2022, chapter 9.

7 Breakdown of Classical Methods and Outlook

While we have employed forward automatic differentiation from the area of machine learning we have in our overall schemes remained in the realm of classical numerical methods. Nonetheless, these traditional numerical approaches have inherent limitations, two of which deserve specific attention.

In our illustrative example, we have chosen to model a simple diffusive system. However, the focus often shifts towards turbulent systems characterized by high Reynolds numbers, such as the airflow around aircraft wings. The length-scale our simulation would need to resolve becomes too small to be practically resolved. One idea there it to use insights from machine learning allowing for a coarser grid. The problem of turbulent simulations and a machine learning approach are discussed in Kochkov et al., 2021.

Another challenge is handling partial differential equations of high dimensionality. Many immensely significant differential equations, such as the Schrödinger equation for multi-particle systems or the Black-Scholes equation in finance, manifest as exceedingly high-dimensional partial differential equations. The computational expense of solving such equations rises exponentially with the dimensionality (Han et al., 2018). In this realm as well, deep learning techniques are tested as in Han et al., 2018.

These are only two examples of many problems faced in the realm of computationally solving differential equation. In a broader context, the need for inverse modeling and blending differential equations with vast data sets would also necessitate a more in-depth discussion. However, we reserve these topics for forthcoming Scientific Machine Learning seminar talks.

References

Chou, C et al. (2007). »Numerical methods for stiff reaction-diffusion systems«. In: *DISCRETE AND CONTINUOUS DYNAMICAL SYSTEMS SERIES B* 7.3, page 515.

Courant, R., K. Friedrichs, and H. Lewy (Dec. 1928). »Über die partiellen Differenzengleichungen der mathematischen Physik«. In: *Mathematische Annalen* 100.1, pages 32–74. DOI: 10.1007/BF01448839. URL: https://doi.org/10.1007/BF01448839.

Delis, Anargiros I., Maria Kazolea, and Maria Gaitani (2022). »On the Numerical Solution of Sparse Linear Systems Emerging in Finite Volume Discretizations of 2D Boussinesq-Type Models on Unstructured Grids«. In: *Water* 14.21. DOI: 10.3390/w14213584. URL: https://www.mdpi.com/2073-4441/14/21/3584.

Gebremedhin, Assefaw Hadish, Fredrik Manne, and Alex Pothen (2005). »What Color Is Your Jacobian? Graph Coloring for Computing Derivatives«. In: *SIAM Review* 47.4, pages 629–705. DOI: 10.1137/S0036144504444711. URL: https://doi.org/10.1137/S0036144504444711.

Gutknecht, Martin H (2007). »A brief introduction to Krylov space methods for solving linear systems«. In: *Frontiers of Computational Science: Proceedings of the International Symposium on Frontiers of Computational Science 2005*. Springer, pages 53–62.

Hairer, Ernst (June 2006). »Long-time energy conservation of numerical integrators«. In: *Lecture Notes Ser. FoCM Santander 2005* 331. DOI: 10.1017/CBO9780511721571.005.

Hairer, Ernst, Christian Lubich, and Gerhard Wanner (2003). »Geometric numerical integration illustrated by the Störmer–Verlet method«. In: *Acta Numerica* 12, pages 399–450. DOI: 10.1017/S0962492902000144.

Hairer, Ernst and Gerhard Wanner (1996). *Solving Ordinary Differential Equations II*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-05221-7. URL: https://doi.org/10.1007/978-3-642-05221-7.

Hairer, Ernst, Gerhard Wanner, and Christian Lubich (2006). *Geometric Numerical Integration*. Springer-Verlag. DOI: 10.1007/3-540-30666-8. URL: https://doi.org/10.1007/3-540-30666-8.

Hairer, Ernst, Gerhard Wanner, and Syvert P. Nørsett (1993). *Solving Ordinary Differential Equations I*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-540-78862-1. URL: https://doi.org/10.1007/978-3-540-78862-1.

Han, Jiequn, Arnulf Jentzen, and Weinan E (2018). »Solving high-dimensional partial differential equations using deep learning«. In: *Proceedings of the National Academy of Sciences* 115.34, pages 8505–8510. DOI: 10.1073/pnas.1718942115. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1718942115.

Heiter, Pascal Frederik (2012). »On Numerical Methods for Stiff Ordinary Differential Equation Systems«. Master's thesis. Ulm University. URL: https://www.uni-ulm.de/fileadmin/website_uni_ulm/mawi.inst.070/abschlussarbeiten/masterthesis_pfh.pdf.

Kochkov, Dmitrii et al. (May 2021). »Machine learning-accelerated computational fluid dynamics«. en. In: *Proc Natl Acad Sci U S A* 118.21.

Lambert, J.D. (1991). *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem*. Wiley. URL: https://books.google.de/books?id=P0vPnQEACAAJ.

Lefever, R. and G. Nicolis (1971). »Chemical instabilities and sustained oscillations«. In: *Journal of Theoretical Biology* 30.2, pages 267–284. DOI: https://doi.org/10.1016/0022-5193(71)90054-3. URL: https://www.sciencedirect.com/science/article/pii/0022519371900543.

Moser, Jürgen (1978). »Is the solar system stable?« In: *The Mathematical Intelligencer* 1.2, pages 65–71. DOI: 10.1007/BF03023062. URL: https://doi.org/10.1007/BF03023062.

Press, William H. et al. (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3rd edition. USA: Cambridge University Press.

Rackauckas, Christopher and Qing Nie (2017). »Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia«. In: *Journal of Open Research Software* 5.1, page 15.

Rackauckas, Christopher, Sciemon, et al. (Nov. 2022). *SciML/SciMLBook: v1.1*. Version v1.1. DOI: 10.5281/zenodo.7347643. URL: https://doi.org/10.5281/zenodo.7347643.

Scott, Jennifer and Miroslav Tůma (2023). *Algorithms for Sparse Linear Systems*. Springer International Publishing. DOI: 10.1007/978-3-031-25820-6. URL: https://doi.org/10.1007/978-3-031-25820-6.

Springel, Volker et al. (2023). *Lecture notes in Fundamentals of Simulation Methods*.

Tsitouras, Ch. (2011). »Runge–Kutta pairs of order 5(4) satisfying only the first column simplifying assumption«. In: *Computers and Mathematics with Applications* 62.2, pages 770–775. DOI: https://doi.org/10.1016/j.camwa.2011.06.002. URL: https://www.sciencedirect.com/science/article/pii/S0898122111004706.