University of Heidelberg
Institute of Computer Science

Bachelor Thesis

Comparison of Methods for Integrating Linear Assignment Flows

Name:	Fabian Schneider
Supervisor:	Christoph Schnörr
Advisor:	Alexander Zeilmann
Date of Submission:	1st October 2021

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Sowohl inhaltlich als auch wörtlich entnommene Inhalte wurden als solche kenntlich gemacht. Die Arbeit ist in gleicher oder vergleichbarer Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Heidelberg, den 01. Oktober 2021

Zusammenfassung

In moderner Computer Vision ist das Segmentieren von Bildern ein stark erforschtes Anwendungsgebiet, für das viele verschiedene Algorithmen und Ansätze entwickelt wurden. Zeilmann et al. (2020) haben vor kurzem einen neuen Ansatz veröffentlicht: **Linear Assignment Flows**, bei denen jedoch ein sehr großes, mit unseren Bilddaten versehenes, System von linearen Gewöhnlichen Differentialgleichungen gelöst werden muss, was je nach Methode aber sehr viel Zeit in Anspruch nehmen kann. In dieser Arbeit werden wir deshalb zwei mögliche Methoden vergleichen: Das explizite Euler Verfahren und das Krylovraum Verfahren. Sowohl der Berechnungsaufwand als auch die Fehleranfälligkeit werden in Abhängigkeit von Ihren Verfahrensparametern, der Schrittweite (Euler) und der Dimension des Krylovraums (Krylov) untersucht.
Hierbei hat sich gezeigt, dass sowohl zeitlich als auch qualitativ die Ergebnisse des Krylovraum Verfahrens die des Euler Verfahrens deutlich übertreffen. Mit ihr ist es möglich selbst große Bilder in sehr kurzer Zeit zu verarbeiten.

Abstract

In modern computer vision, image segmentation is a heavily researched application area for which many different algorithms and approaches have been developed. Zeilmann et al. (2020) have recently published a new approach: **Linear Assignment Flows**, which however require solving a very large system of linear ordinary differential equations provided with our image data, but this can take a long time depending on the solving method. In this paper, we will therefore compare two possible methods: The explicit Euler method and the Krylov subspace method. Both the computational effort and the error-proneness are investigated depending on the method parameters, step size (Euler) and the dimension of the Krylov space (Krylov).
It has been shown that both temporally and qualitatively the results of the Krylov space method clearly surpass those of the Euler method. With it, it is possible to process even large images in a very short time.

Notation

ODE	Ordinary Differential Equation
EEM	Explicit Euler Method
KSM	Krylov Subspace Method
LAF	Linear Assignment Flow
i_1, i_2	height and width of an image
I	$i_1 \cdot i_2$, Number of pixels in the image,
J	Number of labels
$\mathbb{I}$	Identity Matrix
$\mathbb{1}$	Matrix of ones
$\mathbb{0}$	Matrix of zeros
$\bigotimes$	Kronecker Product
$\|\cdot\|$	Euclidean Norm for vectors, spectral norm for matrices

Contents

1 Introduction

1.1 Motivation

Our world is becoming more and more digital and our society relies more and more on the work of computers. However, these computers must be able to recognize and understand the world around them in order to function correctly. The key to this is **Image Labeling** - where the individual components of an image are recognized and, if necessary, assigned to a group of objects. One of the most prominent, but also critical applications for this at present is autonomous driving. In order to ensure the safety of the passengers, all images from all cameras attached to the car must be segmented and analyzed several times per second. This cannot be done with high computing power alone; an efficient algorithm is needed that is capable of segmenting the images in real time and doing so with the highest possible accuracy, since any error, no matter how small, could put people in danger.

1.2 Objectives & Structure

The objective of this thesis is to evaluate the error and time complexity of both the Explicit Euler Method and the Krylov Subspace Method, depending on their respective parameters *step size* and *subspace dimension*, so we can assess their performance in a direct comparison. In the following chapter 2 we will introduce some basic concepts, the reader needs to be familiar with to understand the discussed topics. Among other things, we will explain the Linear Assignment Flow, how to solve it with our respective methods and where the challenges lie. In chapter 3 we begin our comparison by first analyzing the theoretical error and then approximating the time complexity, which is followed by an experimental evaluation of our theorems in chapter 4. In the end we will conclude our work and give an outlook onto possible improvements and future work.

2 Background and Related Work

In the following chapter, we will explain what systems of linear *ODE*s, especially the ones produced by the Linear Assignment Flow, are and how we will solve them, to label our images.

2.1 Ordinary Differential Equations

Definition 2.1.1 (Implicit ordinary differential equation)**.** *Generally, an ordinary differential equation of order k takes the form*

$$F(t, x, x^1, ..., x^k) = 0 \tag{2.1}$$

where

- $k \in \mathbb{N}_0$ *is the order of the highest derivative*
- $x \in C^k(J), J \subseteq \mathbb{R}$ *is unknown, also called* ***dependent*** *variable*
- $x^1, ..., x^k$ *are the derivatives* $x^j(t) = \frac{d^j x(t)}{dt^j}, j \in \mathbb{N}_0$ *of* x
- t *is an* ***independent*** *variable*
- $F \in C^0(U), U$ *open subset of* $\mathbb{R}^{k+2}$*.*

Definition 2.1.2 (Explicit ordinary differential equation)**.** *An implicit ODE solved for its highest derivative.*

$$F(t, x, x^1, ..., x^{k-1}) = x^k \tag{2.2}$$

In the following chapters we will exclusively use the explicit form.
A **system** of ODEs is formed when the function x becomes an n-dimensional vector of functions:

$$\begin{pmatrix} f_1(t, x, x^1, ..., x^{k-1}) \\ \vdots \\ f_n(t, x, x^1, ..., x^{k-1}) \end{pmatrix} = \begin{pmatrix} x_1^k \\ \vdots \\ x_n^k \end{pmatrix} \tag{2.3}$$

Additionally, an *ODE* can have the following characteristics:
Linearity : A system is called linear, if it can be represented as

$$x_i^k = b_i(t) + \sum_{l=1}^{n} \sum_{j=0}^{k-1} f_{i,j,l}(t) x_l^j, \tag{2.4}$$

respectively F can be written as a linear combination of all given derivatives.
Homogeneity : A system is homogeneous, if

$$b_i(t) \equiv 0 \tag{2.5}$$

Autonomy : A system is autonomous, if it's independent of t.

In this thesis, we will focus on systems of linear, autonomous first-order *ODE*s, which gives us the opportunity to simplify its form. Most importantly the first-order property allows us to abandon all derivatives higher than 1, in addition to the linearity we can use a Jacobian to pack all the coefficients of x into a matrix A and make the system a matrix-vector multiplication. We add b, if the system is inhomogeneous, which finally results in

$$\begin{aligned} \dot{x} &= f(x) \\ &= A\,x + b. \end{aligned} \tag{2.6}$$

2.2 Linear Assignment Flow

The Linear Assignment Flow (Zeilmann et al., 2020) is represented by an *ODE* on the tangent space

$$\mathcal{T}_0 = \{T \in \mathbb{R}^{I \times J} : T\mathbb{1} = \mathbb{0}\} \tag{2.7}$$

at the barycenter of the assignment Manifold

$$\mathcal{W} = \{W \in \mathbb{R}^{I \times J} : W_{i,j} > 0, W\mathbb{1} = \mathbb{1}\} \tag{2.8}$$

meaning that every row of $W \in \mathcal{W}$ represents a pixel and every column a label. The resulting entries then represent the probability of pixel i being mapped to label j, so that every row sums up to 1.
Our *ODE* will have the form

$$\dot{V}(t) = A\,V(t) + b, \quad V(0) = 0 \in \mathbb{R}^{IJ}, \tag{2.9}$$

where

- $V(t) \in \mathcal{T}_0$ is our tangent vector

$$V(t) = \begin{pmatrix} label_{1,1} \\ \vdots \\ label_{1,J} \\ \\ \vdots \\ \\ label_{I,1} \\ \vdots \\ label_{I,J} \end{pmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} a \\ \vdots \\ a \end{matrix}}\right\}\text{Pixel 1} \\ \\ \\ \\ \left.\vphantom{\begin{matrix} a \\ \vdots \\ a \end{matrix}}\right\}\text{Pixel I} \end{matrix} \qquad \sum_{j=1}^{J} label_{i,j} = 0, \quad i \in [1, I] \tag{2.10}$$

- $A \in \mathbb{R}^{IJ \times IJ}$ is the Kronecker Product of the block-circulant matrix Ω and the identity matrix of size $J \times J$

$$A = \Omega \otimes \mathbb{I}_J. \tag{2.11}$$

$\Omega \in \mathbb{R}^{I \times I}$ is a 2D-Convolution with the $\frac{1}{9}$-uniform-weights kernel ("box blur") h

$$h = \begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}, \tag{2.12}$$

constructed by

$$\Omega = \frac{1}{i_1 i_2} W \, Diag(vec(\hat{h}')) \, \overline{W}^T, \quad W = W_1 \otimes W_2, \tag{2.13}$$

with the Fourier Matrix W_n of size i_n and the Fourier Transform of (i_1, i_2)-periodic h, $\hat{h}$.

- $b \in \mathbb{R}^{IJ}$ carries the data of our labeling problem, as it is a vectorized version of a distance matrix $D \in \mathbb{R}^{I \times J}$ whose entries are the euclidean distance between the color of every pixel and the color of every prototype

$$\|D_{ij} = pixel - prototype\|_2. \tag{2.14}$$

  It's constructed by projecting $-D$ to the tangent space and then vectorizing it rowwise.

For a standard sized image of 500×500 pixels and 5 prototypes, the system will result in the dimensions

$$\mathbb{R}^{1\,250\,000} = \mathbb{R}^{1\,250\,000 \times 1\,250\,000} \cdot \mathbb{R}^{1\,250\,000}, \tag{2.15}$$

Which naively implemented, not only leads to an infeasible computation time, but also memory usage. If one entry takes 8 bytes, only A would need $8 \cdot 1\,250\,000^2$ bytes = 12.5 TB. Luckily our Matrix A of the mentioned dimensions has a sparsity of 99.99928%, which saves a lot of time when the right algorithm is used and takes up only 180MB (90MB for the data and 90MB for the coordinates).

2.3 Matrix Exponential and φ-Functions

Now that we know the system that we are working with, we need to understand how to solve it. For simplicity we begin with a homogeneous *ODE* and its initial value

$$\dot{x}(t) = Ax(t), \quad x(0) = x_0. \tag{2.16}$$

Definition 2.3.1 (Matrix Exponential)**.** *For a matrix $A \in \mathbb{R}^{n \times n}$ we define the matrix exponential to be the $n \times n$ matrix*

$$e^A = \sum_{j=0}^{\infty} \frac{A^j}{j!}.$$

Theorem 2.3.1. *The solution of a homogeneous ODE* (2.16) *is*

$$x(t) = e^{tA}x_0.$$

Proof. (Teschl, 2004, pp. 59-60) We use the Picard Iteration

$$\begin{aligned} x_0(t) &= x_0 \\ x_{i+1}(t) &= x_0 + \int_{t_0}^{t} f(s, x_i(s))ds, \quad t \in [t_0, t_0 + \epsilon] \end{aligned}$$

to approximate the solution

$$\begin{aligned} x_0(t) &= x_0 \\ x_1(t) &= x_0 + \int_0^t Ax_0(s)ds = x_0 + Ax_0 \int_0^t ds = x_0 + tAx_0 \\ x_2(t) &= x_0 + \int_0^t Ax_1(s)ds = x_0 + \int_0^t A(x_0 + sAx_0)ds \\ &= x_0 + \int_0^t Ax_0 ds + \int_0^t sA^2x_0 ds \\ &= x_0 + tAx_0 + \frac{t^2}{2}A^2x_0. \end{aligned}$$

By induction we can generalize the solution to

$$x_m(t) = \sum_{j=0}^{m} \frac{t^j}{j!}A^j x_0 \tag{2.17}$$

and further to

$$x(t) = \lim_{m\to\infty} x_m(t) = \sum_{j=0}^{\infty} \frac{t^j}{j!}A^j x_0. \tag{2.18}$$

By now plugging in the power series for the exponential function

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \tag{2.19}$$

we see that our unknown function corresponds to

$$x(t) = e^{tA}x_0. \tag{2.20}$$

□

Even though the Picard Iteration guarantees, that $e^A x_0$ will converge (Teschl, 2004, p. 60), we don't know how fast this will happen, so naively calculating the matrix exponential is not computationally feasible. There are plenty of ways for calculating the matrix exponential (Moler and Loan, 2003), that all have certain advantages or disadvantages, depending on the matrix A, but before looking at them, we solve the inhomogeneous *ODE*

$$\dot{x}(t) = Ax(t) + b, \quad x(0) = x_0,\ b \text{ const.} \tag{2.21}$$

Definition 2.3.2 (φ-Functions)**.** *The φ-Functions represent special cases of the* ***Mittag-Leffler Function*** *(Haubold et al., 2011). We define $\varphi_{\beta-1}(z)$ as $E_{1,\beta}(z)$ for*

$$\begin{aligned} E_{1,\beta}(z) &= \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(k+\beta)} \\ &= \sum_{k=0}^{\infty} \frac{z^k}{(k+\beta-1)!}, \quad \beta \in \mathbb{N}_+. \end{aligned}$$

Especially the cases of $\beta = 1$ and $\beta = 2$ will be helpful:
$E_{1,1} = \varphi_0$:

$$\varphi_0(z) = e^z, \quad \varphi_0(A) = e^A \tag{2.22}$$

$E_{1,2} = \varphi_1$:

$$\varphi_1(z) = \frac{e^z - 1}{z}, \quad \varphi_1(A) = \frac{e^A - \mathbb{I}}{A} \tag{2.23}$$

Theorem 2.3.2. *The solution of an inhomogeneous* ODE *is given by*

$$x(t) = \varphi_0(tA)x_0 + t \cdot \varphi_1(tA) \cdot b$$

respectively

$$x(t) = t \cdot \varphi_1(tA) \cdot b \tag{2.24}$$

if $x_0 = 0$.

Proof. Under the assumption that A is invertible, we begin by applying **Duhamel's Formula** to our system (2.21)

$$\begin{aligned} x(t) &= e^{tA}x_0 + \int_0^t e^{(t-s)A}b(s)ds \\ &= e^{tA}x_0 + b \cdot \int_0^t e^{(t-s)A}ds \\ &= e^{tA}x_0 - b \cdot A^{-1} \int_0^t e^{u}du \\ &= e^{tA}x_0 - b \cdot A^{-1} \cdot [e^u]_0^t \\ &= e^{tA}x_0 - b \cdot A^{-1} \cdot [e^{(t-s)A}]_0^t \\ &= e^{tA}x_0 + b \cdot A^{-1} \cdot (e^{At} - \mathbb{I}) \end{aligned} \tag{2.25}$$

We can now replace parts of (2.25) with the φ-Functions.

$$\begin{aligned} x(t) &= e^{tA}x_0 + b \cdot A^{-1} \cdot (e^{At} - \mathbb{I}) \\ &= \varphi_0(tA)x_0 + b \cdot t \cdot (tA)^{-1} \cdot (e^{At} - \mathbb{I}) \\ &= \varphi_0(tA)x_0 + t \cdot \varphi_1(tA) \cdot b \end{aligned} \tag{2.26}$$

If we additionally have the constraint that $x_0 = 0$, we can simplify the formula to

$$x(t) = t \cdot \varphi_1(tA) \cdot b. \tag{2.27}$$

□

Now that we have formulas to solve homogeneous as well as inhomogeneous systems, there's still the open question of how to calculate the matrix exponential. In the following, we will take a look at some possible methods.

Taylor Series(Moler and Loan, 2003, pp. 9-11)
Of course, it's possible to just calculate the matrix exponential naively by (2.3.1), but that's neither efficient/possible for big matrices, nor very reliable in terms of accuracy. That's because the computational cost and the error of this method grow together with the spectral norm of A.

Matrix Decomposition(Moler and Loan, 2003, pp. 22-29)
By splitting A into easier to compute parts, we can drastically improve the computation

time and error. This is possible for example with the **Jordan Decomposition**

$$A = PJP^{-1} \tag{2.28}$$

$$J = \begin{pmatrix} J_1 & & \\ & \ddots & \\ & & J_m \end{pmatrix} \tag{2.29}$$

$$J_n = \begin{pmatrix} \alpha & 1 & & \\ & \alpha & \ddots & \\ & & \ddots & 1 \\ & & & \alpha \end{pmatrix} \tag{2.30}$$

with α being the eigenvalues of A. If we now consider this when calculating A^j

$$A^j = (PJP^{-1})^j, \tag{2.31}$$

we see that most of P and P^{-1} cancel each other out to

$$A^j = PJ^jP^{-1}. \tag{2.32}$$

This can now be inserted into our definition of the matrix exponential (2.3.1).

$$e^A = \sum_{j=0}^{\infty} \frac{PJ^jP^{-1}}{j!} = P\left(\sum_{j=0}^{\infty} \frac{J^j}{j!}\right)P^{-1} = Pe^JP^{-1} \tag{2.33}$$

Finally the last thing to do is to compute e^J, which is very easy due to the properties of J

$$e^J = \begin{pmatrix} e^{J_1} & & \\ & \ddots & \\ & & e^{J_m} \end{pmatrix}$$

$$e^J = e^\alpha \begin{pmatrix} 1 & 1 & \frac{1}{2!} & \dots & \frac{1}{(k-1)!} \\ & 1 & 1 & \ddots & \vdots \\ & & 1 & \ddots & \frac{1}{2!} \\ & & & \ddots & 1 \\ & & & & 1 \end{pmatrix} \tag{2.34}$$

So in general the problem of calculating the exponential of a matrix comes down to cal-

culating the exponential of some scalars and multiplying this to an easy to form matrix. Sadly the Jordan Decomposition is not stable for floating point operations, but there are several of other decompositions like Schur, Block-Diagonal, etc. that come with advantages and disadvantages for different matrices A.

Padé approximation + Scaling and Squaring(Moler and Loan, 2003, pp. 11-14) The method mostly used in modern libraries like Matlab and Scipy, is a mixture between the so called Padé approximation and the Scaling and Squaring Method. The latter of those exploits the fact that

$$e^A = e^{(A/n)^n} \tag{2.35}$$

and

$$\|A/n\| = \|A\|/n, \tag{2.36}$$

so that we can reduce the spectral radius of A to a point, where we don't have to worry about errors when calculating the exponential. Normally the divisor n is a power of 2 that brings our spectral radius under 1. Even though this allows us to even compute the Taylor Series Method without problems, mostly the Padé Approximation is preferred. This one is computed by

$$e^A = R_{pq}(A) = [D_{pq}(A)]^{-1} N_{pq}(A) \tag{2.37}$$

$$\begin{aligned} N_{pq}(A) &= \sum_{j=0}^{p} \frac{(p+q-j)!p!}{(p+q)!j!(p-j)!} A^j \\ D_{pq}(A) &= \sum_{j=0}^{q} \frac{(p+q-j)!q!}{(p+q)!j!(q-j)!} (-A)^j \end{aligned} \tag{2.38}$$

which is very efficient because usually p and q are chosen quite small. Even though it has the same problem as the Taylor Series, that a high $\|A\|$ makes it inaccurate, this is can be corrected with Scaling and Squaring. The p and q we choose depends on the spectral radius of A and the accuracy we want to achieve, but mostly very small values are used, like $(6, 6)$ in Matlab. If $p = q$, we call this method diagonal Padé approximation.

2.4 Explicit Euler Method

The Explicit Euler Method (also called "forward Euler") is the most basic method to solve an *ODE*. The idea behind it is, that starting from our initial value $x_0 = x(t_0)$, we calculate the direction the system is pointing to at our position, then move into that direction for a certain time and then repeat these steps.
Mathematically we can derive it from the **Taylor expansion**

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \dots$$

by calculating our function $x(t_0+h)$ around t_0 and omitting the second order term.

$$\begin{aligned} x(t_0+h) &= x(t_0) + \frac{\dot{x}(t_0)}{1!}(t_0+h-t_0) + \frac{\ddot{x}(t_0)}{2!}(t_0+h-t_0)^2 + \dots \\ &= x(t) + \dot{x}(t_0)h + \mathcal{O}(h^2) \end{aligned} \tag{2.39}$$

If we now substitute for our definition of the first derivative (2.6) we get the full formula for the *EEM*

$$x(t_0+h) = x(t_0) + (Ax(t_0) + b)h. \tag{2.40}$$

But to just use this function once and jump directly to the t we want, would not make a lot of sense. That's why we define the iterative method

$$a_{n+1} = a_n + (Aa_n + b)h. \tag{2.41}$$

Example:
Even though we will use this method for very big systems later, it's very convenient to show the *EEM* in the 2D-space.
We define

$$A = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad x_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{2.42}$$

With an errorless integration method, this system stay on the unit circle. To achieve a mostly stable circle with the *EEM* however, we have to use a very small step size h.

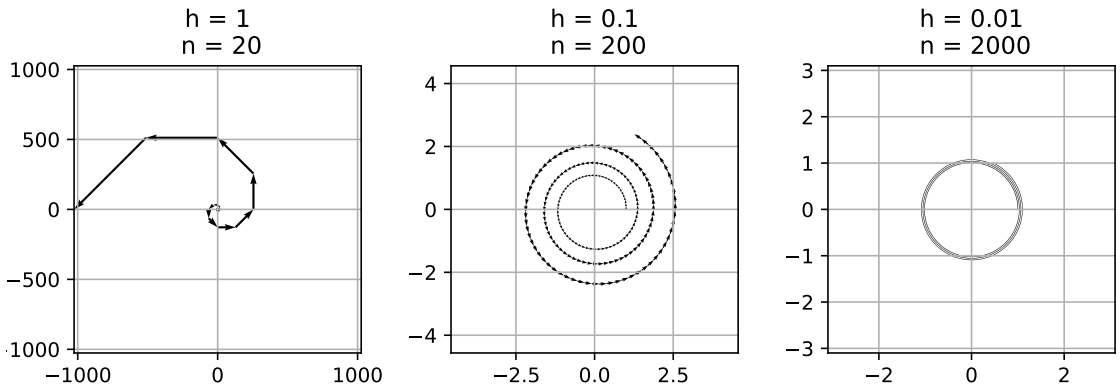


Figure 2.1: The system (2.42) integrated with three different step sizes up to $t = 20$.

2.5 Krylov Subspace Method

The Krylov Subspace Method (Niesen and Wright, 2012, pp. 4-6) is a more advanced method than the *EEM*. It tries to approximate $\varphi_p(A)x$ which lives in $\mathbb{R}^n$ by using the Krylov Subspace

$$\mathcal{K}_m(A, x) = \text{span}\{x, Ax, A^2x, \ldots, A^{m-1}x\}, \quad m \leq n. \tag{2.43}$$

This subspace has the problem, that the power iteration will converge to the eigenvector associated with the dominant eigenvalue, resulting in almost linear dependent basis vectors, the bigger m becomes. By using the **Gram-Schmidt** procedure though, we can produce the orthonormal subspace

$$\mathcal{K}_m(A, x) = \text{span}\{v_1, v_2, \ldots, v_m\}. \tag{2.44}$$

These basis vectors combined, form the matrix $V_m \in \mathbb{R}^{n \times m}$ such that

$$AV_m = V_m H_m + h_{m,m+1} v_{m+1} e_m^T \tag{2.45}$$

respectively

$$H_m = V_m^T A V_m, \quad A \approx V_m H_m V_m^T \tag{2.46}$$

where e_m is the mth vector in the standard basis and H_m is a **Hessenberg** matrix, meaning that its entries h_{ij} are 0 if $i > j + 1$. One algorithm to split A into these parts is the **Arnoldi iteration** (1), which first starts to build V_m by normalizing our vector x from (2.43) and iteratively appending orthonormal vectors to it. In the special

Algorithm 1 Arnoldi Iteration (Niesen and Wright, 2012, p. 5)

Require: Matrix **A**, Vector **b**, number of dimensions **m**

$H \leftarrow 0 \in \mathbb{R}^{m \times m}$
$V \leftarrow 0 \in \mathbb{R}^{IJ \times m}$
$v_1 \leftarrow b/\|b\|$ ▷ v columns of V, h entries of H
for $j \leftarrow 1, \ldots, m$ **do**
    $w \leftarrow Av_j$
    for $i \leftarrow 1, \ldots j$ **do**
        $h_{i,j} \leftarrow v_i^T w$
        $w \leftarrow w - h_{i,j} v_i$
    end for
    $h_{j+1,j} \leftarrow \|w\|$
    $v_{j+1} \leftarrow w/h_{j,j-1}$
end for
return $V, H, v_{m+1}, h_{m,m+1}$

case, that A is symmetric, our H_m will be symmetric too, which in addition with the Hessenberg form makes it tridiagonal and reduces the Arnoldi iteration to the **Lanczos iteration**(2), which uses a for-loop less.

The actual advantage of decomposing A into $V_m H_m V_m^T$ becomes obvious when we substitute it in the φ-Functions.

$$\varphi_p(A)x \approx \varphi_p(V_m H_m V_m^T)x = V_m \varphi_p(H_m) V_m^T x \tag{2.47}$$

Additionally $V_m^T x = \|x\| e_1$ where e_1 is the first vector in the standard basis, which results in

$$V_m \varphi_p(H_m) V_m^T x = V_m \varphi_p(H_m) \|x\| e_1. \tag{2.48}$$

This means that from now on, we don't have to calculate φ_p for an enormous matrix $\mathbb{R}^{IJ \times IJ}$ anymore, but for one that is of small size $m \times m$.
Finally, there's an additional trick, we can use to calculate $\varphi_p(H_m)e_1$ easier. The matrix exponential of a slightly augmented matrix

$$\hat{H}_m = \begin{pmatrix} H_m & e_1 & 0 \\ 0 & 0 & \mathbb{I} \\ 0 & 0 & 0 \end{pmatrix} \begin{matrix} m \\ p-1 \\ 1 \end{matrix} \tag{2.49}$$

$e^{\hat{H}_m}$ contains $\varphi_p(H_m)e_1$ in the first m entries of its last column. This is especially useful if φ_p has to be calculated for a lot of different p, because we only need to calculate the

Algorithm 2 Lanczos Iteration (Niesen and Wright, 2012, p. 5)

Require: Matrix **A**, Vector **b**, number of dimensions **m**

$H \leftarrow 0 \in \mathbb{R}^{m \times m}$
$V \leftarrow 0 \in \mathbb{R}^{IJ \times m}$
$v_0 \leftarrow 0$ ▷ Not part of the actual matrix
$h_{1,0} \leftarrow 0$ ▷ Not part of the actual matrix
$v_1 \leftarrow b/\|b\|$
for $j \leftarrow 1, \ldots, m$ **do**
    $w \leftarrow Av_j$
    $h_{j,j} \leftarrow v_j^T w$
    if j = 1 **then**
        $w \leftarrow w - h_{j,j} v_j$
    else
        $w \leftarrow w - h_{j,j} v_j - h_{j,j-1} v_{j-1}$
    end if
    $h_{j+1,j}, h_{j,j+1} \leftarrow \|w\|$
    $v_{j+1} \leftarrow w/h_{j,j+1}$
end for
return $V, H, v_{m+1}, h_{m,m+1}$

matrix exponential once and can then find all the φ_l for $l \leq p$.

$$\varphi_0 \left(\begin{pmatrix} H_m & e_1 & 0 \\ 0 & 0 & \mathbb{I}_{p-1} \\ 0 & 0 & 0 \end{pmatrix} \right) = \begin{pmatrix} \varphi_{p-1}(H_m) & \varphi_p(H_m)e_1 \\ 0 & 1 \end{pmatrix} \tag{2.50}$$

3 Comparison

Now that we are familiar with all the fundamentals we will need, it is time to begin the actual topic of this thesis. The goal of this chapter is to find the best possible estimation or bound for the produced error and the required computational cost of our two methods.

3.1 Error Estimation

In this chapter, we will examine how accurate both methods are and what this accuracy depends on. We define the accuracy by the size of the produced error, the difference between the exact solution and the approximation, but there we differentiate between two different kinds of errors: local and global.

Definition 3.1.1 (Local and Global Error)**.** *(Führer and Schroll, 2001, pp. 101–102) When we define the approximation $a_n \approx x(t_n)$ of the system*

$$\dot{x} = f(x),\ x(0) = x_0,$$

then the local error

$$\hat{\epsilon}_{n+1} = |a_{n+1} - \hat{x}(t_{n+1})| \tag{3.1}$$

is the difference produced in one timestep, where $x(t_{n+1})$ is the solution of the system

$$\dot{\hat{x}} = f(\hat{x}),\ \hat{x}(t_n) = a_n \tag{3.2}$$

that starts at the approximation a_n. On the other hand the global error

$$\epsilon_n = |a_n - x(t_n)| \tag{3.3}$$

is the accumulation of all local errors at the end of our integration.

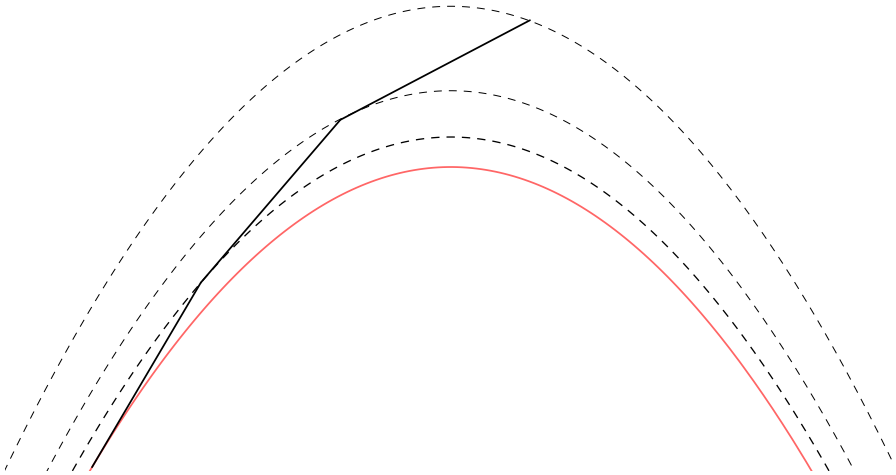


Figure 3.1: Explicit Euler Method
red: $\dot{x} = f(x),\ x(0) = x_0$
dashed: $\dot{\hat{x}} = f(\hat{x}),\ \hat{x}(t_n) = a_n$

3.1.1 Error Estimates for the Explicit Euler Integration

To get a feeling for how accurate the Explicit Euler Method is, we will start with a simple example:

$$A = \begin{pmatrix} -5 & 0 \\ 0 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad x_0 = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \tag{3.4}$$

So our system will be

$$\dot{x} = \begin{pmatrix} -5 & 0 \\ 0 & 2 \end{pmatrix} x, \quad x_0 = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \tag{3.5}$$

and we want to integrate it up to $t = 2$. Before we will apply the *EEM*, it makes sense to first calculate the exact solution, which is very easy because A is a diagonal matrix.

$$\begin{aligned} x(2) &= e^{2A}x_0 \\ &= e^{\begin{pmatrix} -10 & 0 \\ 0 & 4 \end{pmatrix}} x_0 \\ &= \begin{pmatrix} e^{-10} & 0 \\ 0 & e^4 \end{pmatrix} x_0 \\ &\approx \begin{pmatrix} 0 \\ 54 \end{pmatrix} \end{aligned}$$

Now we can take a look at the *EEM* with step size h:

$$\begin{aligned} a_{n+1} &= a_n + Aa_nh \\ &= a_n + \begin{pmatrix} -5 & 0 \\ 0 & 2 \end{pmatrix} a_nh \\ &= a_n \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} -5 & 0 \\ 0 & 2 \end{pmatrix} h \right) \end{aligned}$$

Case **h=1**

$$\begin{aligned} a_{n+1} &= a_n \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} -5 & 0 \\ 0 & 2 \end{pmatrix} \right) \\ &= a_n \begin{pmatrix} -4 & 0 \\ 0 & 3 \end{pmatrix} \\ a_n &= x_0 \begin{pmatrix} -4 & 0 \\ 0 & 3 \end{pmatrix}^n = x_0 \begin{pmatrix} (-4)^n & 0 \\ 0 & 3^n \end{pmatrix} \end{aligned}$$

So with $h = 1$ the solution is $x(2) \approx a_2 = \begin{pmatrix} 32 \\ 9 \end{pmatrix}$. One can see, that this won't be very accurate, because it alternates between the first and second quadrant of the coordinate system, diverging exponentially.

Case **h=0.1**

$$a_n = x_0 \begin{pmatrix} 0.5^n & 0 \\ 0 & 1.2 \end{pmatrix}$$

The solution with $h = 0.1$ is $x(2) \approx a_{20} = \begin{pmatrix} 0 \\ 38 \end{pmatrix}$. This is not perfect, but much better than the last case. Additionally we can see, that with this smaller step size, the solution won't alternate and diverge anymore.
If continued, the error ϵ, respectively the euclidean distance between the exact and approximated solution, would develop as follows:

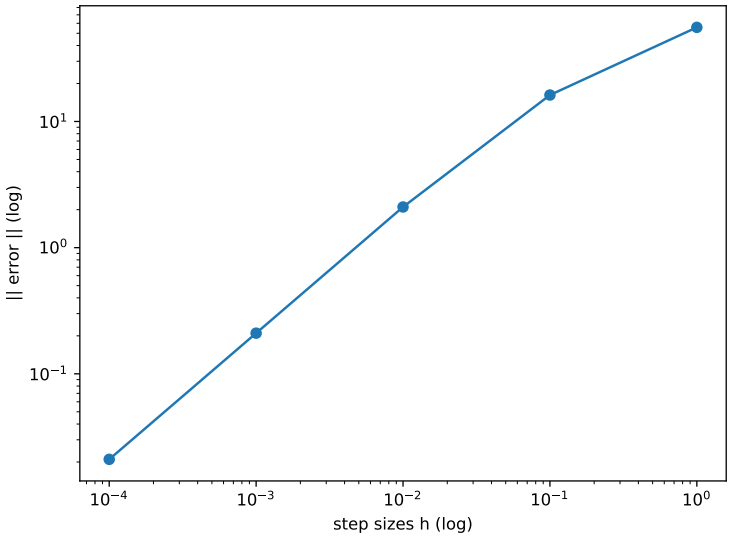


Figure 3.2: Step sizes and the corresponding errors for (3.5) at t=2

But more to these results later, after we examine the error theoretically.

Theorem 3.1.1. *The local error of the Explicit Euler Integration is*

$$\hat{\epsilon}_{n+1} = \frac{h^2}{2} |\ddot{\hat{x}}(t_n)|$$

where $\ddot{\hat{x}}(t_n)$ *is the second derivative of* (3.2).

Proof. (Führer and Schroll, 2001, p. 113) To calculate the local error, we need our approximation a at step $n + 1$

$$a_{n+1} = \hat{x}(t_n) + f(\hat{x}(t_n))h$$

and our solution $\hat{x}$ at time t_{n+1}, which we can define by the Taylor expansion around t_n

$$\begin{aligned}\hat{x}(t_{n+1}) &= \hat{x}(t_n) + (t_{n+1} - t_n)\dot{\hat{x}}(t_n) + (t_{n+1} - t_n)^2 \frac{\ddot{\hat{x}}(t_n)}{2} + \mathcal{O}(h^3) \\ &= \hat{x}(t_n) + h\dot{\hat{x}}(t_n) + h^2 \frac{\ddot{\hat{x}}(t_n)}{2} + \mathcal{O}(h^3) \\ &= \hat{x}(t_n) + hf(\hat{x}(t_n)) + h^2 \frac{\ddot{\hat{x}}(t_n)}{2} + \mathcal{O}(h^3).\end{aligned}$$

By substituting this in our formula for the local error (3.1), we get

$$\hat{\epsilon}_{n+1} = |h^2 \frac{\ddot{\hat{x}}(t_n)}{2} + \mathcal{O}(h^3)|, \tag{3.6}$$

where we will ignore everything higher than order two.

$$\hat{\epsilon}_{n+1} = \frac{h^2}{2} |\ddot{\hat{x}}(t_n)|$$

□

By this we can see that our local error is roughly proportional to h^2, respectively $\hat{\epsilon} \in \mathcal{O}(h^2)$, meaning that if we for example halve the step size, we will quarter the local error. But this is only true for small h, because for big ones, the higher order terms in (3.6) would start to gain importance.
The accumulation of this is the global error. Before we properly approximate it, we can already intuitively guess it: It's the local error of one step times the number of steps and because the number of steps is proportional to $\frac{1}{h}$ (when we halve h, we have to double the steps to reach the destination), our global error will be proportional to $\frac{h^2}{h} = h$.

Theorem 3.1.2. *The global error of the Explicit Euler Integration has an upper bound*

$$\|\epsilon_n\| \leq \frac{h \max \|\ddot{x}\|}{2} \frac{e^{t_n L_f} - 1}{L_f},$$

where L_f is the Lipschitz constant of f.

Proof. (Führer and Schroll, 2001, p. 114) To find a bound for the global error we

subtract the Euler Method and the true solution $x(t_{n+1})$:

$$\begin{aligned}
\epsilon_{n+1} &= a_{n+1} - x(t_{n+1}) \\
&= a_n + hf(a_n) - (x(t_n) + hf(x(t_n)) + h^2\frac{\ddot{x}(t_n)}{2} + \mathcal{O}(h^3)) \\
&= \epsilon_n + hf(a_n) - (hf(x(t_n)) + h^2\frac{\ddot{x}(t_n)}{2} + \mathcal{O}(h^3)) \\
&= \epsilon_n + hf(x(t_n + \epsilon_n)) - hf(x(t_n)) - h^2\frac{\ddot{x}(t_n)}{2} + \mathcal{O}(h^3) \\
&= \epsilon_n + hf(x(t_n + \epsilon_n)) - hf(x(t_n)) - e_{n+1}. \qquad (3.7)
\end{aligned}$$

e_{n+1} is the *global error increment*, that is, like the local error, in $\mathcal{O}(h^2)$. By now assuming *Lipschitz continuity* of f we get

$$\|\epsilon_{n+1}\| \leq \|\epsilon_n\| + hL_f\|\epsilon_n\| + \|e_{n+1}\|.$$

Now we can use *Grönwalls Inequality*

$$u_n \leq \frac{b}{h}\frac{e^{t_n\mu} - 1}{\mu}$$

for $u_{n+1} \leq (1 + h\mu)a_n + b$ with $h, \mu, b > 0, u_0 = 0$ and $t_n = nh$ to get

$$\|\epsilon_n\| \leq \frac{\|e_{n+1}\|}{h}\frac{e^{t_nL_f} - 1}{L_f} \qquad (3.8)$$

$$\leq \frac{h\ \max\|\ddot{x}\|}{2}\frac{e^{t_nL_f} - 1}{L_f} \qquad (3.9)$$

□

By that we can finally see, that our assumption was correct and that the global error is proportional to h, respectively $\|\epsilon_n\| \in \mathcal{O}(h)$, for $h \to 0$. Because of that the *EEM* is called a first-order Method. It also confirms our results from 3.2: When we took the step size times $\frac{1}{10}$, the error also reduced by that factor.

3.1.2 Error Estimates for the Krylov Subspace Method

As the *KSM* has the advantage of not having to iteratively integrate up to t, but rather "jumps" there directly, we won't have a local but just a global error here. Instead we will distinguish between the a priori and a posteriori error. But before looking at any

of them we will again look at our approximation

$$\varphi_p(A)x \approx \beta V_m \varphi_p(H_m) e_1, \quad \beta = \|x\|, \tag{3.10}$$

as we can improve it a bit, using v_{m+1} and $h_{m,m+1}$, that were produced in the last step of the Arnoldi/Lanczos iteration. This corrected scheme

$$\varphi_p(A)x \approx \beta V_m \varphi_p(H_m) e_1 + \beta h_{m,m+1} e_m^T \varphi_{p+1}(H_m) e_1 v_{m+1} \tag{3.11}$$

was first defined and proofed by Saad (1992)[p. 4] for $p = 0$ and then generalized for all $p > 0$ by Sidje (1998)[Th. 2]. It will improve the solutions, while barely increasing the computational cost, because v_{m+1} and $h_{m,m+1}$ are already computed and φ_{p+1} can easily be obtained from the exponential of on augmented matrix (2.50). We could also improve the corrector further with the these following formulas by Sidje (1998)

$$\varphi_p(A)v - \beta V_m \varphi_p(H_m) e_1 = \beta \sum_{j=p+1}^{\infty} h_{m,m+1} e_m^T \varphi_j(H_m) e_1 A^{j-p-1} v_{m+1} \tag{3.12}$$

$$\tau^p \varphi_p(\tau A)v - \tau^p \beta V_m \varphi_p(\tau H_m) e_1 = \beta \sum_{j=p+1}^{\infty} h_{m,m+1} \tau^j e_m^T \varphi_j(\tau H_m) e_1 A^{j-p-1} v_{m+1} \tag{3.13}$$

but this would involve computations of matrix powers.
So for our standard inhomogeneous ODE system the corrected approximated solution is

$$\begin{aligned} x(t) &= t\, \varphi_1(tA)\, b \\ &\approx t\, (\beta V_m \varphi_1(tH_m) e_1 + \beta h_{m,m+1} t e_m^T \varphi_2(tH_m) e_1 v_{m+1}). \end{aligned} \tag{3.14}$$

In the future, we call (3.10) the basic scheme and (3.11) the corrected scheme.
Now we can begin with the approximation of the a priori error, which was also defined by Saad (1992). One will see that the approximation is actually for φ_0 and not for φ_1, like we need it in the solution of our inhomogeneous system, but we can transfer the results, as φ_0 and φ_1 converge equally fast (Hochbruck and Lubich, 1997, p. 2).

Theorem 3.1.3. *The error of the basic scheme for a matrix A, scaled by τ, and vector b is bounded by*

$$\epsilon_B = \|e^{\tau A} b - \beta V_m e^{\tau H_m} e_1\| \leq 2\beta \frac{(\tau\rho)^m e^{\tau\rho}}{m!}$$

where $\rho = \|A\|$.

Proof. (Saad, 1992, pp. 11–12) We define $f(x)$ to be any function on the matrices A and H_m, then p_{m-1} is any polynomial of degree $\leq m-1$ that approximates f. We call r_m the remainder of $e^x - p_{m-1}(x)$, so that we can define

$$f(\tau A)b - \beta V_m f(\tau H_m)e_1 = \beta(r_m(\tau A)b_1 - V_m r_m(\tau H_m)e_1)$$

with $b_1 = b/\|b\|$.
Using triangle inequality we get

$$\|f(\tau A)b - \beta V_m f(\tau H_m)e_1\| \leq \beta(\|r_m(\tau A)b_1\| + \|V_m r_m(\tau H_m)e_1\|)$$

which can be simplified because $\|V_m\|$ is 1 due to its orthogonality

$$\|f(\tau A)b - \beta V_m f(\tau H_m)e_1\| \leq \beta(\|r_m(\tau A)b_1\| - \|r_m(H_m)e_1\|).$$

To continue, we have to remember that r_m is the remainder of the approximation of e^x meaning that we can describe it by

$$r_m(A) = \sum_{j=m}^{\infty} \frac{A^j}{j!}$$

which lets us bound $\|r_m(\tau A)b_1\|$

$$\|r_m(\tau A)b_1\| = \|\sum_{j=m}^{\infty} \frac{(\tau A)^j b_1}{j!}\| \leq \sum_{j=m}^{\infty} \frac{1}{j!}\|(\tau A)^j b_1\| \leq \sum_{j=m}^{\infty} \frac{1}{j!}\|(\tau A)^j\| \leq \sum_{j=m}^{\infty} \frac{1}{j!}(\tau\rho)^j.$$

The same holds for $\|r_m(\tau H_m)e_1\|$ as $\|e_1\|$ is, like $\|b_1\|$, also 1.

$$\|r_m(\tau H_m)e_1\| \leq \sum_{j=m}^{\infty} \frac{1}{j!}(\tau\hat{\rho})^j, \quad \hat{\rho} = \|H_m\|$$

Saad (1992) showed that $\sum_{j=m}^{\infty} \frac{1}{j!}\rho^j$ is increasing with ρ and that $\hat{\rho} \leq \rho$, which means that

$$\sum_{j=m}^{\infty} \frac{1}{j!}\hat{\rho}^j \leq \sum_{j=m}^{\infty} \frac{1}{j!}\rho^j.$$

By bringing together all inequalities we get

$$\begin{aligned}\|f(\tau A)b - \beta V_m f(\tau H_m)e_1\| &\leq \beta(\|r_m(\tau A)b_1\| + \|r_m(\tau H_m)e_1\|) \\ &\leq 2\beta \sum_{j=m}^{\infty} \frac{(\tau\rho)^j}{j!} \end{aligned} \tag{3.15}$$

When we additionally define t_m as the remainder of e^x and its Taylor Series approximation

$$t_m(x) = e^x - \sum_{j=0}^{m-1} \frac{x^j}{j!}$$

for which Saad (1992)[p. 11] proofed that

$$t_m(x) = \frac{x^m}{m!}(1 + o(1)) \leq \frac{x^m e^x}{m!}$$

we can further specify (3.15) to (3.1.3). □

The approximation for the corrected scheme looks similar

$$\epsilon_C = \|e^{\tau A}b - \beta V_m e^{\tau H_m} e_1 + \beta h_{m,m+1}\tau e_m^T \varphi_1(\tau H_m)e_1 v_{m+1}\| \leq 2\beta \frac{(\tau\rho)^{m+1}e^{\tau\rho}}{(m+1)!} \tag{3.16}$$

and so does its proof so we won't show it again. But one can see that the mth approximation of the corrected scheme, is the $(m+1)$th approximation of the basic scheme, showing that the bound indeed became a bit sharper by the correction.
Before we continue with the error approximation, we will take a look at the spectral norm and the spectral radius of our matrix A.

Theorem 3.1.4. *For a LAF matrix A, its spectral radius and spectral norm are* $\rho(A) = \|A\| = 1$.

Proof. The matrix A is built by $\Omega \otimes \mathbb{I}_J$. Ω is symmetric, making $\rho(\Omega) = \|\Omega\|$, and its doubly stochastic, meaning that all rows and all columns sum up to 1, from which follows that $\rho(\Omega) = 1$. The same is valid for $\mathbb{I}_J$. The set of eigenvalues of an Kronecker product of two square matrices K and L is $\sigma(K \otimes\ L) = \sigma(K) \otimes \sigma(L)$ (Schäcke, 2004, Th. 2.3), meaning that the largest absolute eigenvalue (spectral radius) will be the product of the largest absolute eigenvalues of K and L, meaning that $\rho(\Omega) \cdot \rho(\mathbb{I}_J) = \rho(\Omega \otimes \mathbb{I}_J) = 1$. □

By knowing that, we can simplify our a priori error to

$$\epsilon_B \leq 2\beta \frac{\tau^m e^\tau}{m!} \tag{3.17}$$

$$\epsilon_C \leq 2\beta \frac{\tau^{m+1} e^\tau}{(m+1)!,} \tag{3.18}$$

meaning that it's not dependent of A itself, but only of its scaling factor, respectively the t we want integrate up to.
Gallopoulos and Saad (1992) sharpened the error for some matrices B, by using the logarithmic norm

$$\mu(B) \equiv \lim_{h \to 0+} \frac{\|\mathbb{I} + hB\| - 1}{h} \tag{3.19}$$

instead of the spectral norm. Unfortunately we won't profit from this, because the logarithmic norms becomes equal to the largest eigenvalue of the symmetric part of B, if the the 2-norm is used in (3.19) (Gallopoulos and Saad, 1992, p. 1241). As our matrix A is already symmetric $\mu(A) = \rho(A) = \|A\|$.
As the a priori error is not very helpful for estimating the actual error, we will also define some a posteriori error estimates, whose accuracy we will later demonstrate in the experiments. They are based on the idea of taking Sidjes error approximation (3.12), that we also used for the corrector, and calculate it up to a higher term. For the basic scheme of our system, integrated up to t,

$$x(t) \approx t\beta V_m \varphi_1(tH_m) e_1 \tag{3.20}$$

we will define the approximations

$$error1 = \|\beta h_{m,m+1} t^2 e_m^T \varphi_2(tH_m) e_1 v_{m+1}\| \tag{3.21}$$

$$error2 = \|\beta \sum_{j=2}^{3} h_{m,m+1} t^j e_m^T \varphi_j(tH_m) e_1 A^{j-2} v_{m+1}\| \tag{3.22}$$

$$error3 = \|\beta \sum_{j=2}^{4} h_{m,m+1} t^j e_m^T \varphi_j(tH_m) e_1 A^{j-2} v_{m+1}\|, \tag{3.23}$$

where error1 is the norm of the corrector, we defined earlier. Similarly the approximations for the corrected scheme

$$x(t) \approx t\beta V_m \varphi_1(tH_m) e_1 + \beta h_{m,m+1} t^2 e_m^T \varphi_2(tH_m) e_1 v_{m+1} \tag{3.24}$$

will be

$$error4 = \|\beta h_{m,m+1} t^3 e_m^T \varphi_3(tH_m) e_1 A v_{m+1}\| \tag{3.25}$$

$$error5 = \|\beta \sum_{j=3}^{4} h_{m,m+1} t^j e_m^T \varphi_j(tH_m) e_1 A^{j-2} v_{m+1}\| \tag{3.26}$$

$$error6 = \|\beta \sum_{j=3}^{5} h_{m,m+1} t^j e_m^T \varphi_j(tH_m) e_1 A^{j-2} v_{m+1}\|. \tag{3.27}$$

3.2 Time Complexity

To analyze the computational cost of the methods we will have a detailed look at all operations that have to be executed. When we have those single costs, we can approximate how much time a method will need in total, depending on its parameters. But to get comparable results we first have to set some standards, of what we count as operation and what not. First we will only count mathematical operations, meaning that memory allocations and matrix/vector indexing are not considered. Those mathematical operations are the basic scalar operators $+, -, \cdot, /$. Respectively more complex commands are approximated by the number of basic operations they need. For example, the euclidean norm of a vector of length n needs n multiplications, $n-1$ sums and 1 square root, resulting in $2n$ operations. Additionally, more complex algorithms, like the computation of the matrix exponential are also just approximated by their complexity named in the literature.

3.2.1 Time Complexity of the Explicit Euler Integration

We consider the iteration formula for the *EEM*

$$a_{n+1} = a_n + (Aa_n + b) \cdot h. \tag{3.28}$$

The costs of computing it are dependent on 5 different parameters, that can be divided into two different groups of parameters, which we will call **quantitative** and **qualitative**. Quantitative ones are the step size h and the difference between t_0 and t, δt. Changing them, will only influence the number of iterations we have to do. Qualitative parameters influence the complexity of our calculations, but the number of iterations stay the same. They are I and J, the number of pixels and labels, which will determine the dimensions of our system, and N_A, the number of non-zero entries in A. As A is very sparse it's unnecessary to compute the full matrix-vector multiplication, but rather only

consider all non-zero values. In the future we will approximate the cost of calculating this sparse matrix-vector multiplication by $2N_A$ (Niesen and Wright, 2012, p. 5).
Now that we know all possible parameters, we can start to analyze the total number of operations, which we denoted by the function $T(\delta t, h, I, J, N_A)$. While the number of iterations can be defined quite easily by $\frac{\delta t}{h}$, we have to look closer for the operations per iteration. Here we have $2N_A$ operations for the matrix-vector multiplication, IJ for adding b, IJ for multiplying h and IJ for adding a_n in the end, which results in $3IJ + 2N_A$ operations. So our total number of operations results in

$$T(\delta t, h, I, J, N_A) = \frac{\delta t}{h}(3IJ + 2N_A). \tag{3.29}$$

But if we look closer at how the LAF is constructed we see that actually N_A is dependent of I and J.

Theorem 3.2.1. *The number of non-zero entries N_A in the matrix A is smaller than $9IJ$.*

Proof. We consider our definition of the Matrix A (2.11). While $\mathbb{I}_J$ has J non-zero entries on its diagonal, the matrix Ω has less than $9I$, as it's simply the convolution of all pixels with a 3×3-kernel, meaning that all inner pixels account for 9 non-zero entries and all border pixels for less than 9. That $N_A = N_{\mathbb{I}_J} \cdot N_\Omega$ follows by the basic properties of the Kronecker Product. □

$N_A < 9IJ$ is a relatively sharp upper bound and for large images, we can say that $N_A \approx 9IJ$. Now that we now the number of N_A, we can simplify (3.29) to

$$T_{Euler}(\delta t, h, I, J) \approx \frac{21 \cdot \delta t\, IJ}{h}. \tag{3.30}$$

But because we are usually not interested in the exact number of operations, but rather how the number will scale if we change a parameter, we will define that the Explicit Euler Method lies in

$$\mathcal{O}\left(\frac{\delta t\, IJ}{h}\right) \tag{3.31}$$

meaning that the number of operations is bounded above by $\frac{\delta t\, IJ}{h}$.

3.2.2 Time Complexity of the Krylov Subspace Method

Lastly we will also look at the time complexity of the *KSM*. All of its steps can be summarized by following algorithm, but we won't include step 2 and 4 in our calculation,

Algorithm 3 Krylov Subspace Method

Require: Matrix **A**, Vector **b**, number of dimensions **m**
1: $V_m, H_m \leftarrow$ Orthonormalization with Arnoldi/Lanczos iteration
2: $\hat{H}_m \leftarrow$ Augment Matrix H_m
3: $e^{\hat{H}_m} \leftarrow$ Calculate the Matrix Exponential of $\hat{H}_m$
4: $\varphi_1(H_m)e_1, \varphi_2(H_m)e_1 \leftarrow$ Extract φ-vectors from $e^{\hat{H}_m}$
5: **return** $\|b\|V_m\varphi_1(H_m)e_1 + \|b\|h_{m+1,m}e_m^T\varphi_2(H_m)e_1v_{m+1}$

as they are not mathematical operations. In this method our parameters will be I, J, m, q and s (q and s will be explained later).
So we begin with the orthonormalization of A, where we can either use the Arnoldi (Alg. 1) or the Lanczos iteration (Alg. 2). Even though our Matrix A is always symmetric, which makes the use of the Lanczos iteration possible, we will include both of them. We won't add together every single of their steps, but rather rely on the number of operations given by Niesen and Wright (2012). They define

$$\begin{aligned} T_{Arnoldi}(m, I, J) &= \frac{3}{2}(m^2 - m + 1)IJ + 2mN_A \\ &\approx \frac{3}{2}(m^2 + 11m + 1)IJ \end{aligned} \tag{3.32}$$

and

$$\begin{aligned} T_{Lanczos}(m, I, J) &= 3(2m - 1)IJ + 2mN_A \\ &\approx 3(8m - 1)IJ \end{aligned} \tag{3.33}$$

As already explained in Section 2.3, there are multiple ways to calculate the matrix exponential, but as the Diagonal Padé Approximation with Scaling and Squaring is the most commonly used one, we will use it here. (Moler and Loan, 2003, p. 12) defines its number of operations by

$$T_{Padé}(q, s, m) = (q + s + \frac{1}{3})m^3 \tag{3.34}$$

where q is the degree of the Padé approximation and s the number that we need to bring the spectral radius of A below 0.5 by $A/2^s$, to make it accurate and efficient.
The only thing that's left is to calculate the approximated matrix exponential of A.

That's one norm, one non-sparse matrix-vector ($2mIJ - IJ$), 1 scalar-scalar, 1 vector-vector and 3 scalar-vector multiplications, resulting in $2mn + 2n + 3m$ operations for this step.
When we add everything up we end up with

$$\begin{aligned} T_{KSM\_Arnoldi}(m, I, J, q, s) &\approx \frac{3}{2}(m^2 - 11m + 1)IJ + (q + s + \frac{1}{3})m^3 + 2mIJ + 2IJ + 3m \\ &\approx \frac{3}{2}m^2IJ + \frac{37}{2}mIJ + \frac{7}{2}IJ + 3m + (q + s + \frac{1}{3})m^3 \end{aligned} \tag{3.35}$$

operations for the Arnoldi iteration and

$$\begin{aligned} T_{KSM\_Lanczos}(m, I, J, q, s) &\approx 3(8m - 1)IJ + (q + s + \frac{1}{3})m^3 + 2mIJ + 2IJ + 3m \\ &\approx 26mIJ - IJ + 3m + (q + s + \frac{1}{3})m^3 \end{aligned} \tag{3.36}$$

operations for the Lanczos iteration.
Hence, we can say that the complexity of the Krylov Subspace Method lies in

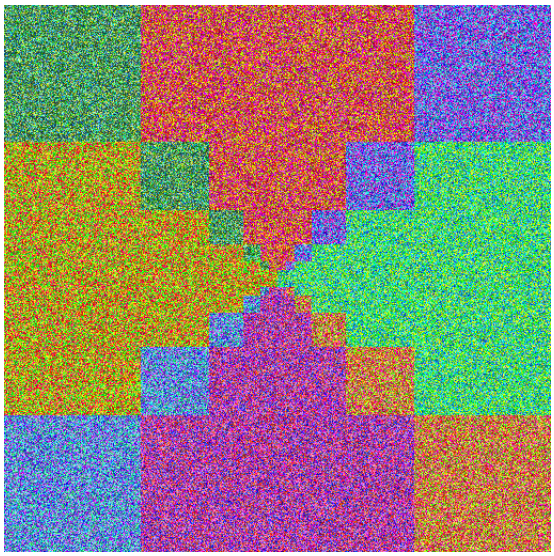
$$\mathcal{O}\left(m^2IJ + (q + s)m^3\right) \quad \text{(Arnoldi)} \tag{3.37}$$

$$\mathcal{O}\left(mIJ + (q + s)m^3\right) \quad \text{(Lanczos)} \tag{3.38}$$

Although the complexity may not look very good on the first sight, due to the m^3 on the right-hand side, we have to remember that we choose $m \ll IJ$, resulting in the left-hand side mIJ / m^2IJ being dominant. So for small m, the computational cost of this method grows linearly proportional to m, I and J. Also a higher degree Padé approximation or a big spectral radius of A won't influence the cost a lot, because q and s are much smaller than IJ.

4 Experiments

In this last chapter of the thesis, we will look at the theorems we did the last chapter and try to validate them experimentally. All experiments were run on an Acer Aspire 5 (Intel i5-8250U, 8GB Memory) in Python 3.8. Our testing *ODE* system will be a Linear Assignment Flow of the following image of size 512×512:



4.1 Error

In the following, we will validate our error approximations of our two methods. As we are not able to compute a correct solution to compare to, we will define an almost correct one by the result of the *KSM* with $m = 150$. If we look at our a priori error estimate, this m should be sufficiently large enough to make to the approximation indistinguishable from the correct solution.

4.1.1 Error experiments of the Explicit Euler Method

For the *EEM*, we approximated that the global error is bounded by

$$\|\epsilon_n\| \leq \frac{h \max \|\ddot{x}\|}{2} \frac{e^{t_n L_f} - 1}{L_f} \tag{4.1}$$

which means, that the it should grow linearly with the step size h, at least for small h, and exponentially with t_n (later we'll call it δt), the time we integrate up to. In Figure

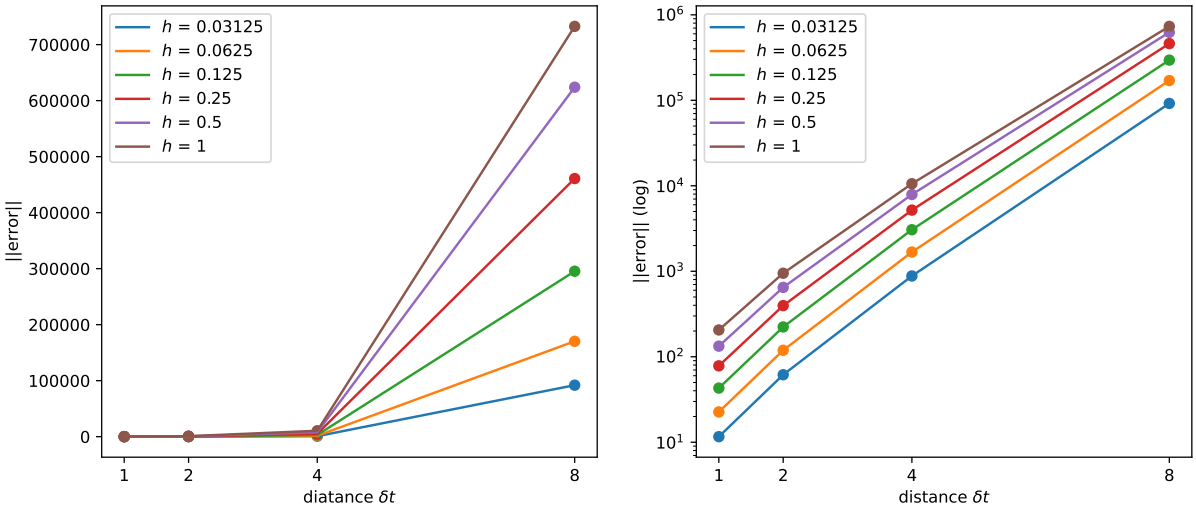


Figure 4.1: (Euler) Error for varying δt. left: linear, right: log

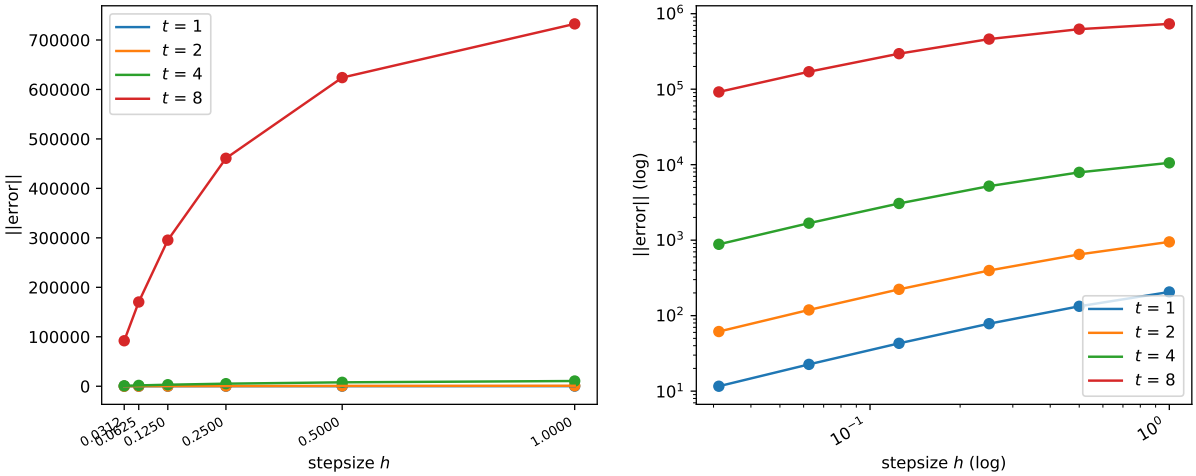


Figure 4.2: (Euler) Error for varying step size h. left: linear, right: log

4.1 we see, that indeed the error grows exponentially, which makes it hard to recognize the values for a smaller dt in the linear plot (left). That those values produce an almost linear graph in the log plot (right), shows that there is an exponential relationship

between dt and the error.

Figure 4.2 on the other hand, shows that for a small h, the error grows linearly with h. Again large value differences make it hard to see all graphs in the purely linear scaled plot, which is why all values are also plotted in log-log.

4.1.2 Error experiments of the Krylov Subspace Method

For the Krylov Subspace Method we defined the a priori error bounds

$$\epsilon_B \leq 2\beta \frac{\tau^m e^\tau}{m!} \tag{4.2}$$

$$\epsilon_C \leq 2\beta \frac{\tau^{m+1} e^\tau}{(m+1)!,} \tag{4.3}$$

for the basic and the corrected scheme. As ϵ shrinks factorially fast, we should observe an error that converges to 0 for a growing m. The larger our integration distance t is, the larger m has to be, to let the error shrink to zero. Additionally ϵ_C should be sharper than ϵ_B. In Figure 4.3 we see that all this expectations satisfied as our approximation is an upper bound to the actual error, although not a sharp one, especially for a large t. Also converges the error to 0 very fast, even for an $m \ll IJ$, with the corrected scheme converging a bit faster than the basic one.

For the a posteriori error, we defined three approximations for the basic (3.21) and

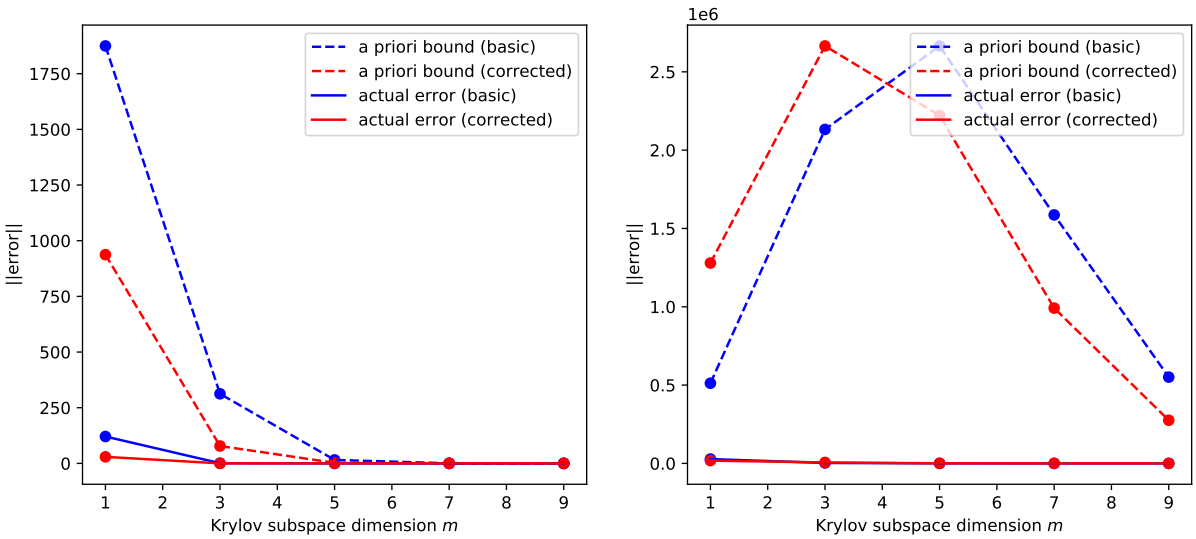


Figure 4.3: (Krylov) A priori bounds and the actual error for $t = 1$ (left) and $t = 5$ (right).

three for the corrected scheme (3.25). In Figure 4.4 we can see their performance for

different t and m. What we can say about them, is that indeed the more terms of Sidjes error approximation (3.12) we calculate, the higher the accuracy. We can also see, that the higher t is, the more error terms we need to get a sufficient approximation, as all our errors perform badly for $t = 16$.

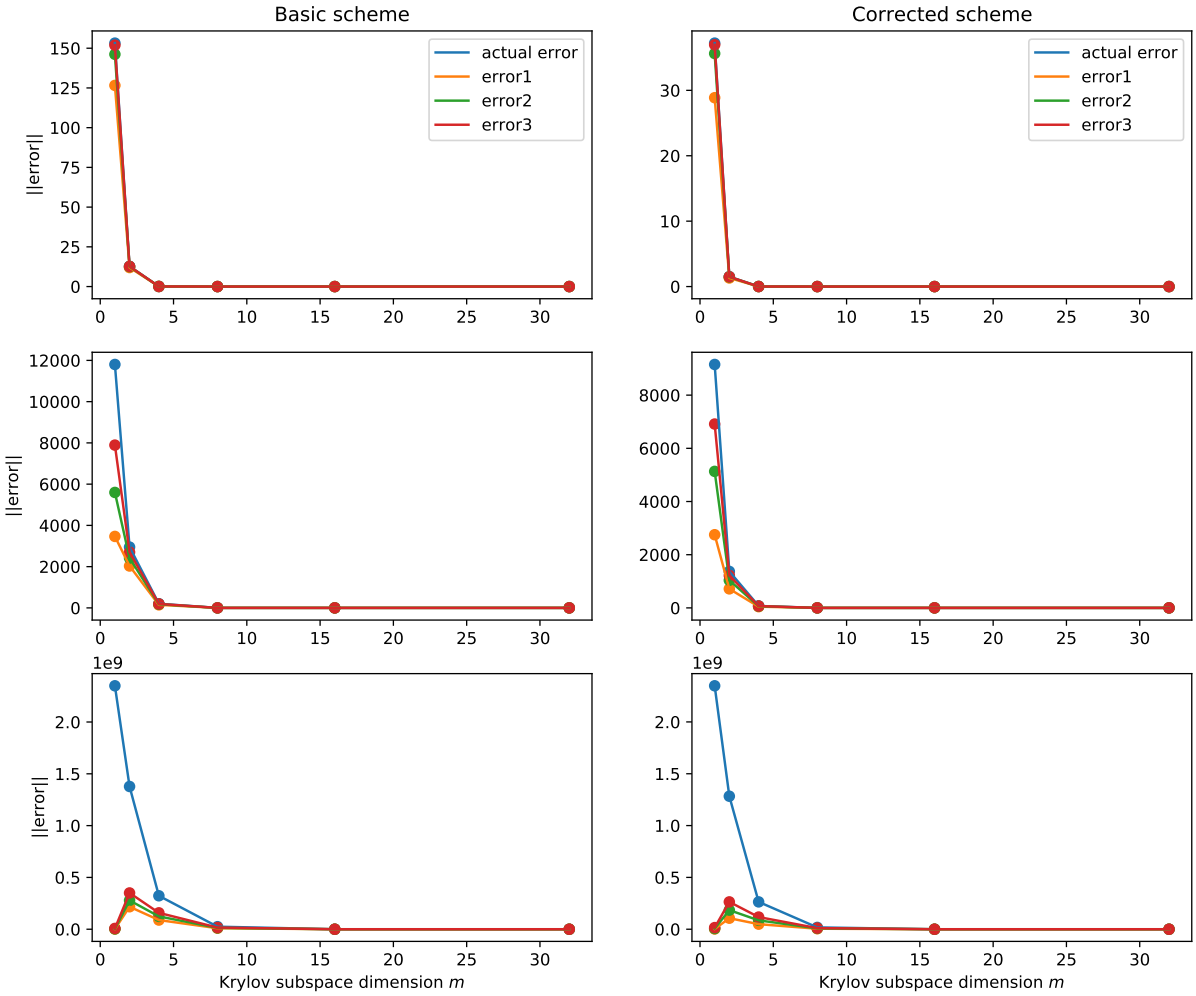


Figure 4.4: ((Krylov) A posteriori error approximations and the actual error in the basic and the corrected scheme for $t = 1$ (top), $t = 4$ (middle) and $t = 16$ (bottom).

4.2 Time

To validate our theorems about the time complexity of the algorithms, we repeat the integration of our example system multiple times, varying only in one parameter, while fixing the others. The emerging time difference should be proportional to our claimed time complexity. As the parameters I and J always show up together, we will treat them as one parameter IJ in the future.

4.2.1 Time experiments of the Explicit Euler Method

We claimed that the *EEM*s time complexity lies in

$$\mathcal{O}\left(\frac{\delta t\, IJ}{h}\right),$$

so while the run time should grow linearly with δt and IJ it should decrease inversely with h.
We tested this with five different inputs for every parameter

- δt: $1, 2, 4, 8, 16$
- IJ: $40\,000, 80\,000, 160\,000, 320\,000, 640\,000$
- h: $0.0625, 0.125, 0.25, 0.5, 1$

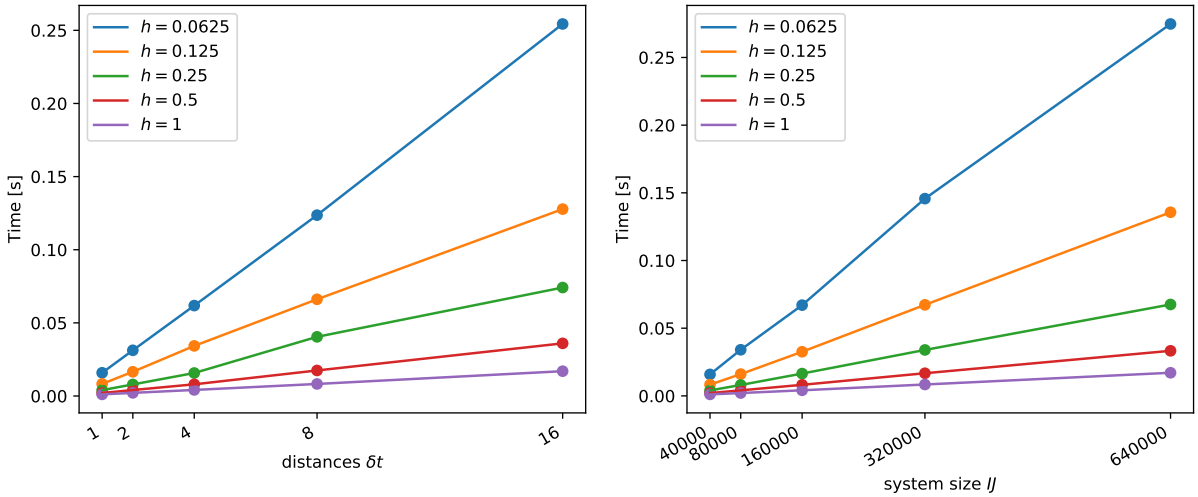


Figure 4.5: (Euler) Run times for varying δt (left) and IJ (right).

where every input is always the double of the last one. The results look as follows: As one can see in Figure 4.5, as predicted the run time grows linearly with δt and IJ, with every doubling of the parameter, the time also roughly doubles. But with every doubling of the step size, the time halves, so it decreases inversely (see Figure 4.6).

4.2.2 Time experiments of the Krylov Subspace Method

We estimated the complexity of the *KSM* with Lanczos iteration to be

$$\mathcal{O}\left(mIJ + (q+s)m^3\right), \tag{4.4}$$

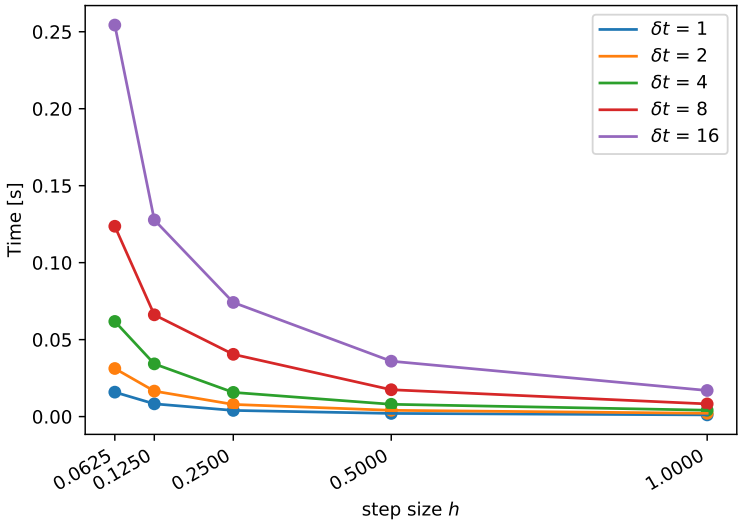


Figure 4.6: (Euler) Run times for varying h and a fixed IJ of 40 000.

thus we should be able to observe the following run time behavior for the case $m \ll IJ$:

- grows linear with IJ
- stays constant with t, respectively $\|A\|$
- grows linear with m

In Figure 4.7 we see, that the first two points are true, the run time grows linear with the system size IJ and stays mostly constant for all different spectral norms of A. In Figure 4.8 (left), we see that also the third assumption is correct and the time grows linear with m. For the case of $m \approx IJ$ however we should be able to observe that the run time

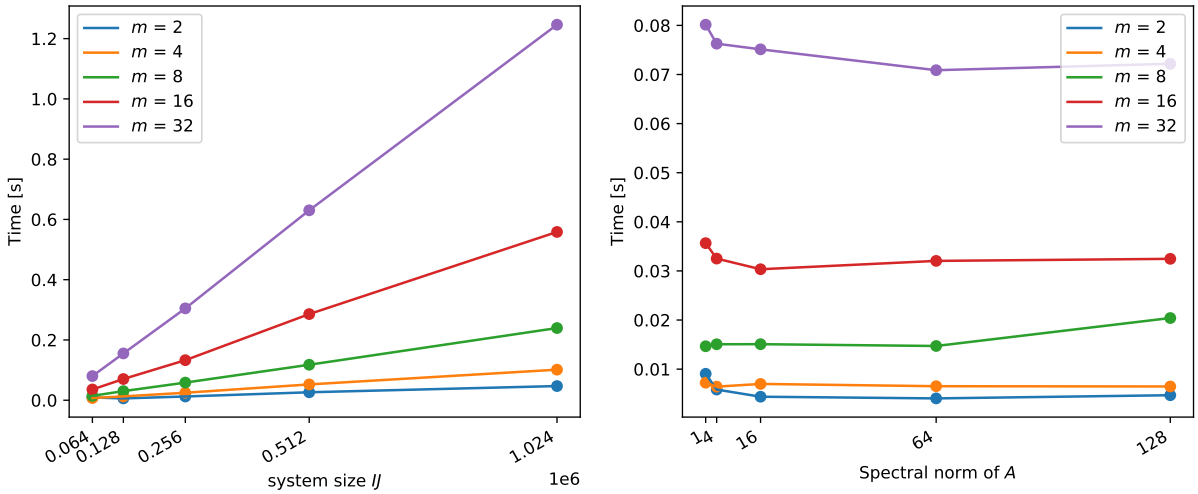


Figure 4.7: (Krylov) Run times for varying IJ (left) and varying $\|A\|$ (right).

grows exponentially fast with m and IJ, while it also now grows with t, respectively $\|A\|$. How fast the run time will grow with t is very dependent of the implementation of the matrix exponential, but in Figure 4.9 we can see, that it grows logarithmic for the implementation "scipy.linalg.expm". Also the theory, that the run time now grows exponentially fast with m and IJ, is true, as can be seen in Figure 4.8 (right).

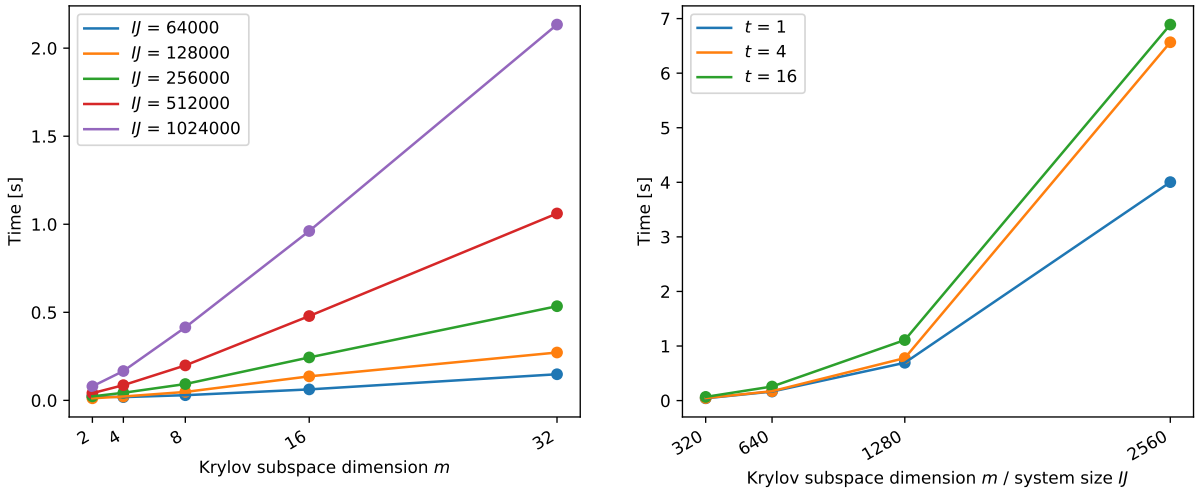


Figure 4.8: (Krylov) Run times for varying m, if $m \ll IJ$ (left) and $m = IJ$ (right).

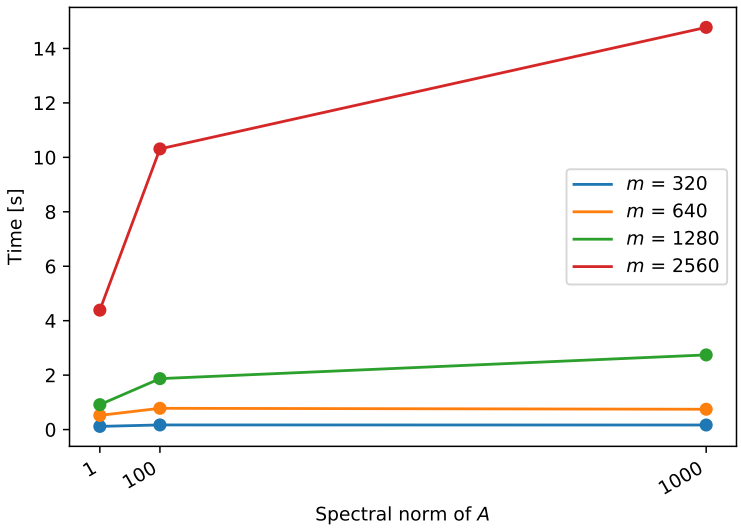


Figure 4.9: (Krylov) Run times for varying $\|A\|$, respectively t.

4.3 Further experiments

Sparse matrix-vector multiplication

A normal matrix-vector multiplication of a matrix $A \in \mathbb{R}^{n \times n}$ and a vector $b \in \mathbb{R}^n$ needs n^2 multiplications and $n(n-1)$ summations, resulting in $2n^2 - n$ total operations. On the other hand we approximated that the sparse matrix-vector multiplication only takes $2N_A$ operations, where N_A is the number of non-zero entries in A. So while the time needed to compute the dense multiplication grows exponentially fast with the matrix size, the sparse multiplication will grow linearly with the number of nonzero entries. And because we showed in Theorem 3.2.1, that the number of nonzero entries of the LAF matrix A grows linearly with IJ, the time needed to compute A should grow linearly too.

To test our approximations, we create sparse LAF matrices of the different sizes and their dense counterparts. After that, we measure the time needed to multiply each one of them with a vector b. In Figure 4.10 we see the run times for different matrix sizes n compared with our approximation $2n^2 - n$ from before. One can see that they progress almost equally, so our approximation should be correct.

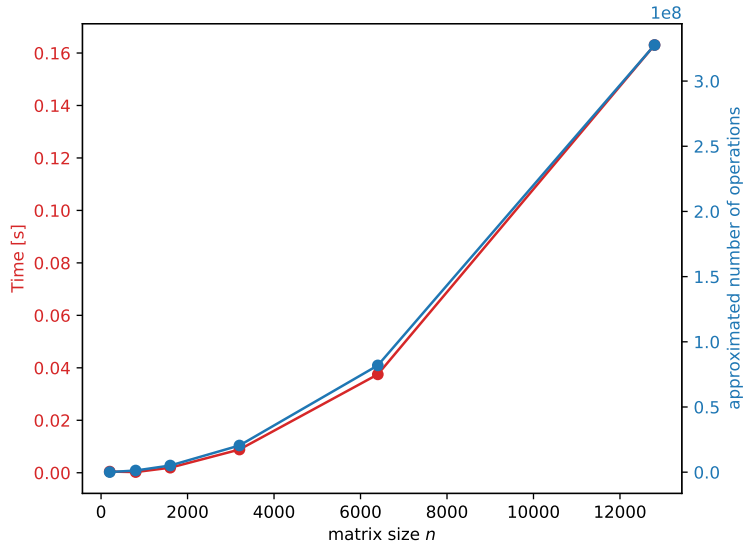


Figure 4.10: Run times of the dense matrix-vector multiplication (red), approximated number of operations (blue)

In Figure 4.11 we see the run time for sparse multiplications together with the number of non-zero entries in A. Like before both of them progress almost equally, which should show that our assumptions about the dense and sparse matrix-vector multiplication were correct.

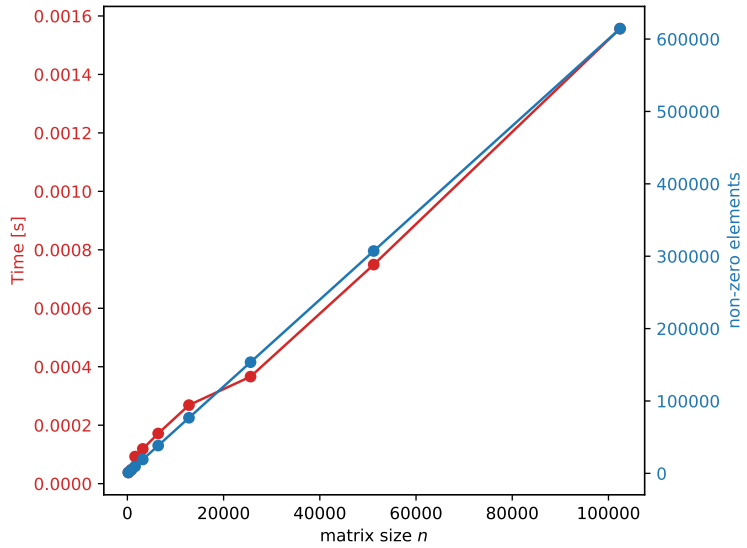


Figure 4.11: Run times of the sparse matrix-vector multiplication (red), number of non-zero entries in A (blue)

Error vs computation time

We now know the time and error complexity of the *EEM* as well as of the *KSM*, but we don't know yet in which situation which of them performs best. In Figure 4.12, we plotted the error (x-axis) and the computation time (y-axis) for a certain m in the *KSM* or h in the *EEM*, so that an optimal algorithm would sit in the bottom left, with no error and a fast computation, and a not optimal one in the top right corner, with a high error and long computation. We can see, that even though their errors are similar

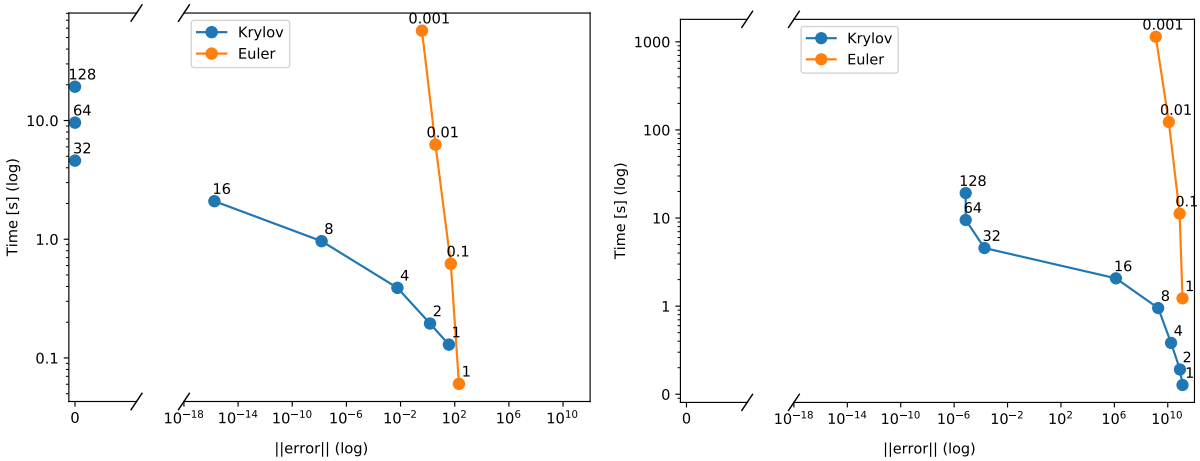


Figure 4.12: Error and computation time of *EEM* and *KSM* for $t = 1$ (left) and $t = 20$ (right).

for their fastest parameters ($m = h = 1$) and the needed time grows linearly for both

methods, the factorially fast shrinking error of the *KSM* makes it perform much better in basically every situation. For a small t like 1, it does not make a huge difference which method is used, as even the highest errors here are pretty low for a huge vector of roughly size 2 000 000 in this experiment. For a large t however, only the *KSM* with a reasonably large m is a feasible option.

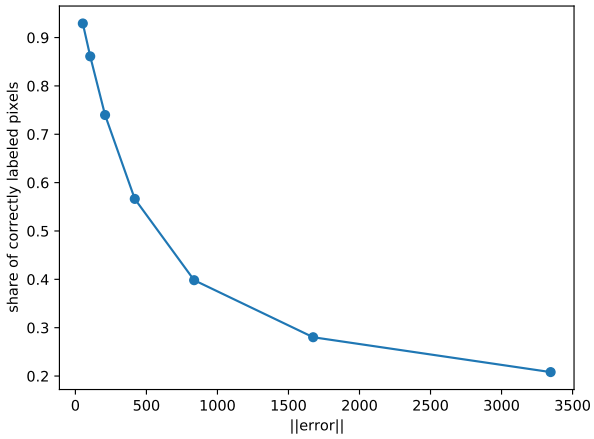


Figure 4.13: Share of correctly labeled pixels, depending on an error produced by random noise.

Mislabeled pixels

Even though we classified errors as large or small before, we don't know yet how large an error actually needs to be, to affect our image labeling and how much computation time we have to invest, to get a correctly labeled image.

We can see, that for a purely random produced error, the number of correctly labeled

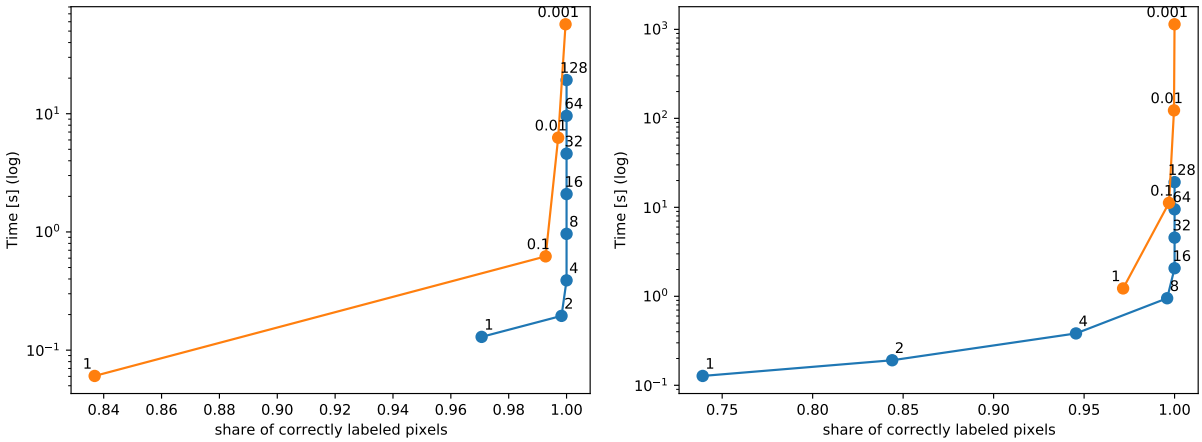


Figure 4.14: Share of correctly labeled pixels and computation time of *EEM* and *KSM* for $t = 1$ (left) and $t = 20$ (right).

pixels decays exponentially (Figure 4.13). Figure 4.14 however shows, that even for a large h or a small m, the actual labeling can be pretty good, even though a higher m and lower h should be used, if one wants certainty that the image is definitely labeled correct.

5 Summary and Future Work

The objective of this thesis was, to compare the time and error complexity of the Explicit Euler Method and the Krylov Subspace Method, depending on their main parameters. We have seen, that while both methods grow linear in time (Krylov: $m \ll IJ$), their error complexity is quite different. While the *EEM*, as order 1 method, increases its error linearly with the step size, the upper error bound of the *KSM* shrinks factorially fast with the subspace dimension. Our experiments have shown, that the *KSM* performs better in almost every way, except maybe a special situation, where we have a small t to integrate up to and we don't care about the error as long it's reasonably low, for example seen seen in Figure 4.12 (left).
Despite all its advantages, the *KSM* can also be inaccurate, if t, respectively the spectral norm of A, is large and m is chosen too small. It would be interesting to compare the results of this thesis to a third method, where the *EEM* and *KSM* are combined, meaning that we would use the Krylov Subspace to approximate the φ-Functions in the Explicit Exponential Euler Method

$$a_{n+1} = \varphi_0(tA)a_n + t\varphi_1(tA)b. \tag{5.1}$$

This method has a big potential, as it combines the advantages of our methods: high accuracy, even for a small subspace dimension m, and small step sizes, that result in a small spectral radius. Additionally it could be provided with adaptive time stepping, where we use our a posteriori approximation to find the local error and then adaptively scale m or t, to be as accurate as needed.
In our motivation we mentioned, that a modern image labeling algorithm needs to be fast, preferably real time computable, and reliable. Under the assumption, that the Linear Assignment Flow itself works reliable, we can say that using the *KSM* as integration method definitely fulfills these requirements. The *EEM* on the other hand would need to much time to work reliable.

Bibliography

Führer, C. and Schroll, A. (2001). Numerical analysis—an introduction. *Lecture Notes*, pages 107–110.

Gallopoulos, E. and Saad, Y. (1992). Efficient solution of parabolic equations by krylov approximation methods. *SIAM J. Sci. Stat. Comput.*, 13(5):1236–1264.

Haubold, H. J., Mathai, A. M., and Saxena, R. K. (2011). Mittag-Leffler Functions and Their Applications. *Journal of Applied Mathematics*, 2011:1–51.

Hochbruck, M. and Lubich, C. (1997). On Krylov Subspace Approximations to the Matrix Exponential Operator. *SIAM Journal on Numerical Analysis*, 34(5):1911–1925.

Moler, C. and Loan, C. V. (2003). Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49.

Niesen, J. and Wright, W. M. (2012). Algorithm 919. *ACM Transactions on Mathematical Software*, 38(3):1–19.

Saad, Y. (1992). Analysis of some krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 29(1):209–228.

Schäcke, K. (2004). On the kronecker product. *Preprint*.

Sidje, R. (1998). Expokit: A software package for computing matrix exponentials. *ACM Trans. Math. Softw.*, 24:130–156.

Teschl, G. (2004). *Ordinary Differential Equations and Dynamical Systems*. American Mathematical Soc.

Zeilmann, A., Savarino, F., Petra, S., and Schnörr, C. (2020). Geometric numerical integration of the assignment flow. *Inverse Problems*, 36(3):034003.