An Overview of Rosenbrock-Krylov Methods for the Numerical Solution of Ordinary Differential Equations

Jens Axelsson

Bachelor's thesis
2021:K24

Lund University

Faculty of Science
Centre for Mathematical Sciences
Numerical Analysis

Centrum Scientiarum Mathematicarum

Populärvetenskaplig sammanfattning

En differentialekvation beskriver en funktion där förändringshastigheten beror på funktionsvärdet. Som ett exempel kan nämnas befolkningstillväxt, där tillväxthastigheten (i form av födslar) kommer att vara högre för en större befolkning. Numerisk lösning av en differentialekvation innebär att givet ett visst startvärde förutsäga funktionsvärdet vid en senare tidpunkt med tillräckligt god noggrannhet.

Det finns flera olika kategorier av metoder för numerisk lösning av differentialekvationer, varav en av de mest spridda och använda är så kallade explicita Runge-Kutta-metoder. Dessa utgår från det kända startvärdet, tar små steg framåt i tiden och beräknar approximationer av lösningen för varje tidssteg. För somliga problem, kända som styva problem, har det visat sig att explicita metoder fungerar otillfredsställande. Sådana problem kräver implicita metoder, vilket Rosenbrockmetoder är ett exempel på.

Rosenbrockmetoder använder funktionens Jacobian, som beskriver förändringen, för att approximera varje nytt värde. För ett problem med N ekvationer är Jacobianen av storlek $N \times N$, så mängden beräkningar växer snabbt med problemets storlek. Rosenbrock-Krylov-metoder är också implicita metoder och fungerar mycket snarlikt Rosenbrockmetoder, men ersätter Jacobianen med en approximation som är avsevärt mindre. Detta minskar mängden beräkningar som behöver göras när varje nytt värde beräknas, men i gengäld kräver det viss beräkningskraft att ta fram approximationen av Jacobianen.

I den här uppsatsen implementeras och jämförs två explicita Runge-Kuttametoder, en vanlig Rosenbrockmetod och två Rosenbrock-Krylov-metoder. De testade Rosenbrock-Krylov-metoderna uppvisar likartad noggrannhet per tidssteg jämfört med Rosenbrockmetoder, men med avsevärt kortare exekveringstid för stora system av differentialekvationer. Jämfört med de explicita Runge-Kuttametoderna presterar Rosenbrock-Krylov-metoderna, som förväntat, mycket bättre på styva problem.

Abstract

This thesis offers an overview of the relatively new family of Rosenbrock-Krylov numerical methods for ODEs. These methods are a further development of Rosenbrock methods, using a lower-dimension approximation of the Jacobian. The thesis gives the mathematical background to Rosenbrock-Krylov methods by first presenting Runge-Kutta methods and Rosenbrock methods, followed by the concept of Krylov spaces and the Arnoldi Iteration. Rosenbrock-Krylov methods are described in relation to these concepts.

The implementation of the three mentioned families of methods is chronicled in the thesis, and their numerical performance on some problems is studied. Rosenbrock-Krylov methods show promising results for accuracy of the solution for fixed step sizes. Regarding raw computing performance, the implemention of Rosenbrock-Krylov methods made for this thesis is considerably faster than an ordinary Rosenbrock method for larger problems. As such, Rosenbrock-Krylov methods appear to be an appealing alternative to current methods.

Contents

1 Introduction

An initial value problem taking the form of a system of ordinary differential equations (ODEs) can be given by

$$\boldsymbol{y}' = \boldsymbol{f}\left(t, \boldsymbol{y}(t)\right), \quad \boldsymbol{y}(0) = \boldsymbol{y}_0, \quad t > 0, \quad \boldsymbol{y}(t), \boldsymbol{f}(t, \boldsymbol{y}) \in \mathbb{R}^N. \tag{1}$$

A *solution* to this problem is a function $\boldsymbol{y}(t)$ fulfilling these conditions. For certain problems, the solution can be found quite easily by hand, but for many problems the solution is hard or even impossible to find. In these cases, it is necessary to fall back on numerical methods that approximate $\boldsymbol{y}(\tilde{t})$ at some desired $\tilde{t}$. Even though this technically is not a solution to the differential equation, finding these approximations will be referred to as solving the differential equation when used in the context of a numerical method.

ODEs arise in models within diverse subjects, such as the laws of motion, chemical reaction rates and the modelling of populations. Furthermore, many problems within areas such as fluid dynamics and material sciences, can be modelled by partial differential equations, which have more than one independent variable. One standard strategy for solving such equations is through discretization, which results in systems of ODEs. Thus, it should be clear that finding efficient and reliable numerical methods for ODEs is an important area of numerical research.

Certain models incorporate processes with rates-of-change occurring on very different time scales. For instance, one could imagine a chemical reaction in which one compound slowly decays to some other compound (with a reaction time measured in, for instance, hours), and the resulting compound reacts violently with the environment (with a reaction time measured in, for example, milliseconds). The computationally cheapest numerical methods may struggle with giving correct answers for these kinds of problems, in which case they are referred to as *stiff* problems. This motivates the usage of so-called *implicit* methods, which are computationally more expensive but handle stiff problems better.

A rather recent development in the field is the family of Rosenbrock-Krylov methods, which is a further development of the implicit Rosenbrock methods, aiming to reduce the computational cost for large systems. This thesis gives a mathematical background for the different ideas leading to the development of this family of methods, starting from simpler methods and successively increasing in sophistication. Furthermore, it chronicles the process of implementing these methods and presents some results of applying them on a few select problems.

2 Background

One of the earliest methods for the numerical solution of ODEs was developed by Leonhard Euler in the 18th century. The method, known as the explicit Euler method, solves the problem (1) by calculating $\boldsymbol{y}'(0)$ (which can be done since

$\boldsymbol{y}(0)$ is given), taking a small step of size h in t and approximating $\boldsymbol{y}(h) \approx \boldsymbol{y}_0 + h\boldsymbol{y}'(0) = \boldsymbol{y}_1$. This is repeated until the desired t is reached, so that each successive approximation $\boldsymbol{y}_n$ is given by the previous according to $\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + h\boldsymbol{y}'_n$.

Broadly speaking, there are two categories of numerical methods in use for solving ODEs: One-step methods and linear multistep methods. Both could be considered to be extensions of the explicit Euler method. One-step methods, much like the explicit Euler method, only uses the most recently calculated solution value $\boldsymbol{y}_{n-1}$ to approximate $\boldsymbol{y}_n$, while linear multistep methods take several previous approximations into account [Sha94, p. 138]. All of the methods discussed in this thesis belong to the category of one-step methods.

Runge-Kutta methods for solving ODEs numerically were developed by Carl Runge and Martin Kutta around the turn of the 20th century. Runge proposed to extend the Euler method with several function evaluations per step and Kutta developed the most widely used version of the method [But87, p. 128].

In the early 1960s, two different modifications of the Runge-Kutta methods were introduced, both of which happen to be relevant to the subject of this thesis: J. Kuntzmann (1961) and J.C. Butcher (1964) suggested the so-called implicit Runge-Kutta methods [But96], which would prove to have more favorable stability properties than earlier methods, but at a higher computational cost. In a 1963 paper, H. H. Rosenbrock suggested using the partial derivatives of $\boldsymbol{f}$ to approximate the solutions of this nonlinear system directly in the method [Ros63], which marked the birth of the family of so-called Rosenbrock methods. These families of methods are introduced in section 3.

Ordinary differential equations can be categorized into *autonomous* and *non-autonomous*. The difference being that autonomous ODEs can be written $\boldsymbol{y}' = \boldsymbol{f}(\boldsymbol{y})$, while non-autonomous equations are of the form $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$. That is, for non-autonomous ODEs the derivative depends on t directly. Non-autonomous problems can be solved by defining

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{y} \\ t \end{bmatrix}, \quad \boldsymbol{g}(\boldsymbol{z}) = \begin{bmatrix} \boldsymbol{y}' \\ 1 \end{bmatrix} \tag{2}$$

which results in the (autonomous) problem $\boldsymbol{z}' = \boldsymbol{g}(\boldsymbol{z})$. For this reason and to improve readability, $\boldsymbol{f}(\boldsymbol{y})$ rather than $\boldsymbol{f}(t, \boldsymbol{y})$ will be used throughout this thesis, except in cases where it is desirable to emphasize that $\boldsymbol{f}$ is non-autonomous.

Finally, a concept, which is one of the motivations behind developing more "advanced" methods than the explicit Euler method is *order of convergence*. For a method of order p, the error asymptotically behaves as $error = Ch^p$, where C is a method constant. That is, when the step size h decreases, the higher the order of the method, the more the error can shrink. As an example, the explicit Euler method is a first order method, meaning that $p = 1$ and halving h halves the error (roughly). Higher order methods require more calculations per time step, but since the error shrinks faster, bigger step sizes are possible without sacrificing precision.

3 Numerical Methods for ODEs

This section will present the above mentioned families of Runge-Kutta and Rosenbrock methods and in which ways they are similar and differ for solving ODEs.

3.1 Runge-Kutta Methods

When solving the problem (1), an s-stage Runge-Kutta method is any method which given a known $\boldsymbol{y}_{n-1} \approx \boldsymbol{y}(t_{n-1})$ calculates $\boldsymbol{y}_n \approx \boldsymbol{y}(t_n)$ where $t_n = t_{n-1}+h$ by

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + h \sum_{i=1}^{s} b_i \boldsymbol{k}_i \tag{3}$$

where

$$\boldsymbol{k}_i = \boldsymbol{f}\Big(\boldsymbol{y}_{n-1} + h \sum_{j=1}^{s} a_{ij} \boldsymbol{k}_j\Big) \tag{4}$$

and $a_{11} \dots a_{ss}, b_1 \dots b_s$ are coefficients which define the specific method in question.[1] Applying the method to (2) gives

$$\begin{bmatrix} \boldsymbol{y}_n \\ t_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}_{n-1} \\ t_{n-1} \end{bmatrix} + h \sum_{i=1}^{s} b_i \boldsymbol{k}_i \tag{5a}$$

$$\boldsymbol{k}_i = \boldsymbol{g}\left(\begin{bmatrix} \boldsymbol{y}_{n-1} \\ t_{n-1} \end{bmatrix} + h \sum_{j=1}^{s} a_{ij} \boldsymbol{k}_j \right). \tag{5b}$$

From (5b) and the definition of $\boldsymbol{g}$, it can be deduced that each $\boldsymbol{k}_i$ will have 1 as its $(N+1)$st element. This means that the final row of (5a) will read

$$t_n = t_{n-1} + h \sum_{i=1}^{s} b_i$$

and since $t_n = t_{n-1} + h$ by definition, it must follow that $\sum b_i = 1$. Apart from this, the a_{ij}'s and b_i's are chosen such that they fulfill conditions that make sure the method is of the desired order [But87, ch. 22]. Oftentimes, the coefficients of a method are presented in a *Butcher tableau* of the form

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array}$$

where the c coefficients are defined as $c_i = \sum_{j=1}^{s} a_{ij}$ and appear in forms of the method written explicitly for the non-autonomous case.

[1] The reader who has previously encountered Runge-Kutta methods might have seen them defined as $y_n = y_{n-1} + h \sum_{i=1}^{s} b_i f(Y_i)$ where $Y_i = y_{n-1} + h \sum_{j=1}^{s} a_{ij} f(Y_j)$. These definitions

A method having more than one stage corresponds to calculating the slope (using the one-variable terminology) at several points between y_{n-1} and y_n, given by the a_{ij}'s, and summing them after giving each of them a specific "weight" with the b_i's (recall that $\sum b_i = 1$).

3.1.1 Explicit Runge-Kutta methods

Had this section been presented in a chronological manner, the definition of a Runge-Kutta method would have been different. This is because all of the original Runge-Kutta methods are what is nowadays called *explicit* Runge-Kutta methods.

This class of methods is characterized by the fact that the $\boldsymbol{k}_i$'s can be calculated recursively and are given *explicitly* from the previously calculated $\boldsymbol{k}_i$'s. That is, (4) can be written

$$\boldsymbol{k}_i = \boldsymbol{f}\Big(\boldsymbol{y}_{n-1} + h\sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j\Big).$$

In the more general definition of a Runge-Kutta method, this corresponds to $a_{ij} = 0$ for $j \geq i$, that is every a_{ij} on or above the diagonal of the Butcher tableau is zero (and often omitted from print).

The simplest example of an explicit Runge-Kutta method (albeit conceived long before either Runge or Kutta) is the explicit Euler method mentioned earlier. It is a one-stage method with the very unassuming Butcher tableau

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

One of the variants developed by Kutta (in 1901) has become so ubiquitous that it is referred to as the *classical* Runge-Kutta method. It is an explicit four-stage fourth order method with the coefficients

$$\begin{array}{c|cccc} 0 & & & & \\ 0.5 & 0.5 & & & \\ 0.5 & 0 & 0.5 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Since only three a coefficients are non-zero, this method requires very few computations per step (which was an even bigger advantage at the time since the calculations were done by hand!) and has proven to be a mostly reliable way of approximating the solutions of differential equations. Because of its four stages, this method is also referred to as RK4.

RK4 is also a fairly clear example method from a pedagogical point-of-view. In Figure 1, the process for finding the k_i values in a one variable case is represented

are equivalent, but the one used in this text is more reminiscent of the Rosenbrock methods which will be introduced later.

graphically. Note that since there is only one a coefficient in each row of the Butcher tableau, each k_i is given from just the immediately previous one by $k_i = f(y_{n-1} + ha_{i,i-1}k_{i-1})$. That is, by "moving" $ha_{i,i-1}$ from $(0, y_0)$ in the t-direction at a "slope" k_{i-1} (in the figure, most obviously visible for k_2 given by k_1).

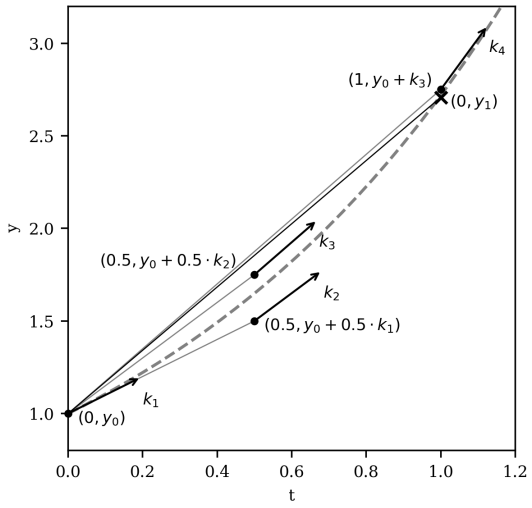

FIGURE 1: *RK4 used to approximate the solution of $y' = y$ at $t = 1$, taking one step from $t_0 = 0$. The four k_i values are represented by arrows showing the slope at the point from which the k_i value is calculated. Corresponding slopes originating from $y(0)$ are displayed with thin, gray lines. Final approximation of $y(1) \approx y_0 + h \sum b_i k_i$ is marked with an X. Exact solution is shown as a dashed, gray line.*

3.1.2 A NOTE ON STABILITY AND STIFFNESS

If RK4 is such a computationally cheap method and has such a proven track record, why bother with the upper triangle of the a coefficient matrix?

The answer to the above question is the existence of *stiff* problems. Though there is no water-tight mathematical definition of exactly what constitutes a stiff initial value problem, what they all have in common is that explicit methods perform poorly when used for solving them [HW10, p. 1]. This becomes a problem when the step sizes required for stability becomes so small that the method in question becomes too computationally expensive to be considered practical.

To get an idea of the stability properties of a method, one can look at its stability region. In order to describe the stability region, it is first necessary to define the stability function of a method. The problem

$$y' = \lambda y, \quad y(0) = y_0, \quad \lambda \in \mathbb{C} \tag{6}$$

is called the test equation. Applying one iteration of some method with step size h to (6) results in an expression $y_1 = R(h\lambda)y_0$, where $R(h\lambda)$ is called the stability function of the method. The stability region of the method is then defined as the region in which $|R(h\lambda)| < 1$.

As an example, consider the explicit Euler method. Applying this method to $y' = \lambda y$ gives $y_1 = y_0 + h\lambda y_0 = (1 + h\lambda)y_0$. Thus, the stability function is $R(h\lambda) = (1 + h\lambda)$ and $(1 + h\lambda)^n \to 0$ as $n \to \infty$ if and only if $|R(h\lambda)| = |1 + h\lambda| < 1$. The stability region for the explicit Euler method becomes the unit disk centered around -1.

Thus, it can be seen that for large $|\lambda|$ it is necessary to decrease the step size h in order to remain within the stability region.

3.1.3 Implicit Runge-Kutta methods

An implicit Runge-Kutta method is a method where some nonzero a_{ij} with $j \geq i$ is taken into account. This means that the $\boldsymbol{k}_i$s are given *implicitly* by their mutual relations, and a system of non-linear equations must be solved to find their values. Since this is done in a numerical context, the solution to the system is generally approximated using some iterative method, which in itself may require quite a lot of computations.

Despite being significantly more computationally expensive per step, implicit methods can be an attractive choice. This is because of their more favourable stability properties, often being able to be applied to stiff problems without having to limit the step size in the way required by explicit methods. The variation of the Euler method, known as the implicit Euler method, where y_n is given by $y_n = y_{n-1} + hf(y_n)$ is an example of an implicit method. Just like the explicit Euler method, this is a one-stage Runge-Kutta method. Its Butcher tableau is

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

and it can be seen that the only formal difference is that the single a coefficient is 1 instead of 0.

Applying the implicit Euler method to the test equation (6) gives $y_n = y_{n-1} + h\lambda y_n \iff y_n = \frac{1}{1-h\lambda} y_{n-1}$. The stability function of the method is $R(h\lambda) = \frac{1}{1-h\lambda}$ and thus $|R(h\lambda)| < 1$ if and only if $h\lambda$ is outside the unit disk centered around 1. Compared to the explicit Euler method, it is clear that the step size is not nearly as limited by the λ. The stability regions of these two methods are visualized in Figure 2.

The implicit Euler method is what is called an *A-stable* method, which is a method with a stability region that contains all complex numbers with negative real part. This type of stability is special in the sense that the solution to (6), which is $y = y_0 e^{\lambda t} \to 0$ as $t \to \infty$ for any λ with $\mathrm{Re}(\lambda) < 0$, and the same will hold true for any A-stable numerical method. If it furthermore holds true that $|R(h\lambda)| \to 0$

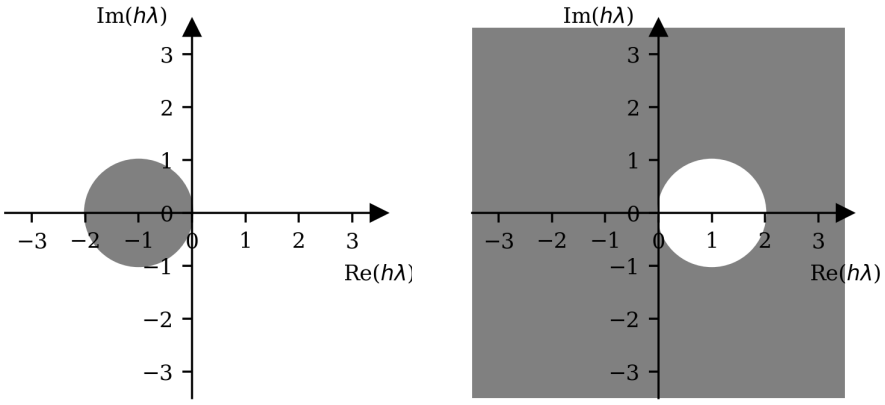

FIGURE 2: *Stability regions of the explicit Euler method (left) and the implicit Euler method (right) highlighted in gray.*

as $h\lambda \to \infty$, the method is called *L-stable* and will typically perform better on stiff problems [HW10, pp. 44–45].

3.2 ROSENBROCK METHODS

Rosenbrock methods are a family of implicit numerical methods for ODEs, suggested by H. H. Rosenbrock in a 1963 article [Ros63]. They can be derived as follows: Consider a *semi-implicit* Runge-Kutta method, meaning that it only has coefficients $a_{ij} \neq 0$ for $i \leq j$. Starting from (4), the definition of $\boldsymbol{k}_i$ in the Runge-Kutta method, and letting $j = 1, \ldots, i$ results in

$$\boldsymbol{k}_i = \boldsymbol{f}\Big(\boldsymbol{y}_{n-1} + h\sum_{j=1}^{i} a_{ij}\boldsymbol{k}_j\Big) = \boldsymbol{f}\Big(\boldsymbol{y}_{n-1} + h\sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j + ha_{ii}\boldsymbol{k}_i\Big).$$

Letting $\boldsymbol{g}_i = \boldsymbol{y}_{n-1} + h\sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j$ (for brevity) and doing the linearization

$$\boldsymbol{f}(\boldsymbol{g}_i + ha_{ii}\boldsymbol{k}_i) \approx \boldsymbol{f}(\boldsymbol{g}_i) + \boldsymbol{f}'(\boldsymbol{g}_i) \cdot a_{ii}\boldsymbol{k}_i$$

leads to the expression

$$\boldsymbol{k}_i = \boldsymbol{f}(\boldsymbol{g}_i) + \boldsymbol{f}'(\boldsymbol{g}_i) \cdot a_{ii}\boldsymbol{k}_i, \quad i = 1, \ldots, s \tag{7}$$

from which the $\boldsymbol{k}_i$'s can be determined recursively.

To arrive at the final form of Rosenbrock methods, two additional modifications are done to (7): First, $\boldsymbol{f}'(\boldsymbol{g}_i)$ is replaced by the Jacobian matrix given by

$$J \in \mathbb{R}^{N\times N}, \quad J_{ij} = \frac{\partial f_i}{\partial y_j}(\boldsymbol{y}_{n-1}). \tag{8}$$

This is not unreasonable for small enough h; note that $J\boldsymbol{g}_1 = \boldsymbol{f}'(\boldsymbol{g}_1)$. Secondly, it has been shown to be beneficial to include additional coefficients (called γ in the final definition of the method) that define additional linear relations between the $\boldsymbol{k}_i$'s [HW10, p. 103].

The final definition of an s stage Rosenbrock method is given by

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \sum_{i=1}^{s} b_i \boldsymbol{k}_i \tag{9}$$

where

$$\boldsymbol{k}_i = h\boldsymbol{f}\Big(\boldsymbol{y}_{n-1} + \sum_{j=1}^{i-1} \alpha_{ij}\boldsymbol{k}_j\Big) + hJ\sum_{j=1}^{i} \gamma_{ij}\boldsymbol{k}_j\,, \tag{10}$$

α_{ij}, b_i and γ_{ij} are coefficients which define the specific method and J is the Jacobian defined in (8).

As can be seen, this is still a semi-implicit method, which has the consequence that each $\boldsymbol{k}_i$ can be found recursively by solving a linear system of equations. In this case, each $\boldsymbol{k}_i$ is given by rewriting (10) as

$$(I - h\gamma_{ii}J)\boldsymbol{k}_i = h\boldsymbol{f}(\boldsymbol{y}_{n-1} + \sum_{j=1}^{i-1} \alpha_{ij}\boldsymbol{k}_j) + hJ\sum_{j=1}^{i-1} \gamma_{ij}\boldsymbol{k}_j, \tag{11}$$

which is a linear system of equations for the elements of $\boldsymbol{k}_i$.

The α, b and γ coefficients need to be chosen in such a way that they fulfill order conditions for the desired order of convergence in a similar way to Runge-Kutta methods. Most often, the coefficients are chosen such that $\gamma_{11} = \gamma_{22} = \cdots = \gamma_{ss}$, which gives the same left-hand side in (11) for each of the $\boldsymbol{k}_i$'s and thus saves some calculations.

A later development of the method are the so-called Rosenbrock-W (Rosenbrock-Wolfbrandt) methods. These have additional order conditions, that guarantee that the method will converge for any approximation $A \neq 0$ of the Jacobian. The idea being that a computed Jacobian can be used over several consecutive steps, thereby saving computations [HW10, pp. 114–116]. The remaining thesis will not focus on Rosenbrock-W methods, but they are mentioned to highlight that it is possible for Rosenbrock methods to converge even without the exact Jacobian.

4 Krylov subspaces and the Arnoldi iteration

The idea behind Rosenbrock-Krylov methods is to substitute the exact Jacobian J in the Rosenbrock method with a lower dimension approximation. The intent is to approximate the solution of (11), thereby avoiding having to solve the $N \times N$ system of equations. For large problems, this could save a substantial amount of computations. The lower dimension approximation of the Jacobian is obtained

using Krylov subspaces and the Arnoldi iteration, both of which are presented in this section.

Starting from a square $N \times N$ matrix A and an N element vector $\boldsymbol{b}$, the M-dimensional *Krylov subspace* generated by A and $\boldsymbol{b}$ is the subspace spanned by $\{\boldsymbol{b}, A\boldsymbol{b}, A^2\boldsymbol{b}, \ldots, A^{M-1}\boldsymbol{b}\}$. This space is generally denoted $\mathcal{K}_M$ (or $\mathcal{K}_M(A, \boldsymbol{b})$ if there can be ambiguity). In the best of worlds, there is some M much smaller than the dimension of A such that span $\mathcal{K}_M$ = span A, in which case A can be substituted by a lower dimension matrix without loss of precision. In the Rosenbrock-Krylov methods, even in the case where there is no such luck, the process is stopped after M iterations, where $M \ll N$, and that subspace is used instead of the full space of the Jacobian.

4.1 The Arnoldi Iteration

The Arnoldi iteration was suggested by W. E. Arnoldi in 1951, and was originally intended for finding approximations of eigenvalues of large matrices [Arn51]. It does this by constructing an orthonormal basis for a Krylov subspace, and it is for that purpose that it is employed in the case of Rosenbrock-Krylov methods. The iteration is given in Algorithm 1.

Algorithm 1: The Arnoldi iteration

Data: A: $N \times N$ matrix
$\boldsymbol{b}$: N element vector

$\boldsymbol{v}_1 = \boldsymbol{b}/||\boldsymbol{b}||$
for $i = 2, \ldots, M+1$ **do**
  $\boldsymbol{v}_i = A\boldsymbol{v}_{i-1}$
  $c = ||\boldsymbol{v}_i||$
  for $j = 1, \ldots, i-1$ **do**
    $H_{j,i-1} = (\boldsymbol{v}_j, \boldsymbol{v}_i)$
    $\boldsymbol{v}_i = \boldsymbol{v}_i - H_{j,i-1}\boldsymbol{v}_j$
  if $||\boldsymbol{v}_i||/c < \kappa$ **then**
    for $j = i, \ldots, i-1$ **do**
      $\rho = (\boldsymbol{v}_j, \boldsymbol{v}_i)$
      $\boldsymbol{v}_i = \boldsymbol{v}_i - \rho\boldsymbol{v}_j$
      $H_{j,i-1} = H_{j,i-1} - \rho$
  if $||\boldsymbol{v}_i|| = 0$ **then**
    `break`
  $H_{i,i-1} = \boldsymbol{v}_i/||\boldsymbol{v}_i||$

Deriving the iteration starts from the simliarity transformation

$$AV = VH, \quad A, V, H \in \mathbb{R}^{N \times N} \tag{12}$$

where V is orthogonal and H is an upper Hessenberg matrix, that is all entries below the first subdiagonal are zero. However, A is assumed to be so large as to

make it unfeasible to calculate the entire reduction, so instead only the first m columns are considered.

The first m columns of AV will be determined by the entirety of A but only the first m columns of V, and as such the left hand side of (12) can be written AV_m, where V_m is the $N \times m$ matrix consisting of the first m columns of V.

Next, looking at the right hand side, the entirety of V but only the first m columns of H is needed. But this is where the fact that H is an upper Hessenberg matrix comes into play: Since it is known that every element below the first subdiagonal is zero, all the relevant information in the first m columns of H is contained within the $(m+1) \times m$ submatrix consisting of the intersection of the first $m+1$ rows and the first m columns of H. Thus, it follows that every column past the $(m+1)$st of V is irrelevant and the resulting expression becomes

$$AV_m = V_{m+1}\hat{H}_m \tag{13}$$

where $\hat{H}_m$ is the submatrix of H which was described above. This relation is shown graphically in Figure 3.

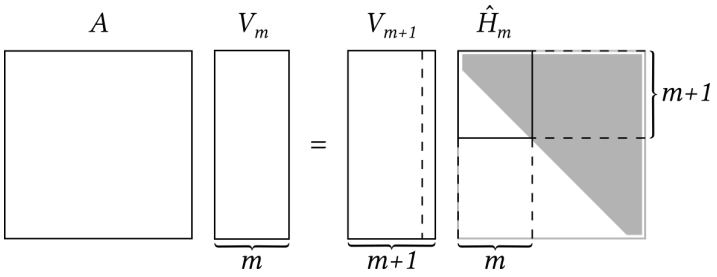

FIGURE 3: *The relation $AV_m = V_{m+1}\hat{H}_m$, with the entirety of H shown in gray.*

Looking at only the final column of (13) establishes a relationship between $\boldsymbol{v}_{m+1}$, the $m+1$st column of V, and all the previous columns of V in the form of $A\boldsymbol{v}_m = V_{m+1}\boldsymbol{h}_m$, where $\boldsymbol{h}_m$ is the mth column of $\hat{H}_m$. More explicitly, this can be expressed as

$$h_{(m+1)_m}\boldsymbol{v}_{m+1} = A\boldsymbol{v}_m - \boldsymbol{v}_1 h_{1_m} - \cdots - \boldsymbol{v}_m h_{m_m}. \tag{14}$$

What remains is orthogonalizing $h_{m+1,m}\boldsymbol{v}_{m+1}$ against all the previous $\boldsymbol{v}_i$'s, using the Gram-Schmidt process, which also finds the correct h_{i_m}'s. In Algorithm 1 above, this is done on lines 5 – 7. Note that this algorithm uses the so-called *modified* Gram-Schmidt process,[2]which is mathematically equivalent to the ordinary Gram-Schmidt process but numerically stable [Saa03, p. 162]. Finally $h_{(m+1)_m}$ is chosen such that $||\boldsymbol{v}_{m+1}|| = 1$, which in Algorithm 1 is done on line 15. This concludes one step of the iteration.

[2]In the ordinary Gram-Schmidt process, each "new" vector $\boldsymbol{v}_i$ is orthogonalized against the previous $\boldsymbol{v}_j$, $j = 1, \ldots, i-1$, by subtracting the projection of $\boldsymbol{v}_i$ on each $\boldsymbol{v}_j$ from $\boldsymbol{v}_i$. In the modified process, $\boldsymbol{v}_i^{(1)}$ is created by subtracting the projection of $\boldsymbol{v}_i$ on $\boldsymbol{v}_1$ from $\boldsymbol{v}_i$, then $\boldsymbol{v}_i^{(2)}$ is created by subtracting the projection of $\boldsymbol{v}_i^{(1)}$ on $\boldsymbol{v}_2$ from $\boldsymbol{v}_i^{(1)}$, and so on for each previous vector.

The iteration is started with a vector $\boldsymbol{v}_1 = \frac{1}{||\boldsymbol{b}||}\boldsymbol{b}$ and after M steps have matrices fulfilling (13) in the form $AV_M = V_{M+1}\hat{H}_M$. By dropping the last row of $\hat{H}_M$, giving $H_M \in \mathbb{R}^{M\times M}$, the relation

$$AV_M = V_M H_M \tag{15}$$

is obtained. The columns of V_M span the Krylov subspace $\mathcal{K}_M(A, \boldsymbol{b})$ [Saa03, p. 160].

Recall that it is ideal if span $\mathcal{K}_m$ = span $\{\boldsymbol{b}, A\boldsymbol{b}, \ldots, A^{m-1}\boldsymbol{b}\}$ for some m much smaller than the number of columns in A. In the Arnoldi iteration, this corresponds to $A\boldsymbol{v}_i = 0$ at some iteration step i. Thus, an implementation of the iteration should preferably include a check for this eventuality and if it happens abort the iteration (since every $\boldsymbol{v}_j = 0$, $j > i$). In Algorithm 1, this is done on lines 13 – 14.

One final feature which is useful to incorporate in the Arnoldi iteration, is to check for numerical cancellation. This is done by comparing the norm of $\boldsymbol{v}_{m+1}$ after orthogonalization with the norm before orthogonalization and if it falls below some threshold κ, orthogonalize the resulting vector against the previous one more time. It can be shown that doing this more than once per new vector is unnecessary [Saa03, pp. 162–163]. In Algorithm 1, the norms are compared on line 8 and the reorthogonalization is performed on lines 9 – 12.

5 Rosenbrock-Krylov methods

The Rosenbrock-Krylov methods were introduced by Tranquilli and Sandu in a 2014 article [TS14]. The principle behind this family of methods is replacing the Jacobian in a Rosenbrock method with an approximation spanning a Krylov subspace of significantly lower dimension. Thus, the definition is essentially the same as that for Rosenbrock methods except that the $\boldsymbol{k}_i$'s are found through (compare to (10))

$$\boldsymbol{k}_i = h\boldsymbol{f}\Big(y_{n-1} + \sum_{j=1}^{i-1}\alpha_{ij}\boldsymbol{k}_j\Big) + hA_M\sum_{j=1}^{i}\gamma_{ij}\boldsymbol{k}_j \tag{16}$$

where $A_M \in \mathbb{R}^{N\times N}$ is an approximation of J in the M-dimensional Krylov subspace $\mathcal{K}_M(J, \boldsymbol{f}(\boldsymbol{y}_{n-1}))$, with $M \ll N$.

The α, b and γ coefficients are chosen according to the order conditions of ordinary Rosenbrock methods and a number of additional order conditions which guarantee convergence when using A_M rather than J.

Applying the Arnoldi iteration to J with starting vector $\boldsymbol{f}(\boldsymbol{y}_{n-1})$ yields the matrices V_M and H_M in the relation $JV_M = V_M H_M$ (from (15)). Since the columns of V_M are orthogonal, this is equivalent to $H_M = V_M^{\mathrm{T}} J V_M$. Next, defining

$$A_M = V_M H_M V_M^{\mathrm{T}} \tag{17}$$

it can be seen that A_M is the restriction of J onto $\mathcal{K}_M$. Analogously to the way in which (11) was obtained, (16) can be written

$$(I_{N\times N} - h\gamma_{ii}A_M)\boldsymbol{k}_i = h\boldsymbol{f}\Big(\boldsymbol{y}_{n-1} + \sum_{j=1}^{i-1}\alpha_{ij}\boldsymbol{k}_j\Big) + hA_M\sum_{j=1}^{i-1}\gamma_{ij}\boldsymbol{k}_j. \tag{18}$$

Next, each $\boldsymbol{k}_i$ must have a component in $\mathcal{K}_M$ and a component orthogonal to $\mathcal{K}_M$, so they can be written

$$\boldsymbol{k}_i = V_M\boldsymbol{\lambda}_i + \boldsymbol{\mu}_i, \quad \boldsymbol{\lambda}_i \in \mathbb{R}^M, \quad \boldsymbol{\mu}_i \in \mathbb{R}^N. \tag{19}$$

Using (17) and (19), (18) can be written

$$\begin{aligned}(I_{N\times N} &- h\gamma_{ii}V_M H_M V_M^{\mathrm{T}})(V_M\boldsymbol{\lambda}_i + \boldsymbol{\mu}_i) \\ &= h\boldsymbol{f}\Big(\boldsymbol{y}_{n-1} + \sum_{j=1}^{i-1}\alpha_{ij}\boldsymbol{k}_j\Big) + hV_M H_M V_M^{\mathrm{T}}\sum_{j=1}^{i-1}\gamma_{ij}(V_M\boldsymbol{\lambda}_j + \boldsymbol{\mu}_j).\end{aligned}$$

Now, keeping in mind that all of the columns of V_M are orthogonal and that $\boldsymbol{\mu}_i$ is orthogonal to each column of V_M, the substitutions $V_M^{\mathrm{T}}V_M = I_{M\times M}$ and $V_M^{\mathrm{T}}\boldsymbol{\mu}_i = 0$ can be made. This gives:

$$V_M(I_{M\times M} - h\gamma H_M)\boldsymbol{\lambda}_i + \boldsymbol{\mu}_i = h\boldsymbol{f}\Big(\boldsymbol{y}_{n-1} + \sum_{j=1}^{i-1}\alpha_{ij}\boldsymbol{k}_j\Big) + hV_M H_M\sum_{j=1}^{i-1}\gamma_{ij}\boldsymbol{\lambda}_j. \tag{20}$$

Finally, two modifications can be done to (20) in order to get expressions for $\boldsymbol{\lambda}_i$ and $\boldsymbol{\mu}_i$ (that is, calculating $\boldsymbol{k}_i$). Multiplying both sides with V_M^{T} (from the left), gives the expression

$$(I_{M\times M} - h\gamma H_M)\boldsymbol{\lambda}_i = hV_M^{\mathrm{T}}\boldsymbol{f}\Big(\boldsymbol{y}_{n-1} + \sum_{j=1}^{i-1}\alpha_{ij}\boldsymbol{k}_j\Big) + hH_M\sum_{j=1}^{i-1}\gamma_{ij}\boldsymbol{\lambda}_j. \tag{21}$$

Note that this now is a linear system of equations with coefficient matrix $(I_{M\times M} - h\gamma H_M)$ of size $M \times M$, from which $\boldsymbol{\lambda}_i$ can be determined. To get an expression for $\boldsymbol{\mu}_i$, (20) is instead multiplied by $(I_{N\times N} - V_M V_M^{\mathrm{T}})$, which results in

$$\boldsymbol{\mu}_i = h(I_{N\times N} - V_M V_M^{\mathrm{T}})\boldsymbol{f}\Big(\boldsymbol{y}_{n-1} + \sum_{j=1}^{i-1}\alpha_{ij}\boldsymbol{k}_j\Big). \tag{22}$$

Thus, the final method for calculating the $\boldsymbol{k}_i$'s is found by substituting the $\boldsymbol{\lambda}_i$ (from (21)) and (22) into (19):

$$\boldsymbol{k}_i = V_M\boldsymbol{\lambda}_i + h(I_{N\times N} - V_M V_M^{\mathrm{T}})\boldsymbol{f}\Big(\boldsymbol{y}_{n-1} + \sum_{j=1}^{i-1}\alpha_{ij}\boldsymbol{k}_j\Big). \tag{23}$$

The approximation of $\boldsymbol{y}_n$ is calculated in exactly the same way as for the ordinary Rosenbrock methods:

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \sum_{i=1}^{s} b_i\boldsymbol{k}_i.$$

6 Implementation of the methods

Rather than implementing the algorithm described in the article [TS14, Algorithm 2] directly, an "evolutionary" approach was chosen: Successively more "sophisticated" methods were implemented, each building on the knowledge gained from the previous implementations.

6.1 Runge-Kutta methods

The starting point was implementing a Runge-Kutta method. Since this method represented a stepping-stone towards Rosenbrock-Krylov methods of order four, it was decided to implement an explicit Runge-Kutta method of order four. However, it was decided to design a general implementation of an explicit Runge-Kutta method, working with any set of coefficients supplied by the user, rather than writing a hard-coded RK4.

The implementation was written in Python and the basic flow of one step followed Algorithm 2.

Algorithm 2: One step of an s stage explicit Runge-Kutta method

Data: $\boldsymbol{f}$: The function in $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$,
t: Current t value,
$\boldsymbol{y}_{n-1}$: Latest $\boldsymbol{y}$ value,
h: Time step size,
a_{ij}, b_i: Method coefficients

for $i = 1, \ldots, s$ **do**
  $\hat{t} = t + h \sum_{j=1}^{i-1} a_{ij}$
  $\hat{\boldsymbol{y}} = \boldsymbol{y}_{n-1} + h \sum_{j=1}^{i-1} a_{ij} \boldsymbol{k}_j$
  $\boldsymbol{k}_i = f(\hat{t}, \hat{\boldsymbol{y}})$
$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + h \sum_{i=1}^{s} \boldsymbol{k}_i b_i$

An advantage of having the Runge-Kutta implementation at hand, despite it not being the object of research, was its utility as a benchmark for testing other methods. For example, for (non-stiff) problems with unknown exact solution, RK4 can be run with a very small step size to produce a reference solution.

6.2 Rosenbrock methods

Since the Rosenbrock-Krylov methods developed in [TS14] are of order 4, it was decided that the logical next step was implementing an ordinary Rosenbrock method of order 4.

Analogously to the Runge-Kutta implementation, the implementation was written in such as way as to be "coefficient agnostic", working with whatever

set of coefficients is supplied by the user. The process for calculating the coefficients of a fourth order method from a set of order conditions is given in [HW10, pp. 108–110].

The order conditions for a fourth order Rosenbrock method are:

$$b_1 + b_2 + b_3 + b_4 = 1 \tag{24a}$$

$$b_2\beta_2' + b_3\beta_3' + b_4\beta_4' = \frac{1}{2} - \gamma \tag{24b}$$

$$b_2\alpha_2^2 + b_3\alpha_3^2 + b_4\alpha_4^2 = \frac{1}{3} \tag{24c}$$

$$b_3\beta_{32}\beta_2' + b_4(\beta_{42}\beta_2' + \beta_{43}\beta_3') = \frac{1}{6} - \gamma + \gamma^2 \tag{24d}$$

$$b_2\alpha_2^3 + b_3\alpha_3^3 + b_4\alpha_4^3 = \frac{1}{4} \tag{24e}$$

$$b_3\alpha_3\alpha_{32}\beta_2' + b_4\alpha_4(\alpha_{42}\beta_2' + \alpha_{43}\beta_3') = \frac{1}{8} - \frac{\gamma}{3} \tag{24f}$$

$$b_3\beta_{32}\alpha_2^2 + b_4(\beta_{42}\alpha_2^2 + \beta_{43}\alpha_3^2) = \frac{1}{12} - \frac{\gamma}{3} \tag{24g}$$

$$b_4\beta_{43}\beta_{32}\beta_2' = \frac{1}{24} - \frac{\gamma}{2} + \frac{3\gamma^2}{2} - \gamma^3 \tag{24h}$$

where $\beta_{ij} = \alpha_{ij} + \gamma_{ij}$, $\alpha_i = \sum_{j=1}^{i-1} \alpha_{ij}$ and $\beta_i' = \sum_{j=1}^{i-1} \beta_{ij}$.

First, γ was chosen to be 0.572816062482135. Next, the choices $b_3 = 0$, $\alpha_2 = 2\gamma$ and $\alpha_3 = \alpha_4 = (1/5 - \alpha_2/4)/(1/4 - \alpha_2/3)$ were made and the system (24a), (24c), (24e) was solved for b_1, b_2 and b_4.

Next, β_2', β_3' and β_4' were determined, which required several steps: Multiplying (24d) with α_2^2 and subtracting from (24g) multiplied by β_2' yield the relation

$$b_4\beta_{43}(\beta_2'\alpha_3^2 - \beta_3'\alpha_2^2) = \beta_2'\left(\frac{1}{12} - \frac{\gamma}{3}\right) - \alpha_2^2\left(\frac{1}{6} - \gamma + \gamma^2\right). \tag{25}$$

Furthermore, to allow for automatic step size selection (a concept which will be introduced in section 6.4), it was required that

$$(\beta_2'\alpha_4^2 - \beta_4'\alpha_2^2)\beta_{32}\beta_2' = (\beta_2'\alpha_3^2 - \beta_3'\alpha_2^2)(\beta_{42}\beta_2' + \beta_{43}\beta_3'). \tag{26}$$

By choosing

$$b_4\beta_{43}\alpha_3^2(\alpha_3 - \alpha_2) = \frac{1}{20} - \frac{\gamma}{4} - \alpha_2\left(\frac{1}{12} - \frac{\gamma}{3}\right),$$

β_{43} was determined. This allowed to find expressions for $\beta_{32}\beta_3'$ and $(\beta_{42}\beta_2' + \beta_{43}\beta_3')$ from (24h) and (24d), respectively. These expressions were inserted into (26), which combined with (24b) and (25) finally resulted in a system of linear equations for β_2', β_3' and β_4'.

β_{32} and β_{42} were determined from the expressions for $\beta_{32}\beta'_3$ and $(\beta_{42}\beta'_2 + \beta_{43}\beta'_3)$, which allowed to find β_{21}, β_{31} and β_{41} from the definition of β'_i. The choices $\alpha_{43} = 0$ and $\alpha_{32} = \alpha_{42}$ were made, which allowed for determining the value of α_{32} (and α_{42}) from (24f). α_{21}, α_{31} and α_{41} were determined through the definition of α_i, which made it possible to determine the γ_{ij} coefficients from the definition of β_{ij}.

This process resulted in the coefficients given in Table 7, in Appendix A. The Rosenbrock method using these coefficients will be referred to as ROS4 throughout the remaining thesis.

Algorithm 3: One step of an s stage Rosenbrock method

Data: $\boldsymbol{f}$: The function in $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$,
t_{n-1}: Latest t value,
$\boldsymbol{y}_{n-1}$: Latest $\boldsymbol{y}$ value,
h: Time step size,
$\alpha_{ij}, b_i, \gamma_{ij}$: Method coefficients

1 $J = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}(t_{n-1}, \boldsymbol{y}_{n-1})$
2 $\boldsymbol{d}_t = \frac{\partial \boldsymbol{f}}{\partial t}(t_{n-1}, \boldsymbol{y}_{n-1})$
3 **for** $i = 1, \ldots, s$ **do**
4 $\quad \hat{t} = t_{n-1} + h \sum_{j=1}^{i-1} \alpha_{ij}$
5 $\quad \hat{\boldsymbol{y}} = \boldsymbol{y}_{n-1} + \sum_{j=1}^{i-1} a_{ij} \boldsymbol{k}_j$
6 $\quad lhs = I - h\gamma_{ii} J$
7 $\quad rhs = h\boldsymbol{f}(\hat{t}, \hat{\boldsymbol{y}}) + h^2 \sum_{j=1}^{i-1} \gamma_{ij} \boldsymbol{d}_t + hJ \sum_{j=1}^{i-1} \gamma_{ij} \boldsymbol{k}_j$
8 $\quad \boldsymbol{k}_i = \texttt{solve}(lhs, rhs)$
9 $\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \sum_{i=1}^{s} b_i \boldsymbol{k}_i$

One step of the method, given in Algorithm 3, is fairly reminiscent of one step of the explicit Runge-Kutta method (Algorithm 2), with the main difference being that each $\boldsymbol{k}_i$ is given by solving a system of linear equations (the system given by (11)). In Algorithm 3 this is done on lines 6 – 8, where `solve` is some linear system solver (in this particular implementation, `numpy.linalg.solve`).

The system requires the variables J and $\boldsymbol{d}_t$, calculated in lines 1 and 2, respectively. Together, these two correspond to the Jacobian of the "augmented" system 2. In the best case scenario, the user supplies the algorithm with a function which given t_{n-1} and $\boldsymbol{y}_{n-1}$ returns the J and $\boldsymbol{d}_t$, but in the case where this is not done, the method uses finite (forward) differences to approximate it. This simply means that since

$$\frac{\partial f_i}{\partial y_j} \approx \frac{f_i(\boldsymbol{y} + \boldsymbol{e}_j \sigma) - f_i(\boldsymbol{y})}{\sigma}$$

where $\boldsymbol{e}_j$ is the unit vector with 1 in the jth position, for small enough σ, all of the elements of the Jacobian are calculated in this way for the point in question. The only pitfall lies in the choice of σ since a too large value will give an inexact

approximation and a too small value will risk catastrophic cancellation due to floating point roundoff errors. The σ was chosen to be $1.490 \cdot 10^{-8}$, which can be shown to be close to the optimal size when the calculations are done using double precision floating point numbers [AWF11].

6.3 Rosenbrock-Krylov methods

The methods ROK4a and ROK4b as defined in [TS14] were implemented in Python. As in the case with the previous methods this was done by implementing a general Rosenbrock-Krylov method, working with any set of coefficients supplied by the user. The order conditions for Rosenbrock-Krylov methods are almost the same as for ordinary Rosenbrock methods, the only difference being that the condition (24g) is replaced with the two conditions

$$b_3\alpha_{32}\alpha_2^2 + b_4(\alpha_{42}\alpha_2^2 + \alpha_{43}\alpha_3^2) = \frac{1}{12} \tag{27a}$$

$$b_3\gamma_{32}\alpha_2^2 + b_4(\gamma_{42}\alpha_2^2 + \gamma_{43}\alpha_3^2) = -\frac{\gamma}{3} \tag{27b}$$

where $\alpha_2 = \alpha_{21}$ and $\alpha_3 = \alpha_{31} + \alpha_{32}$.

Since [TS14, p. A1329] states the coefficients for the methods ROKa and ROK4b, there was no need to calculate them from scratch. However, the article walks through the process of obtaining the coefficients for ROK4a, which is virtually the same as for regular Rosenbrock methods. While following these steps, it was discovered that there are a few misprints in this part of the article: It is claimed that the method ROK4a uses the coefficient choices

$$\alpha_2 = \frac{1}{2}, \quad \alpha_3 = 1, \quad \alpha_4 = 1$$

[TS14, p. A1327], but these do not result in the final coefficients given in the article. Rather, these follow from making the coefficient choices

$$\alpha_2 = 1, \quad \alpha_3 = \frac{1}{2}, \quad \alpha_4 = \frac{1}{2}.$$

Furthermore, the condition which is required for step size selection [TS14, eq. 4.3] should be identical to (26), but is missing an exponent from an α_2. Lastly, there is a coefficient choice $\alpha_{32} = \beta_{32}$ which is done and never mentioned, but can be deduced from the coefficients given in [TS14, Table 1].

Following the steps outlined in [TS14], with the above mentioned changes results in the coefficients for ROK4a, which is a four stage, fourth order, L-stable Rosenbrock-Krylov method. Its coefficients are given in Table 8 in appendix A.

ROK4b is a six stage, fourth order, stiffly accurate Rosenbrock-Krylov method. That an s stage method is stiffly accurate means (in the case of Rosenbrock methods) that $b_i = \beta_{si}$, $i = 1, \ldots, s$ and $\alpha_s = 1$ [HW10, p. 418]. Methods fulfilling these conditions will perform better at very stiff problems. The coefficients of this method is given in Table 9 in appendix A.

Algorithm 4: One step of an s stage Rosenbrock-Krylov method

Data: $\boldsymbol{f}$: The function in $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$,
t_{n-1}: Latest t value,
$\boldsymbol{y}_{n-1}$: Latest $\boldsymbol{y}$ value,
h: Time step size,
$\alpha_{ij}, b_i, \gamma_{ij}$: Method coefficients,
M: Dimension of the Krylov subspace

1 $V_M, H_M, \boldsymbol{w} = \texttt{arnoldi}(\boldsymbol{f}, t_{n-1}, \boldsymbol{y}_{n-1}, M)$
2 **for** $i = 1, \ldots, s$ **do**
3 $\quad \hat{t} = t_{n-1} + h \sum_{j=1}^{i-1} \alpha_{ij}$
4 $\quad \hat{\boldsymbol{y}} = \boldsymbol{y}_{n-1} + \sum_{j=1}^{i-1} a_{ij} \boldsymbol{k}_j$
5 $\quad \boldsymbol{F}_i = \boldsymbol{f}(\hat{t}, \hat{\boldsymbol{y}})$
6 $\quad \boldsymbol{\phi}_i = V_M^{\mathrm{T}} \boldsymbol{F}_i + \boldsymbol{w}$
7 $\quad \boldsymbol{\lambda}_i = \texttt{solve}\left(I - h\gamma_{ii} H_M, h\boldsymbol{\phi}_i + hH_M \sum_{j=1}^{i-1} \gamma_{ij} \boldsymbol{\lambda}_j\right)$
8 $\quad \boldsymbol{k}_i = V_M \boldsymbol{\lambda}_i$
9 $\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \sum_{i=1}^{s} b_i \boldsymbol{k}_i$

A general Rosenbrock-Krylov method is presented in Algorithm 4 and is in many ways reminiscent of an ordinary Rosenbrock method (Algorithm 3). The differences being that in each step of a Rosenbrock-Krylov method, H_M and V_M must be found through the Arnoldi iteration (Algorithm 1). In Algorithm 4, this is done in line 1, where `arnoldi` is assumed to be an implementation of the Arnoldi iteration. Since the algorithm shows an explicitly non-autonomous problem, for $\boldsymbol{y} \in \mathbb{R}^N$ the augmented system (2) will result in a Jacobian of size $(N+1)\times(N+1)$ and thus the Arnoldi iteration will produce $H_M \in \mathbb{R}^{M\times M}$ and $\hat{V}_M \in \mathbb{R}^{(N+1)\times M}$. In this implementation, the row of $\hat{V}_M$ corresponding to the time component is "split off" as $\boldsymbol{w} \in \mathbb{R}^M$, and the remainder of $\hat{V}_M$ is referred to as $V_M \in \mathbb{R}^{N\times M}$.

In lines 5 and 6, respectively, the variables $\boldsymbol{F}_i$ and $\boldsymbol{\phi}_i$ are introduced. This is for brevity in the algorithm description as well as for avoiding repeating calculations in the actual implementation. The equivalent of (21) is solved using a linear system solver in line 7, thereby giving $\boldsymbol{\lambda}_i$, and $\boldsymbol{k}_i$ is calculated according to (23) in line 8.

At first, the algorithm for the Arnoldi iteration from [TS14, Algorithm 1] was difficult to replicate exactly: The constant κ used for cancellation checking (see Algorithm 1, line 8) was not specified. Through a publicly available MATLAB implementation of the Rosenbrock-Krylov integrator authored by Sandu's research group, the value of κ was found to be 0.25 [Com17, line 304]. That is, if the norm of the resulting vector after orthogonalization is less than 1/4 of the vector pre-orthogonalization, a reorthogonalization is performed. This might seem "aggressive", in the sense that it is easy to imagine that the resulting vector shrinks this much in one step without mis-calculations, but it is necessary to keep in mind that

an additional reorthogonalization once in a while is a small price to pay compared to having the algorithm break down because of cancellations.

Secondly, the algorithm described in the article included no check for whether the new-found vector was zero. Thus, the algorithm would never quit before calculating all M basis vectors. This seemed like unnecessary work, and as such a check was included in the Arnoldi iteration used in this implementation (see Algorithm 1, line 13).

6.4 Automatic step size selection

Given a specific error tolerance, it is desirable to take the largest time steps possible, since fewer steps means less operations and faster execution time. However, the user generally does not know what step size is appropriate for a given problem, which obviously is a problem for making a good choice. This makes the case for using adaptive step sizing, that is, adjusting the step size between steps to make sure they are taken at a relatively optimal size.

This is done by using two different sets of b coefficients in the method. The second set of coefficients, denoted $\hat{b}$, are chosen such that they fulfill the conditions for a method of one order *lower* than the main method. During each step of the method, both the approximation $\boldsymbol{y}_n$ (using the b coefficients) and the approximation $\hat{\boldsymbol{y}}_n$ (using the $\hat{b}$ coefficients) are calculated. Computing this additional approximation is relatively cheap for a Rosenbrock or Rosenbrock-Krylov method, since the only calculation that needs to be done is $\hat{\boldsymbol{y}}_n = \boldsymbol{y}_{n-1} + \sum \hat{b}_i \boldsymbol{k}_i$.

A measure of the error is obtained by comparing $\boldsymbol{y}_n$ and $\hat{\boldsymbol{y}}_n$ and if it falls below a pre-determined threshold the approximation is deemed good enough and accepted as the next step of the method. Otherwise, a new, smaller step size is computed, based on how far from the threshold the error was and the process is repeated until an acceptable step size is found. Note that this is really an approximation of the error of the method of lower order, but when the higher order approximation has already been calculated it would be wasteful not to use it.

In this implementation, the step size selection method described by Hairer, Nørsett and Wanner was used [HNW08, pp. 167–169]. In this process, the error measure is defined as

$$err = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(\frac{y_{n_i} - \hat{y}_{n_i}}{sc_i} \right)^2}$$

where $sc_i = \varepsilon_{abs} + \max(y_{(n-1)_i}, y_{n_i}) \cdot \varepsilon_{rel}$ and ε_{abs} and ε_{rel} are user-supplied absolute and relative tolerance levels, respectively.

Next, a new step size is calculated using the formula

$$h_{new} = h \cdot \min(\vartheta_{max}, \max(\vartheta_{min}, \vartheta(1/err)^{1/(q+1)})).$$

where ϑ_{min} and ϑ_{max} are safety factors to make sure that the step size neither shrinks nor grows too fast, ϑ is a safety factor that makes sure steps are slightly

smaller than what is "optimal" (since rejecting steps and recalculating is costly) and q is the smallest of the order of the regular method and the order of the embedded formula. Note that (unless hindered by the safety factors) if $err > 1$, the step size will decrease and if $err < 1$, the step size will increase.

If the error measure is greater than 1 (which means that the average squared error is bigger than the user supplied tolerances) the current step is rejected and a new step from $\boldsymbol{y}_{n-1}$ is attempted, using the new, smaller step size. This process is repeated until a result with acceptable error is found and that step is accepted as $\boldsymbol{y}_n$.

The time loop of the implemented methods with adaptive step sizing is shown in Algorithm 5. All of the methods were equipped with an algorithm for automatically selecting an initial step size (as described in [HNW08, p. 169]), unless one was explicitly provided by the user, and a method for adapting the step size between steps, as described above, using $\vartheta_{min} = 0.2$, $\vartheta_{max} = 6.0$, $\vartheta = 0.9$, for the Rosenbrock and Rosenbrock-Krylov methods, as suggested by Hairer and Wanner [HW10, p. 112].

Algorithm 5: The time loop for any of the implemented methods when incorporating adaptive step sizes. Here, `step` is assumed to be one step of the method in question.

Data: $\boldsymbol{f}$: The function in $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$,
$[t_{start},\ t_{end}]$: Time interval,
$\boldsymbol{y}_0$: Initial $\boldsymbol{y}$ value, N components,
h: Initial step size,
$\varepsilon, \varepsilon_{abs}, \varepsilon_{rel}$: Tolerances for step selection

while $t < t_{end}$ **do**
  $h = \min(h, t_{end} - t)$
  while $err > 1$ **do**
    $\boldsymbol{y},\ \hat{\boldsymbol{y}} = \texttt{step}(t,\ \boldsymbol{y}_n,\ h)$
    $err = \sqrt{1/N \sum (y_i - \hat{y}_i)/sc_i}$ where
      $sc_i = \varepsilon_{abs} + \max(y_{0_i}, y_i) \cdot \varepsilon_{rel}$
    $h = h \cdot \min(\vartheta_{max}, \max(\vartheta_{min}, \vartheta(1/err)^{1/(q+1)}))$
  $t = t + h$
  $\boldsymbol{y}_{n+1} = \boldsymbol{y}$

In the case of the Runge-Kutta method, it was discovered that it is not possible to implement adaptive step sizing for RK4 without adding method-specific subroutines to the implementation. Thus, a second method which supports adaptive step sizes in the way described above, the Runge-Kutta-Fehlberg method, was implemented. The method is an explicit, six-stage, fourth order method with an

embedded fifth-order method with the Butcher tableau

$$
\begin{array}{c|cccccc}
0 & & & & & & \\
1/4 & 1/4 & & & & & \\
3/8 & 3/32 & 9/32 & & & & \\
12/13 & \frac{1932}{2197} & -\frac{7200}{2197} & \frac{7296}{2197} & & & \\
1 & \frac{439}{216} & -8 & \frac{3680}{513} & -\frac{845}{4104} & & \\
1/2 & -8/27 & 2 & -\frac{3544}{2565} & \frac{1859}{4104} & -11/40 & \\
\hline
 & 25/216 & 0 & 1408/2565 & 2197/4104 & -1/5 & 0 \\
 & 16/135 & 0 & 6656/12825 & 28561/56430 & -9/50 & 22/55
\end{array}
$$

where the upper row of b coefficients are for the fourth order method and the lower row are for the fifth order method.

7 Numerical performance

The methods were tested for order of convergence, performance on stiff problems and raw computational efficiency.

7.1 Lorenz-96

In order to verify that the methods were correctly implemented, they were applied to the Lorenz-96 model and their order of convergence was calculated. This mirrors one of the tests done in [TS14]. For N dimensions, the Lorenz-96 model is defined by

$$\frac{dy_i}{dt} = (y_{i+1} - y_{i-2})y_{i-1} - y_i + F, \quad i = 1, \ldots, N \tag{28}$$

where $y_0 = y_N$, $y_{-1} = y_{N-1}$, $y_{N+1} = y_1$ and F is a constant. It is known that $F = 8$ makes (28) a very chaotic system.

In [TS14] the problem is solved for $N = 40$, $F = 8$ and $t \in [0, 3]$, finding that the methods have the convergence rates given in Table 1 [TS14, p. A1332].

	ROS4	ROK4a	ROK4b
M = 40	4.01	4.01	3.99
M = 4	3.03	4.01	3.99

Table 1: *Convergence rates of different methods on the Lorenz-96 model, as found in [TS14].*

The experiment was repeated with the implementations made for this thesis, using the same values of $N = 40$, $F = 8$, and $t \in [0, 0.3]$, and an initial value $\boldsymbol{y}$ given by $y_1 = 1.01$, $y_i = 1$, $i = 2, \ldots, 40$.[3]

[3]Unfortunately, the initial value used is not disclosed in [**tranquilli-sandu.**]

As there is no easy way to calculate the exact state of the Lorenz-96 model at a given time t, a solution $\boldsymbol{y}_{sol}$ calculated with RK45 at a step size of $h = \frac{0.3}{2000}$ (i.e. using 2000 timesteps) was considered the "exact" solution for the purpose of calculating errors.

For each run of each method, the error e_h for a given step size h was calculated as $e_h = ||\boldsymbol{y}_h - \boldsymbol{y}_{sol}||_1$ (where $\boldsymbol{y}_h$ signifies the solution found using the step size h). Next, the convergence rate between two consecutive step sizes h and $h/2$ was calculated as $o = \log_2(\frac{e_h}{e_{h/2}})$.

	Rate of convergece between steps						
Number of steps	RK4	ROS4 (full)	ROK4a (full)	ROK4b (full)	ROS4 (4-dim)	ROK4a (4-dim)	ROK4b (4-dim)
20 – 40	4.01	3.99	3.98	3.98	3.74	3.98	3.98
40 – 80	4.01	4.03	3.99	3.99	3.49	3.99	3.99
80 – 160	4.12	3.82	3.96	3.99	3.33	3.96	3.99
160 – 320	5.11	1.42	3.52	3.84	2.63	3.52	3.82

TABLE 2: *Convergence rates of different methods on the Lorenz-96 model ($N = 40$, $F = 8$) using the full Jacobian and a four dimensional approximation.*

As can be seen in Table 2, similar (albeit not identical) convergence rates as those in Table 1 were found for these implementations of the Rosenbrock-Krylov methods (as well as the ordinary Rosenbrock method). This is reassuring, since it implies that the methods are implemented correctly.

For the case with the full Jacobian, the errors are shown in Figure 4. It is clear that ROS4, ROK4a and ROK4b have fourth order of convergence, at least for all but the smallest step sizes.

Using a four-dimensional approximation rather than the full Jacobian gave the expected results for the Rosenbrock-Krylov methods: Both ROS4a and ROS4b retain fourth order convergence. Since it was mentioned in [TS14, p. A1322] that the coefficients of any fourth order Rosenbrock method correspond to a Rosenbrock-Krylov method of at least order 3, ROS4 was also included in this experiment. As was expected, the method still converges, albeit at a lower convergence rate. The errors are shown in Figure 5.

Thus it is verified that, for this problem, the Rosenbrock-Krylov methods converge as expected when using a lower-dimensions approximation of the Jacobian. The problem does not, however, highlight any advantage over the explicit RK4. The Lorenz-96 model does not appear to be stiff enough to cause problems for explicit methods.

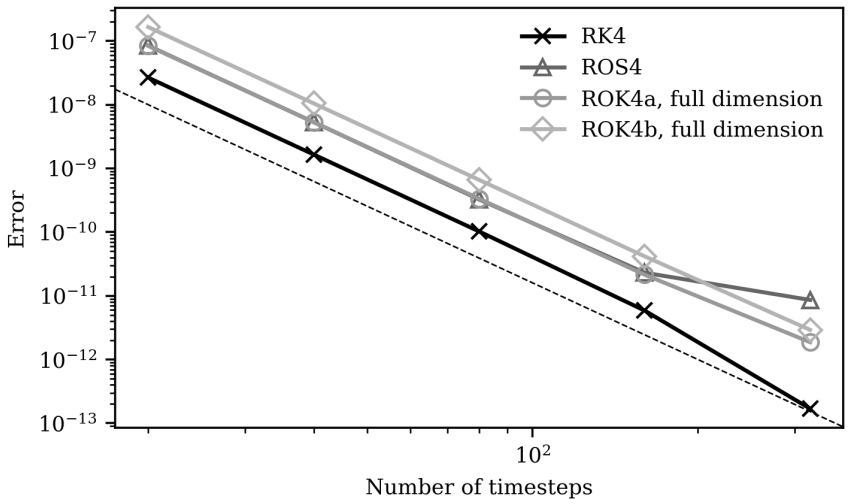

FIGURE 4: *Error versus number of timesteps for different methods when solving the Lorenz-96 model ($N = 40$, $F = 8$) using the full Jacobian. The dashed line shows the slope corresponding to fourth order of convergence.*

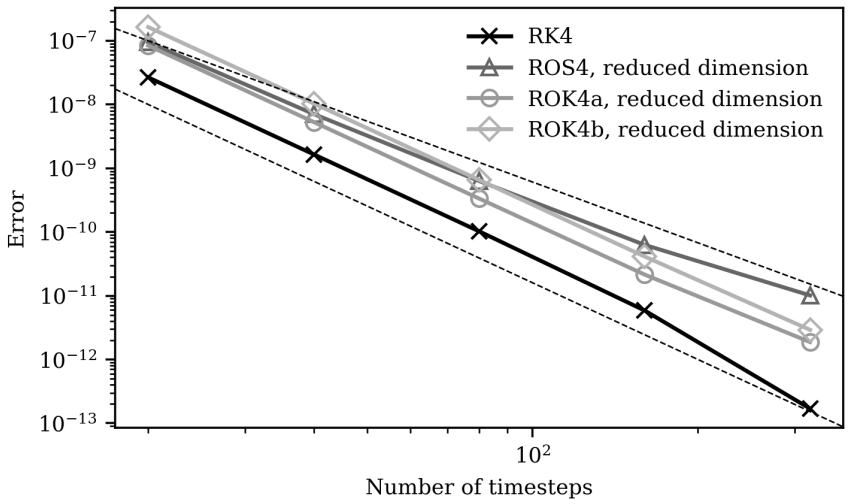

FIGURE 5: *Error versus number of timesteps for different methods when solving the Lorenz-96 model ($N = 40$, $F = 8$) using a reduced, 4-dimensional approximations of the Jacobian. The dashed lines show the slopes corresponding to third and fourth order of convergence.*

7.2 A modification to the Lorenz model

In order to test a non-autonomous system, a simple modification was made to the Lorenz-96 model. Rather than using the problem as defined in (28), everything was multiplied by a damping factor $\frac{1}{t+1}$, giving

$$\frac{dy_i}{dt} = ((y_{i+1} - y_{i-2})y_{i-1} - y_i + F) \cdot \frac{1}{t+1}, \quad i = 1, \ldots, N \tag{29}$$

This problem was solved for the same conditions, $N = 40$, $F = 8$, $t \in [0, 0.3]$, and initial value $y_1 = 1.01$, $y_i = 1$, $i = 2, \ldots, 40$, once using the full Jacobian and once using a four dimensional approximation. The error and order of convergence were calculated in the same way as described in section 7.1.

	Rate of convergece between steps						
Number of steps	RK4	ROS4 (full)	ROK4a (full)	ROK4b (full)	ROS4 (4-dim)	ROK4a (4-dim)	ROK4b (4-dim)
20 – 40	3.99	4.01	4.00	4.00	4.05	4.00	4.00
40 – 80	3.92	4.05	4.00	4.00	4.10	4.00	4.00
80 – 160	2.67	4.57	4.02	4.00	4.42	4.02	4.00
160 – 320	0.14	2.27	4.34	4.06	2.08	4.34	4.07

Table 3: *Convergence rates of different methods on the modified Lorenz-96 model ($N = 40$, $F = 8$) using the full Jacobian and a four dimensional approximation.*

The results are shown in Table 3, and for ROK4a and ROK4b they were consistent with what was observed for the ordinary Lorenz-96 model. When using the full Jacobian, they both exhibit fourth order convergence. The same initially holds true for ROS4, but at the smallest step size the order of convergence deteriorates even worse than for the ordinary Lorenz-96 model, as can be seen in Figure 6.

When using the reduced Jacobian, ROK4a and ROK4b as expected still had fourth order convergence. ROS4, however, initially displayed fourth order convergence (rather than lower, which was expected), before once again abruptly dropping to second order convergence for the smallest step size, as can be seen in Figure 7. As can also be seen in the figures, RK4 performed much better initially and reached the "smallest possible" error earlier than for the ordinary Lorenz-96 model.

7.3 A stiff problem

To study the advantage of the implicit methods over explicit Runge-Kutta methods for stiff problems, it was necessary to test the methods on one such problem. While it is easy enough to find a problem where the explicit methods "explode"

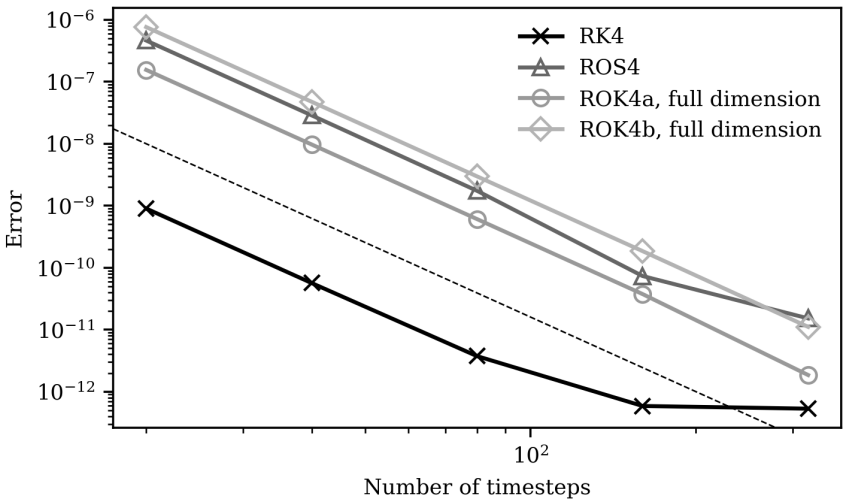

FIGURE 6: *Error versus number of timesteps for different methods when solving the modified Lorenz-96 system ($N = 40$, $F = 8$), using the full Jacobian. The dashed line shows the slope corresponding to fourth order of convergence.*

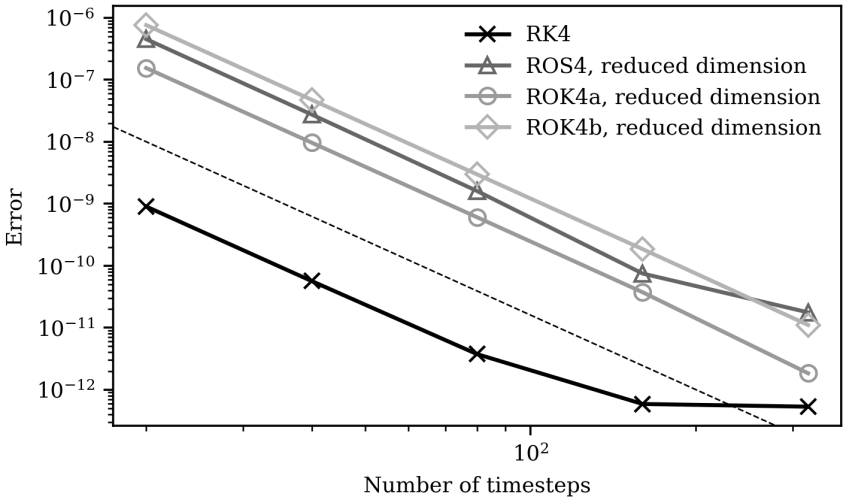

FIGURE 7: *Error versus number of time steps for different methods when solving the modified Lorenz-96 system ($N = 40$, $F = 8$), using a reduced, 4-dimensional approximation of the Jacobian. The dashed line shows the slope corresponding to fourth order of convergence.*

given a particular, fixed, step size, that does not really give a measure on difference in performance. Therefore, it was decided to use methods with adaptive step sizes and a given tolerance level, studying the number of steps required to approximate the solution at that level of precision. The defining characteristic of a stiff problem is, after all, that it requires very small step sizes to get a correct solution with an explicit method.

The problem chosen was a combustion model given as an example of a stiff problem by Shampine and Thompson [ST07]. The concentration of a reactant in a chemical reaction is given by

$$y'(t) = y^2(1-y), \quad t \in [0, 2/d], \quad y(0) = d$$

where d is a small positive number.

RKF45, ROS4, ROK4a and ROK4b were used to solve this model for $d = 0.001$. For all of the methods, adaptive step size selection was turned on and the tolerance level set to $\varepsilon_{abs} = \varepsilon_{rel} = 10^{-7}$. The result are given in Table 4, and as can be seen the implicit methods use significantly fewer steps than RKF45.

	RKF45	ROS4	ROK4a	ROK4b
Steps	3765	271	238	315

TABLE 4: *Number of steps used by each method when solving the combustion model.*

Qualitatively, it can be noted that all of the implicit methods take most of their steps around $t = 1000$, where y goes from very close to 0 to very close to 1 in a relatively short amount of time. The three implicit methods can clear the almost "flat" parts before and after this combustion in a handful of steps, but the explicit RKF45 method needs to keep on using minuscule step sizes. This is shown for RKF45 and ROK4a in Figure 8.

7.4 The Prothero-Robinson Problem

A well known problem for testing stability properties of numerical methods for ODEs is the Prothero-Robinson problem, which was suggested in an 1974 article by two authors bearing those names [PR74]. It is an initial-value problem of the form

$$y' = g'(t) + \lambda(y - g(t)), \quad y(0) = g(0) \tag{30}$$

where λ is a number with negative real part and $g(t)$ is a differentiable function. Note that the solution to this differential equation is $g(t)$.

This is a stiff problem with the property that if λ is large in absolute value compared to the step size of the method, certain methods will achieve a reduced order of convergence. Stiffly accurate methods typically suffer less order reduction than methods that are not stiffly accurate. Keeping in mind that ROK4b is

stiffly accurate whereas ROK4a is not, one would expect that ROK4b would perform better than ROK4a when applied to the Prothero-Robinson problem.

This hypothesis was tested by solving (30) where $g(t) = \sin(\pi/4 + t)$ and $\lambda = -10^6$, that is:

$$y' = \cos\left(\frac{\pi}{4} + t\right) - 10^6\left(y - \sin\left(\frac{\pi}{4} + t\right)\right), \quad y(0) = \frac{1}{\sqrt{2}}.$$

The results of applying the different models to this problem is presented in Table 5. Remarkably, ROK4b suffers from even worse order reduction than ROK4a. However, as Figure 9 shows, when looking at the error versus step size it becomes clear that ROK4b still severely outperforms ROK4a, being able to take comparatively huge steps with a much smaller error. Note that ROK4b at 20 time steps has two orders of magnitude smaller error than ROK4a at 640 time steps and the advantage over ROS4 is even more significant.

	Rate of convergece between steps		
Number of timesteps	ROS4	ROK4a	ROK4b
20 – 40	1.00	2.09	1.17
40 – 80	1.00	2.05	1.09
80 – 160	1.00	2.02	1.05
160 – 320	1.00	2.01	1.02
320 – 640	1.00	2.00	1.01

TABLE 5: *Convergence rates of different methods on the Prothero-Robinson problem for* $g(t) = \sin(\pi/4 + t)$ *on the interval* $[0, 2]$.

This could possibly be explained by the fact that ROK4a and ROK4b are only *guaranteed* to have order 4 if the Krylov subspace has at least dimension 4 [TS14, Thm. 3.6], which is obviously not going to be the case in a one-dimensional problem. In order to test this hypothesis, a multi-dimensional equivalent of the Prothero-Robinson problem would have to be found or developed, but that was beyond the scope of this thesis. The ordinary Rosenbrock method ROS4 also suffers severe order reduction, but that is as expected.

RK4 was not stable for any of the step sizes tested in this experiment and therefore omitted from the results. This serves as a clear example of why implicit methods are preferable when solving very stiff problems.

7.5 Computation times

One main point of using an approximation rather than the full Jacobian is to limit the number of computations performed in the execution of the algorithm, thereby reducing computation time. Furthermore, for very large problems it might not

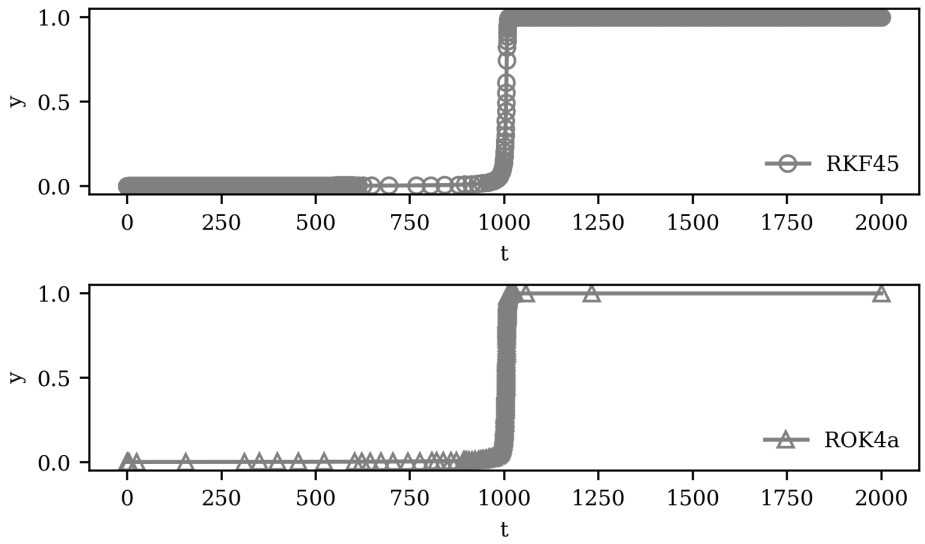

FIGURE 8: *Solution for the combustion model using RKF45 and ROK4a. Steps marked by a circle and a triangle, respectively.*

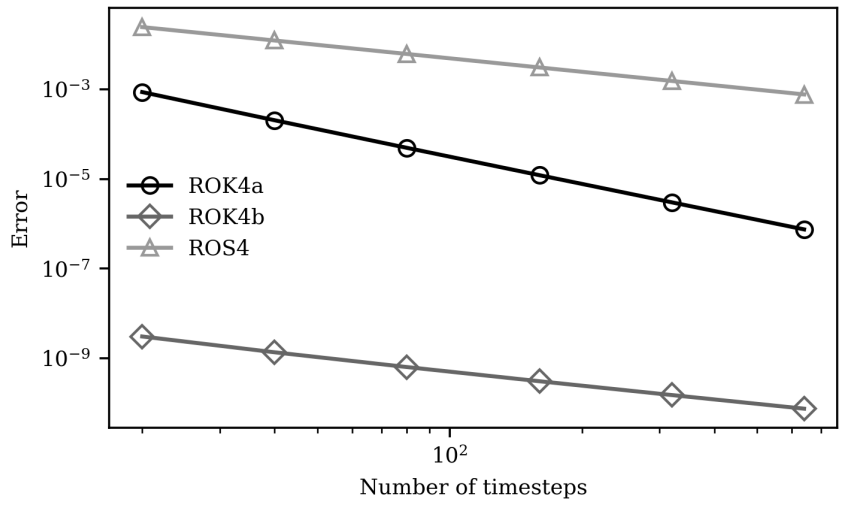

FIGURE 9: *Error versus number of timesteps for different methods when solving the Prothero-Robinson problem for* $g(t) = \sin(\pi/4 + t)$ *on the interval* $[0, 2]$

even be possible to solve the linear systems required by the Rosenbrock method, so in some cases approximations are unavoidable. Thus, a comparison of the efficiency between ROS4 and the Rosenbrock-Krylov methods is in order. The first experiment conducted solved a 100 dimensional version of the Lorenz-96 model (28) with

$$\boldsymbol{y}_{0_1} = 1.01, \quad \boldsymbol{y}_{0_i} = 1, i = 2, \ldots, 100, \quad F = 8$$

using adaptive step sizes for different tolerance levels, on the interval $t \in [0, 0.5]$. ROS4 naturally used the full Jacobian, whereas ROK4a and ROK4b used a four-dimensional approximation.

Each method was run 50 times at each tolerance level and an average execution time was calculated, with execution times being measured by the Python package `timeit`. The errors plotted versus the average execution time for the various methods is shown in Figure 10. As can be seen, ROK4a and ROK4b execute faster than ROS4 for all tolerance levels, in many cases one order of magnitude faster.

Since the Arnoldi iteration in and of itself requires a fair bit of calculation, the next experiment was to see if there was "break point" regarding problem size above which the cost of the Arnoldi iteration was consistently smaller than the cost of using the full Jacobian. The Lorenz-96 model was solved using ROS4, ROK4a and ROK4b for starting y_0 of increasing length, with adaptive step sizes and tolerance level 10^{-7}. As in the previous experiment the average execution times were calculated from 50 runs of each combination of method and problem size. The results are given in Table 6 and shown in Figure 11.

	Dimensions					
Method	20	40	80	160	320	640
ROS4	0.0259	0.0512	0.179	1.986	4.362	7.882
ROK4a (4-dim)	0.0310	0.0365	0.0477	0.119	0.207	0.465
ROK4b (4-dim)	0.0276	0.0320	0.0408	0.0985	0.176	0.352

TABLE 6: *Average execution time (in seconds) for one run of each method when solving the Lorenz-96 model with various numbers of dimensions.*

As can be seen, it is only for the smallest problem size that the cost of the Rosenbrock-Krylov methods are greater than for ROS4. As the problem grows larger, ROK4a and ROK4b execute in less than one tenth of the time required by ROS4, which would indicate that they are good method choices for large systems of ODEs. However, it is also worth noting that the Rosenbrock-Krylov methods "break even" with the ordinary Rosenbrock method at such a small problem size that they could be used even for smaller problems.

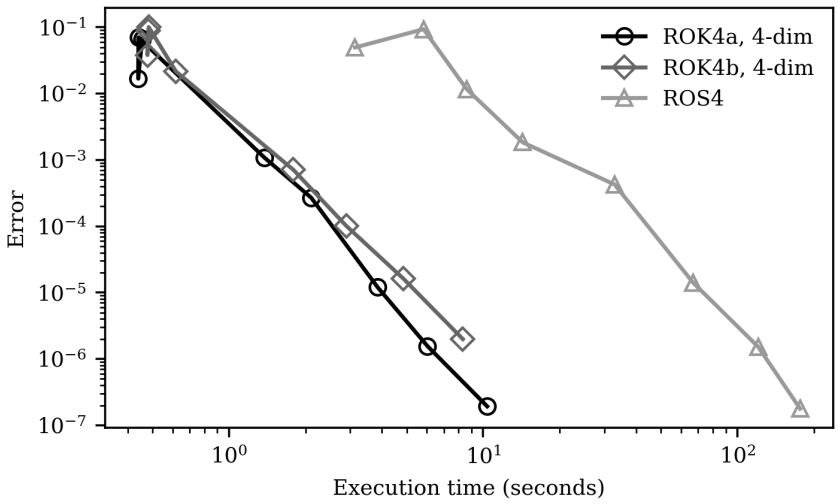

FIGURE 10: *Error versus execution time for the different methods when solving the Lorenz 96 model.*

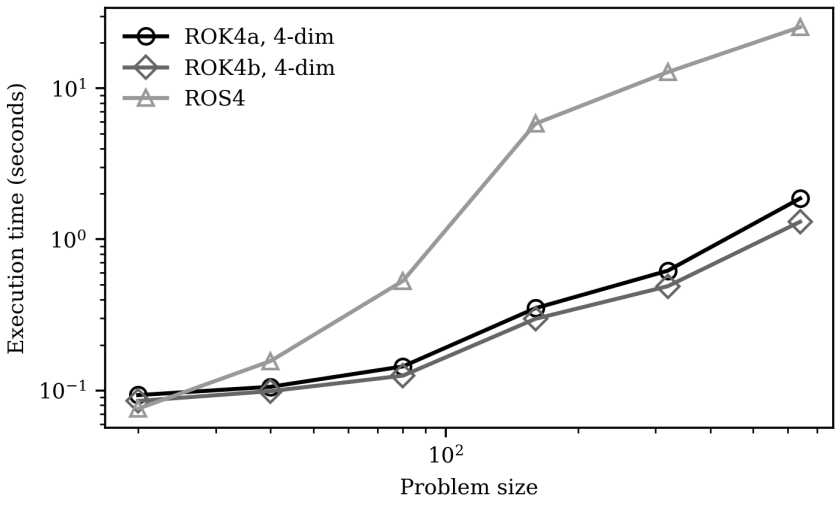

FIGURE 11: *Execution time versus problem size for the different methods when solving the Lorenz-96 model.*

8 Discussion and further research

For the author, what was gained during the process of writing this thesis was perhaps not the deepest possible understanding of Rosenbrock-Krylov methods, but rather insight into the implementation process of numerical methods. Implementations made in previous courses have been small and applied to relatively small problems. Implementing something more advanced, in such a way that it could potentially later be used by other people on "real" problems proved to be another beast entirely.

A not inconsiderable amount of time has been spent on trying to interpret "strange" results, hunting bugs and trying to figure out whether errors are due to mathematical or coding mistakes. This very clearly highlighted the amount of work needed to develop an implementation which feels robust enough that you would trust it with producing accurate results.

The results obtained in the benchmarking should not be considered the final verdict in the question of whether Rosenbrock-Krylov methods are computationally efficient enough. Even though both the Rosenbrock methods and the Rosenbrock-Krylov methods were implemented by the same person using the same programming language, little to no optimization was performed and as such there may be significant differences in performance due to suboptimal coding practices. However, it seems improbable that this would have so large impact that it cannot be said that Rosenbrock-Krylov methods have shown themselves to be efficient for large systems, compared to ordinary Rosenbrock methods. At worst, the smallest systems where the extra effort of the Arnoldi iteration is worth it might be larger than what was shown.

Repeating the benchmarks done in this thesis, but instead comparing the Rosenbrock-Krylov methods to some optimized Rosenbrock methods written in lower-level languages could really shine a light on the potential performance gains in "real life" situations.

The decision to construct general methods, taking the set of coefficients as an input, was somewhat double-edged. On the one hand, it made testing different methods very easy, since all one had to do was to change the supplied coefficients. On the other hand, it made it harder to implement "tricks" which might not work in the general case. As a concrete example, in order to implement adaptive step sizing for RK4, it would have been needed to add some flag to the method and (when that flag is present) reroute the flow through the code to a special version that supports adaptive step sizes. On a positive note, thanks to the generality of the code, when this setback was discovered implementing the six stage RKF45 method was a breeze.

Hairer and Wanner also categorize 32 different combinations of problem properties that each require their own code modifications in order for Rosenbrock methods to be as computationally efficient as possible [HW10, p. 113]. For a general implementation, this would possibly lead to large amounts of introspection followed by highly branching code.

Thus, an exhaustive general implementation for one family of methods risks turning into a labyrinthine piece of code with the data flowing through convoluted and less-than-obvious paths. This could be an issue in high-performance situations, since branching code makes for slower execution. The upside is the ease of use, both in implementing new methods and for the user to be able to simply hand the algorithm a problem and let it do its work, blissfully unaware of what happens "under the hood".

The Lorenz-96 model was chosen since it was used by Tranquilli and Sandu in their article, and as such would not prove to be an incredibly ill-suited problem for the Rosenbrock-Krylov methods. (There were quite enough of kinks to work out during the implementation process without having to worry about if they were caused by the test problem.) Now that there are working implementations of the methods, it would be interesting to test its performance on other problems. This would hopefully give more complete information on what types (and sizes) of problems are good candidates for solving using Rosenbrock-Krylov methods. By the results in sections 7.1 and 7.2, it also appears to be the case that RK4 performs better than the implicit methods on the Lorenz-96 model, which is not a "good look" if one attempts to highlight the advantages of implicit methods.

However, the Lorenz-96 model was not chosen for its stiffness (it apparently is not a very stiff problem), and on the stiff problems, the Rosenbrock-Krylov methods outperformed both explicit Runge-Kutta methods and the ordinary Rosenbrock method. An obvious future experiment is finding some large, stiff problem and test the performance of the methods on it. Given the results within this thesis, it does not seem unreasonable to anticipate that they would be satisfactory.

References

[Arn51] W. E. Arnoldi. "The principle of minimized iterations in the solution of the matrix eigenvalue problem". In: *Quarterly of Applied Mathematics* 9.1 (1951), pp. 17–29. DOI: https://doi.org/10.1090/qam/42792.

[AWF11] Heng-Bin An, Ju Wen, and Tao Feng. "On finite difference approximation of a matrix-vector product in the Jacobian-free Newton–Krylov method". In: *Journal of Computational and Applied Mathematics* 236 (6 2011), pp. 1399–1409. DOI: 10.1016/j.cam.2011.09.003.

[But87] J.C. Butcher. *The Numerical Analysis of Ordinary Differential Equations*. Chichester: John Wiley & Sons Ltd., 1987. ISBN: 0 471 91046 5.

[But96] J.C. Butcher. "A history of Runge-Kutta methods". In: *Applied Numerical Mathematics* 20.3 (1996), pp. 247–260. ISSN: 0168-9274. DOI: https://doi.org/10.1016/0168-9274(95)00108-5.

[Com17] Virginia Tech Computational Science Laboratory. *MATLODE/ROK_FWD_Integrator.m*. 2017. URL: https://web.archive.org/web/20210518071138/https://github.com/ComputationalScienceLaboratory/MATLODE/blob/master/Core_Method/CoreMethod_FWD/ROK_FWD_Integrator.m (visited on 05/18/2021).

[HNW08] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Berlin: Springer, 2008. ISBN: 978-3-540-56670-0.

[HW10] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Berlin: Springer, 2010. ISBN: 978-3-642-05220-0.

[PR74] A. Prothero and A. Robinson. "On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations". In: *Mathematics of Computation* 28.125 (Jan. 1974), pp. 145–145. DOI: 10.1090/s0025-5718-1974-0331793-2.

[Ros63] H. H. Rosenbrock. "Some general implicit processes for the numerical solution of differential equations". In: *The Computer Journal* 5 (4 1963), pp. 329–330. DOI: 10.1093/comjnl/5.4.329.

[Saa03] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003. ISBN: 9780898715347.

[Sha94] Lawrence F. Shampine. *Numerical Solution of Ordinary Differential Equations*. New York: Chapman & Hall, 1994. ISBN: 0-412-05151-6.

[ST07] L. F. Shampine and S. Thompson. "Stiff systems". In: *Scholarpedia* 2.3 (2007). revision #139228, p. 2855. DOI: 10.4249/scholarpedia.2855.

[TS14] P. Tranquilli and A. Sandu. "Rosenbrock-Krylov Methods for Large Systems of Differential Equations". In: *SIAM Journal on Scientific Computing* 36.3 (2014), A1313–A1338. DOI: 10.1137/130923336.

Appendices

A Method coefficients

$\gamma = 0.572816062482135$			
α_{21}	1.14563212496427	γ_{21}	−2.34199314019306
α_{31}	0.520920789953609	γ_{31}	−2.71665784065074
α_{32}	0.134294187208862	γ_{32}	−0.844109972094621
α_{41}	0.520920789953609	γ_{41}	−0.487777398284488
α_{42}	0.134294187208862	γ_{42}	−0.301763622478305
α_{43}	0.0	γ_{43}	0.111830332072784
b_1	0.324534708546765	$\hat{b}_1$	−0.0782106957370679
b_2	0.0490865433683549	$\hat{b}_2$	−0.146687782471748
b_3	0.0	$\hat{b}_3$	0.0765689455763802
b_4	0.626378748084880	$\hat{b}_4$	1.14832953263244

Table 7: *Coefficients for the Rosenbrock method ROS4*

$\gamma = 0.572816062482135$			
α_{21}	1	γ_{21}	−1.91153192976055097824
α_{31}	0.10845300169319391758	γ_{31}	0.32881824061153522156
α_{32}	0.39154699830680608241	γ_{32}	0
α_{41}	0.43453047756004477624	γ_{41}	0.03303644239795811290
α_{42}	0.14484349252001492541	γ_{42}	−0.24375152376108235312
α_{43}	−0.07937397008005970166	γ_{43}	−0.17062602991994029834
b_1	0.16666666666666666667	$\hat{b}_1$	0.50269322573684235345
b_2	0.16666666666666666667	$\hat{b}_2$	0.27867551969005856226
b_3	0	$\hat{b}_3$	0.21863125457309908428
b_4	0.66666666666666666667	$\hat{b}_4$	0

Table 8: *Coefficients for the Rosenbrock-Krylov method ROK4a, as given in [TS14]*

$\gamma = 0.31$			
α_{21}	1	γ_{21}	-22.824608269858540
α_{31}	0.530633333333333	γ_{31}	-69.343635255712726
α_{32}	-0.030633333333333	γ_{32}	-0.030633333333333
α_{41}	0.894444444444444	γ_{41}	404.7106882480958
α_{42}	0.055555555555556	γ_{42}	0.055555555555556
α_{43}	0.05	γ_{43}	0.05
α_{51}	0.738333333333333	γ_{51}	-0.571666666666667
α_{52}	-0.121666666666667	γ_{52}	-0.121666666666667
α_{53}	0.333333333333333	γ_{53}	0.333333333333333
α_{54}	0.05	γ_{54}	0.05
α_{61}	-0.096929102925711	γ_{61}	0.263595769492377
α_{62}	-0.121666666666667	γ_{62}	-0.121666666666667
α_{63}	1.045582889789120	γ_{63}	-0.378916223122453
α_{64}	0.173012879703258	γ_{64}	-0.073012879703258
α_{65}	0	γ_{65}	0
b_1	0.16666666666666666667	$\hat{b}_1$	0.16666666666666666667
b_2	-0.24333333333333333333	$\hat{b}_2$	-0.24333333333333333333
b_3	0.66666666666666666667	$\hat{b}_3$	0.66666666666666666667
b_4	0.1	$\hat{b}_4$	0.1
b_5	0	$\hat{b}_5$	0.31
b_6	0.31	$\hat{b}_6$	0

TABLE 9: *Coefficients for the Rosenbrock-Krylov method ROK4b, as given in [TS14]*

Bachelor's Theses in Mathematical Sciences 2021:K24
ISSN 1654-6229

LUNFNA-4036-2021

Numerical Analysis
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/