NTNU
Department of Mathematical Sciences

Krylov methods for linear Hamiltonian ODEs

Sindre Karlsen Eskeland

March 4, 2016

MASTER thesis

Department of Mathematical Sciences

Norwegian University of Science and Technology

Supervisor: Professor Elena Celledoni

Preface

This is a master thesis as a part of the study program industrial mathematics at NTNU. It was written during the winter 2015-2016. It is assumed that the reader is familiar with numerical difference methods, numerical linear algebra and MATLAB.

Acknowledgment

Thanks to Elena Celledoni for guiding me; to Lu Li for helping me debugging my code; to my family for motivating me; to Trygve for helping me with proofs; to Geir and Morten for allowing me to sleep in their apartment; to Ane for not crashing my computer; and to Harald for distracting me with silly computer games.

Summary

Krylov methods are projection methods that can transform big linear ODEs to smaller linear ODEs with similar properties. Two such methods (the symplectic Lanczos method (SLM) and the Krylov projection method (KPM)), together with a more classical method, are compared with each other on linear Hamiltonian differential equations, restarts are used to improve the solutions via iterative refinement. The behavior of global error and energy as a function of time is examined. Computation time for the different methods are also shown. The methods are also compared to each other on non autonomous linear Hamiltonian differential equation. Energy preservation for the symplectic Lanczos method is proved and shown when restart is not used, along with convergence for both Krylov methods.

Sammendrag

Krylovmetoder er projeksjonsmetoder som kan transformere store lineære differensialligninger til mindre lineære differesialligninger med lignende egenskaper. To slike metoder (symplectic Lanczos method og Krylov projection method), sammen med en mer vanlig metode, er sammenlignet med hverandre på lineære Hamiltonske differensiallikninger, omstarter er brukt for å forbedre løsningen via iterativ forfining. Oppførsel av global feil og energi som en funksjon av tid er er utforsket. Metodene er også sammenlignet med hverandre på ikke autonome lineære Hamiltonian differensiallikningen. Energibevaring for symplectic Lanczos method er bevist og vist om omstart ikke er benyttet, sammen med konvergens for begge Krylov metodene.

Contents

Chapter 1

Introduction

Hamiltonian differential equations are found in several branches of physics and mathematics, eg. in planetary motion, in particle physics and in control theory. This text will be examining how Krylov methods can be utilized on linear Hamiltonian differential equations on the form

$$\begin{aligned} \dot{u}(t) &= Au(t) \\ u(0) &= u_0. \end{aligned} \tag{1.1}$$

For the system to be considered Hamiltonian, the matrix $A \in \mathbb{R}^{2j \times 2j}$ needs to have the following property [6]

$$(J_j A)^\top = J_j A,$$

where

$$J_j = \begin{bmatrix} 0 & I_j \\ -I_j & 0 \end{bmatrix},$$

and I_j is the $j \times j$ identity matrix.

The matrix A is often large and sparse. Computing good approximations of the linear system can be computationally costly. Krylov methods that exploits these properties can therefore be a valuable tool. These methods can transform a big linear system to a small linear system, leading to less computationally demanding calculations. This text will consider two such methods, the symplectic Lanczos method (`SLM`), and the Krylov projection method (`KPM`).

SLM only works with Hamiltonian matrices, it has the property that the projected system also is Hamiltonian. This property has made SLM a popular choice when finding eigenvalues of large Hamiltonian matrices, see eg. [4] and [27]. Since SLM preserves the Hamiltonian structure of the matrix, and the projected problem has the same structure as equation (1.1), the produced approximation is energy preserving. KPM is not designed to have any structure preserving property, but require less computations per step. It also works on any ODE on the form of equation (1.1), not just in the cases where A is Hamiltonian. The Krylov methods can utilize restarts to improve the solutions via iterative refinement.

The aim with this thesis is to study these two Krylov methods and the restart, and compare them to other popular solution methods in: global error, energy preservation, and computation time. The energy preservation for SLM will also be examined, together with derivation of the methods and proof of convergence.

Equation (1.1) has the well known analytical solution

$$u(t) = \exp(At)u_0.$$

Computing the matrix exponential is a very costly operation. But the Krylov subspace methods presented leads to problems with a size suitable for such computations. How energy, error and computation time behaves in this case will be examined.

In addition to the case with Hamiltonian linear ODEs with constant energy, as in equation (1.1), the Krylov methods will be tested on problems with a source term:

$$\begin{aligned} \dot{u}(t) &= Au(t) + bf(t) \\ u(0) &= u_0, \end{aligned} \tag{1.2}$$

where A is an Hamiltonian matrix, p is a vector, and $f(t)$ is a scalar time dependent function. Behavior of the error and energy as a function of time, together with computation time will be explored. MATLAB notation will be used where applicable.

Chapter 2

Background theory

This chapter will contain the theoretical aspects of the Krylov methods, such as derivation, proof of convergence and necessary assumptions.

2.1 Zero initial condition

If `KPM` and `SLM` are to be used with the restarts, it is important that the initial conditions are zero. The reason for this is explained in Section 2.5.4. Equation (1.1) can be transformed so that it has zero initial conditions in the following way [10]:

Start by considering

$$\hat{u}(t) = u(t) - u_0,$$

then rewrite the equation with the new variable:

$$\begin{aligned} \dot{\hat{u}}(t) &= A\hat{u}(t) + Au_0 \\ \hat{u}(0) &= 0. \end{aligned} \tag{2.1}$$

The solution of the original problem can be obtained by

$$u(t) = \hat{u} + u_0.$$

All test problems with a non-zero initial condition will be transformed in this way without using the hat notation. The letter b will be used to describe the product Au_0.

2.2 Energy

It is well known that the energy of a system on the form of equation (1.1) can be expressed as [11]

$$\mathscr{H}_1(u) = \frac{1}{2}u^\top JAu.$$

If the transformation in Section 2.1 is used, the energy is

$$\mathscr{H}_2(\hat{u}) = \frac{1}{2}\hat{u}^\top JA\hat{u} + \hat{u}^\top Jb. \tag{2.2}$$

2.3 Integration methods

The time domain $[0, T_s]$ will be divided in k pieces, so that the step size is $h = T_s/k$. The time discretized solution of $u(t_j)$ is called U_j, with $t_j = jh$ where $j = 1, 2, \ldots, k$. Since the initial value is known to be zero, $j = 0$ is disregarded. The integration methods considered in this text are the trapezoidal rule, forward Euler, and the midpoint rule. The definitions of the different methods are given in Table 2.1.

The trapezoidal rule and the midpoint rule are the same method if the problem is linear with constant coefficients. The midpoint rule is a symplectic Runge-Kutta method, this means that when applied to an autonomous Hamiltonian system it produces a numeric solution which is a symplectic map [19]. In other words, for symplectic methods

$$\left(\frac{\partial u_n}{\partial u_0}\right)^\top J\frac{\partial u_n}{\partial u_0} = J. \tag{2.3}$$

Here $\frac{\partial u_n}{\partial u_0}$ is the Jacobian matrix obtained by differentiating the components of the numerical solution u_n, with respect to all components of the initial condition. The midpoint rule can be divided into two steps: forward Euler, and backwards Euler. By first performing a backward Eu-

Table 2.1: Methods for integrating in time. Note that since the midpoint rule uses the midpoint $F_{i+\frac{1}{2}}$, twice as many points need to be saved for the midpoint rule as for the other methods. The trapezoidal and midpoint rule have quadratic convergence rates, while forward Euler has linear convergence. To compare the methods we use the squared number of points for forward Euler compared to the other methods. $g(t,u)$ is the right hand side of equation (1.1) or (1.2).

Trapezoidal rule (`trap`) [24]	$\frac{U_{i+1}-U_i}{2h} = g(t_i,U_i)+g(t_{i+1},U_{i+1})$
Forward Euler (`Euler`) [25]	$\frac{U_{i+1}-U_i}{h} = g(t_i,U_i)$
Midpoint rule (`mid`) [26]	$\frac{U_{i+1}-U_i}{h} = g\left(t_i+\frac{h}{2},\frac{1}{2}(U_i+U_{i+1})\right)$

Table 2.2: The method for exact integration in time. Since the method is very computationally demanding it will only be used on small projected matrices. It also need the test problem to have constant energy. The expected convergence will be depending on the approximation of A, since this method is only exact in time. The method is explained in MATLAB's docmentation: [1].

Eigenvalue and diagonalization (`diag`)	$[V,D] = \texttt{eig}(A)$ $U_i = V\cdot\texttt{diag}\Big(\texttt{exp}\big(\texttt{diag}(D\cdot t_i)\big)\Big)/V\cdot b - b$

ler step and then a forward Euler step, we are effectively performing a step of the midpoint rule. The trapezoidal rule is not symplectic, but it is conjugate to the midpoint rule. This means that if you first apply forward Euler, and then backwards Euler, you have done one step of the trapezoidal rule. The trapezoidal rule therefore behaves very similar to the midpoint rule.

Forward Euler has no energy preserving properties, and is used to show the difference between a classical integration method, and an energy preserving integration method.

In addition to the iteration schemes in Table 2.1, an exact solver is used for comparison. It is presented in Table 2.2.

2.3.1 Energy conservation for the trapezoidal rule

This section will show the energy preserving properties of the trapezoidal rule on the initial value problem

$$\begin{aligned} \dot{u}(t) &= Au + b \\ u(0) &= 0. \end{aligned} \tag{2.4}$$

The proof is based on [23]. The energy of this function has already been presented in equation (2.2). The main ingredients in this proof are the gradient of $\mathscr{H}_1$ and the deffinition of the trapezoidal rule. Assume that A is a Hamiltonian matrix, so that JA i symmetric.

The gradient of $\mathscr{H}_1(u)$ is

$$\nabla \mathscr{H}_1(u) = J(Au + b).$$

The trapezoidal rule found in Table 2.1, used on equation (2.4) gives

$$\frac{U_{j+1} - U_j}{h} = A\frac{U_{j+1} + U_j}{2} + b.$$

We observe that

$$\nabla \mathscr{H}_1\left(\frac{U_{j+1} + U_j}{2}\right) = JA\frac{U_{j+1} + U_j}{2} + Jb.$$

Since $\frac{U_{j+1}-U_j}{h} = J^{-1}\nabla \mathscr{H}_1\left(\frac{U_{j+1}+U_j}{2}\right)$ and $\nabla \mathscr{H}_1\left(\frac{U_{j+1}+U_j}{2}\right)^\top J^{-1}\nabla \mathscr{H}_1\left(\frac{U_{j+1}+U_j}{2}\right) = 0$, we have

$$\nabla \mathscr{H}_1\left(\frac{U_{j+1} + U_j}{2}\right)^\top \frac{U_{j+1} - U_j}{h} = 0.$$

Substituting $J\left(A\frac{U_{j+1}+U_j}{2} + b\right)$ for $\nabla \mathscr{H}_1\left(\frac{U_{j+1}+U_j}{2}\right)$ gives

$$\frac{1}{2h}U_{j+1}^\top JAU_{j+1} - \frac{1}{2h}U_j^\top JAU_j + \frac{1}{h}Jb^\top U_{j+1} - \frac{1}{h}Jb^\top U_j = 0.$$

This can be rewritten as

$$\mathscr{H}_2(U_{j+1}) - \mathscr{H}_2(U_j) = 0.$$

Hence the trapezoidal rule conserves the energy for functions with constant energy.

2.4 Windowing

The Krylov methods have a tendency to loose its performance when the time domain is large. A solution to this problem is to divide the time domain in smaller sub intervals, and solve each sub interval individually. How this is done is described in Algorithm 1.

Algorithm 1 Windowing

Start with an initial value $U_0 \in \mathbb{R}^{\hat{m}}$, $K \in \mathbb{N}$ and $k \in \mathbb{N}$.
Make an empty vector U.
for $j = 1, 2, \ldots, K$ **do**
  Solve differential equation with $k+1$ points in time and initial value U_0.
  Place the new points at the end of U.
  Update U_0 to be the last value of U.
  Delete the last point of U.
end for
Return U.

2.5 Solution methods

Two Krylov methods is considered in this text. Their names are symplectic Lanczos method (`SLM`), and Arnoldi's algorithm (`KPM`, as it is implemented as the Krylov projection method). In this section we describe the derivation of these methods and give a proof of the energy preservation for `SLM`.

Krylov methods are techniques to produce approximate solutions by projecting the original problem to obtain a smaller dimensional problem. The main feature of these methods is that a smaller linear system can be used to obtain numerical solutions, and this makes the computations much less demanding. However, finding these projected systems can be time consuming. The approximated solution can be improved by a restart approach. This is done using the Krylov method again, to solve a similar equation for the error. The error is simply the difference between the projected solution, and exact solution. This can be done repeatedly until the desired accuracy is obtained.

Assume that the equations are on the form

$$\begin{aligned} \dot{u}(t) &= Au(t) + b \\ u(0) &= 0 \end{aligned} \tag{2.5}$$

when these methods are used. When using `SLM`, it is important that A is a Hamiltonian matrix.

These methods will be compared to equation (1.1) solved with the trapezoidal rule, or the midpoint rule if equation (1.2) is solved. This method will be called direct method (`DM`).

2.5.1 Arnoldi's Algorithm and the Krylov projection method

This section is based on the derivation of the method done in [13] and [15].

The Krylov subspace is the space $W_n(A,b) = \{b, Ab, \cdots, A^{n-1}b\} = \{v_1, v_2, \cdots, v_n\}$, where $n \leq \hat{m}$. The vectors v_i together with $h_{i,j} = v_i^\top A v_j$ are found by Arnoldi's algorithm, shown in Algorithm 2. Let V_n be the $\hat{m} \times n$ matrix consisting of column vectors $[v_1, v_2, \cdots, v_n]$ and H_n be the $n \times n$ upper Hessenberg matrix with elements $(h_{i,j})_{i,j=1,\cdots,n}$. Then the following holds [28]:

$$\begin{aligned} AV_n &= V_n H_n + h_{n+1,n} v_{n+1} e_n^\top \\ V_n^\top A V_n &= H_n \\ V_n^\top V_n &= I_n. \end{aligned} \tag{2.6}$$

Here, e_n is the n-th canonical vector in $\mathbb{R}^n$, where n is the number of iterations performed with Arnoldi's algorithm.

Construct the approximation $u_n(t) = V_n z_n(t)$, by requiring that

$$\begin{aligned} \dot{z}_n(t) &= H_n z_n(t) + \|b\|_2 e_1 \\ z_n(0) &= 0. \end{aligned} \tag{2.7}$$

Note that from (2.6), $H_n = V_n^\top A V_n$, and $\|b\|_2 e_1 = V_n^\top b$. By substituting $u_n(t) = V_n z_n(t)$ in equa-

Algorithm 2 Arnoldi's algorithm[29]

Start with $A \in \mathbb{R}^{\hat{m}\times\hat{m}}$, $b \in \mathbb{R}^{\hat{m}}$, $n \in \mathbb{N}$ and a tolerance $\iota \in \mathbb{R}$.
$v_1 = b/\|b\|_2$
for $j = 1,2,\ldots,n$ **do**
  Compute $h_{i,j} = v_i^\top A v_j, v_i$ for $i = 1,2,\ldots,j$
  Compute $w_j = Av_j - \Sigma_{i=1}^{j} h_{i,j} v_i$
  $h_{j+1,j} = \|w_j\|_2$
  if $h_{j+1,j} < \iota$ **then**
    STOP
  end if
  $v_{j+1} = w_j / h_{j+1,j}$
end for
Return H_n, V_n, v_{n+1}, $h_{n+1,n}$.

tion (2.5) we obtain the residual

$$r_n(t) = b - \dot{u}_n(t) + Au_n(t).$$

This can be rewritten by means of $z_n(t)$ to get

$$r_n(t) = b - V_n\dot{z}_n(t) + AV_n z_n(t).$$

Using (2.6), we obtain

$$r_n(t) = h_{n+1,n} e_n^\top z_n(t) v_{n+1}. \tag{2.8}$$

Since $h_{n+1,n}$ is zero for some $n \leq \hat{m}$, (since $V_{\hat{m}} h_{\hat{m}+1,\hat{m}} v_{\hat{m}+1} = 0$ by construction [30]), the procedure will converge towards the correct solution $u(t)$.

Larger n gives a better approximation of the solution, but also higher computational complexity. The size of $h_{n+1,n}$ can be used to decide how large n should be, by requiring $h_{n+1,n}$ to be smaller than some tolerance. If a fixed n is preferred, restarting can be used to obtain the desired approximation. A restart is based on solving the equation for the error $\epsilon_n(t) = u(t) - u_n(t)$ to improve the current approximation. This equation is obtained by subtracting (2.5) from (2.7) to get

$$\dot{\epsilon}_n(t) = A\epsilon_n(t) - r_n(t). \tag{2.9}$$

Since (2.9) has a format similar to (2.5), we can apply n iterations of KPM to approximated $\epsilon_n(t)$ numerically, and use the obtained approximation to improve the numerical solution of the original problem, ie. $u(t) \approx u_n(t) + \tilde{\epsilon}_n(t)$, with $\tilde{\epsilon}_n(t) \approx \epsilon_n(t)$. The procedure can be repeated, and the equation for the error after $i-1$ restarts is

$$\epsilon_n^{(i)}(t) = A\epsilon_n^{(i)}(t) - r_n^{(i)}(t),$$

with

$$r_n^{(i)}(t) = h_{n+1,n}^{(i-1)} v_{n+1}^{(i-1)} e_n^\top \epsilon_n^{(i-1)}(t).$$

Equation (2.9) can be projected writing $\epsilon_n^{(i)}(t) \approx V_n \delta_n^{(i)}(t)$, and using (2.6), to obtain the following system of ODEs:

$$\dot{\delta}_n^{(i)}(t) = H_n^{(i)} \delta_n^{(i)}(t) + e_1 h_{n+1,n}^{(i-1)} e_n^\top \delta_n^{(i-1)}(t), \quad i \geq 1. \tag{2.10}$$

The numerical solution after i restarts is found by $u_n^{(i)}(t) = \sum_{j=0}^{i} V_n^{(j)} \delta_n^{(j)}(t)$, where $\delta_n^{(0)}(t) = z_n(t)$ and found by equation (2.7).

Repeatedly solving equation (2.10) can increase the accuracy of the approximated solution within a desired accuracy provided, the restarted method converges. See [14] for a proof that under appropriate assumptions on the matrix A, the restarted KPM method converges even for $n = 1$. It is no longer possible to use $h_{n+1,n}$ as a measure for the error when the restart is used, since Arnoldi's algorithm has no way to measure how much this iteration improved the solution compared to previous iterations.

2.5.2 Symplectic Lanczos method

SLM requires the matrix A in equation (2.5) to be Hamiltonian. The method is very similar to KPM, with the main difference beeing that orthonormality in Arnoldi's algorithm is replaced by symplecticity in SLM. This makes the derivations of the method quite similar.

Let $S_n = [v_1, v_2, \ldots, v_{\frac{n}{2}}, w_1, w_2, \ldots, w_{\frac{n}{2}}] \in \mathbb{R}^{\hat{m} \times n}$, $H_n \in \mathbb{R}^{n \times n}$, $v_{n+1} \in \mathbb{R}^{\hat{m}}$ and $\xi_{n+1} \in \mathbb{R}$ be generated

from Algorithm 3, with the restart variable n, the matrix $A \in \mathbb{R}^{\hat{m}\times\hat{m}}$ and the vector $b \in \mathbb{R}^{\hat{m}}$ as arguments. The following properties then hold:

$$\begin{aligned} AS_n &= S_n H_n + \zeta_{n+1} v_{n+1} e_{\hat{m}}^\top \\ J_n^{-1} S_n^\top J_{\hat{m}} A S_n &= H_n \\ S_n^\top J_{\hat{m}} S_n &= J_n. \end{aligned} \tag{2.11}$$

Algorithm 3 Symplectic Lanczos method [8], with reortogonalization from [5].

Start with a Hamiltonian matrix $A \in \mathbb{R}^{\hat{m}\times\hat{m}}$, $b \in \mathbb{R}^{\hat{m}}$, $n \in \mathbb{N}$
$\tilde{n} = \frac{n}{2}$
$v_0 = 0 \in \mathbb{R}^{\hat{m}}$
$\zeta_1 = \|b\|_2$
$v_1 = \frac{1}{\zeta_1} b$
for $j = 1, 2, \ldots, \tilde{n}$ **do**
  $v = A v_j$
  $\delta_j = v_j^\top v$
  $\tilde{w} = v - \delta_j v_j$
  $\kappa_j = v_j^\top J_{\tilde{m}} v$
  $w_j = \frac{1}{\kappa_j} \tilde{w}_j$
  $w = A w^j$
  $\tilde{S}_{j-1} = [v_1, v_2, \ldots, v_{j-1}, w_1, w_2, \ldots, w_{j-1}]$
  $w_j = w_j + \tilde{S}_{j-1} J_{j-1} \tilde{S}_{j-1}^\top J_{\tilde{m}} w_j$
  $\beta = -w_j^\top J_{\tilde{m}} w$
  $\tilde{v}_{j+1} = w - \zeta_j v_{j-1} - \beta_j v_j + \delta_j v_j$
  $\zeta_{j+1} = \|\tilde{v}_{j+1}\|_2$
  $v_{j+1} = \frac{1}{\zeta_{j+1}} \tilde{v}_{j+1}$
  $\tilde{S}_j = [v_1, v_2, \ldots, v_j, w_1, w_2, \ldots, w_j]$
  $v_{j+1} = v_{j+1} + \tilde{S}_j J_j \tilde{S}_j^\top J_{\tilde{m}} v_{j+1}$
end for
$S_n = [v_1, v_2, \ldots, v_{\tilde{n}}, w_1, w_2, \ldots, w_{\tilde{n}}]$
$H_n = \begin{bmatrix} \operatorname{diag}\left([\delta_j]_{j=1}^{\tilde{n}}\right) & \operatorname{tridiag}\left([\zeta_j]_{j=2}^{\tilde{n}}, [\beta_j]_{j=1}^{\tilde{n}}, [\zeta_j]_{j=2}^{\tilde{n}}\right) \\ \operatorname{diag}\left([\kappa_j]_{j=1}^{\tilde{n}}\right) & \operatorname{diag}\left([-\delta_j]_{j=1}^{\tilde{n}}\right) \end{bmatrix}$
Return H_n, S_n, v_{n+1}, ζ_{n+1}.

The algorithm only performs $\frac{n}{2}$ iterations, since two vectors are created per iteration: v_j and w_j. Creating two vectors per iterations almost halves the total work load, and ensure the symplec-

ticity of the method.

Also in this case we consider approximations of the type $u_n(t) = S_n z_n(t)$ such that

$$\dot{z}_n(t) = H_n z_n(t) + \|b\|_2 e_1$$
$$z_n(0) = 0.$$

From (2.11), we have that $H_n = J_n^{-1} S_n^\top J_{\hat{m}} A S_n$, and $\|b\|_2 e_1 = S_n^\top b$. By substituting $u_n(t) = S_n z_n(t)$ in equation (2.5), we get the residual

$$r_n(t) = b - \dot{u}_n(t) + A u_n(t)$$

Writing this in terms of $z_n(t)$ gives

$$r_n(t) = b - S_n \dot{z}_n(t) + A S_n z_n(t)$$

Use (2.11) to obtain

$$\dot{z}_n(t) = H_n z_n(t) + \|b\|_2 e_1. \tag{2.12}$$

Equation (2.12) and (2.7) are identical, except for the underlying assumptions about H_n. Since $S_n^\top J_n \zeta_{n+1} v_{n+1} = 0$ for some $n \leq \hat{m}$ [7], the method converges.

A restart can be performed if a small, fixed n is preferred instead of an increasing n. The restart procedure can be derived by looking at the difference $\epsilon_n(t) = u(t) - u_n(t)$:

$$\epsilon_n(t) = A\epsilon_n(t) - r_n(t).$$

This equation is similar to (2.12), therefore we can apply iterations of `SLM` to approximate $\epsilon_n(t)$ numerically. The obtained approximation can be used to improve the numerical solution of the original problem, with $u(t) = u_n(t) + \tilde{\epsilon}_n(t)$, with $\tilde{\epsilon}_n(t) \approx \epsilon_n(t)$. This procedure can be repeated,

and the equation for the error after $i-1$ restarts is

$$\dot{\epsilon}_n^{(i)}(t) = A\epsilon_n^{(i)}(t) + r_n^{(i)}(t),$$

where

$$r_n^{(i)}(t) = \zeta_{n+1}^{(i-1)} v_{n+1}^{(i-1)} e_{\hat{m}}^\top \epsilon_n^{(i-1)}(t).$$

Write $\epsilon_n^{(i)}(t) = S_n \delta_n^{(i)}(t)$, and use equation (2.11) to obtain

$$\dot{\delta}_n^{(i)}(t) = H_n^{(i)} \delta_n^{(i)}(t) + e_1 \zeta_{n+1}^{(i-1)} e_n^\top \delta_n^{(i-1)}(t), \quad i \geq 1. \tag{2.13}$$

The numerical solution after $i-1$ restarts is found by $u_n^{(i)}(t) = \sum_{j=0}^{i} S_n^{(j)} \delta_n^{(j)}(t)$, where $\delta_n^{(0)}(t) = z_n(t)$ and found by equation (2.12).

Proof that `SLM` without restart is energy preserving

This section will show that if equation (2.12) is solved by the trapezoidal rule, the energy of $u_n(t)$ is preserved [12]. It is well known that the Hamiltonian ODE (1.1), solved with an energy preserving method has constant energy. Since H_n is Hamiltonian, the solution of (2.12) will have a constant energy. The remaining problem is to show that the transformation from $z_n(t)$ to $u_n(t)$ is energy preserving.

The energy of equation (2.12) is

$$\mathscr{H}_2(z_n) = \frac{1}{2} z_n^\top J_n H_n z_n + z_n^\top J_n e_1 \|b\|_2$$

While the energy of the original problem is

$$\mathscr{H}_2(u_n) = \frac{1}{2} u_n^\top J A u_n + u_n^\top J b.$$

Perform the substitution $u_n(t) = S_n z_n(t)$ to get

$$\mathscr{H}_2(u_n) = \frac{1}{2} z_n^\top S_n^\top J A S_n z_n + z_n^\top S_n^\top J b.$$

Using that $b = S_n e_1 \|b\|_2$, and simplifying with equation (2.11) gives

$$\mathscr{H}_2(u_n) = \frac{1}{2} z_n^\top J_n H_n z_n + z_n^\top J_n e_1 \|b\|_2.$$

This results in:

$$\mathscr{H}_2(z_n) - \mathscr{H}_2(u_n) = 0.$$

Since the transformation does not change the energy, `SLM` is energy preserving. This will not hold for `KPM` for a couple of reasons. First, H_n is not a Hamiltonian matrix, so equation (2.7) is not a Hamiltonian ODE, and therefore does not have constant energy. The other reason is that the transformation $u_n(t) = V_n z_n(t)$ is not symplectic, and will change the energy. Results are shown in Section 4.6.

To the best of our knowledge, there are no results regarding the energy preservation of the restarted `SLM`.

2.5.3 Linearity of the methods

The Krylov methods require a vector that can be used to generate the orthogonal basis. In the general problem,

$$\dot{u}(t) = Au(t) + b,$$

b is used to create the orthogonal basis. But what if instead of just b, there were some time dependance, eg. $b_1 + b_2 f(t)$, or:

$$\dot{u}(t) = Au(t) + b_1 + b_2 f(t).$$

In this case the Krylov method needs to be used two times, one time to solve $\dot{u}_1(t) = Au_1(t) + b_1$, and another to solve $\dot{u}_2(t) = Au_2(t) + b_2 f(t)$. $u_1(t)$ and $u_2(t)$ can then be added together to solve

the original problem. The reason for this is the need for a vector that can generate the orthogonal space. In this case there is no common vector $\tilde{b}$ so that $\tilde{b}\tilde{f}(t) = b_1 + b_2 f(t)$.

An even bigger problem arises when a differential equation has a source term that is not separable in time and space, see eg. [13] and [15]. As an example, consider the wave equation [18]:

$$\frac{\partial^2 q(t,x,y)}{\partial t^2} = \frac{\partial^2 q(t,x,y)}{\partial x^2} + \frac{\partial^2 q(t,x,y)}{\partial y^2} + g(t,x,y) \quad \text{where } g(t,x,y) \neq p(x,y)f(t).$$

If this equation is discretized with the method described in Section 3.1.1, it would give a unique time dependent function $f(t)_i$, for each point x and y. This results in

$$\dot{u}(t) = Au(t) + \sum_{i=1}^{\hat{m}} e_i f_i(t), \tag{2.14}$$

where $\hat{m}$ is the size of the matrix. This means that equation (2.14) needs to be solved $\hat{m}$ times to obtain the solution if a Krylov method is used, making run times increase. Our earlier studies suggest that the computational cost increases too much with this approach. [15].

2.5.4 A comment on the restart

Ensuring efficient convergence with restarts can be challenging. How this is done is discussed in Section 3.4 and 4.3.

The restart is the reason why the initial conditions need to be zero. In equation (2.7) and (2.12) it is possible to remove the term be_1 by shifting the initial conditions. However, in equation (2.10) and (2.12) the term is time dependent, making the shift impossible. If restarts are used without shifting the initial conditions, the initial condition for the error equation is unknown. Hence, the shift is necessary.

2.5.5 Direct method

Until now, the methods presented do not solve the original problem, but some transformed problem. We will compare these methods to more usual solutions. The method used for com-

Table 2.3: Computational cost of some mathematical operations. n is restart variable, $\hat{m}$ is the size of the full linear system, and k is the number of steps in time. Computational cost for the different operations are found in [21].

Operation	Cost
Integration with forward Euler	$\mathcal{O}(kn^2)$
Integration with Trapezoidal or midpoint rule	$\mathcal{O}(kn^3)$
Arnoldi's algorithm	$\mathcal{O}(n^2\hat{m})$
Symplectic Lanczos method	$\mathcal{O}(n^2\hat{m})$
Transforming from $z_n(t)$ to $u_n(t)$	$\mathcal{O}(\hat{m}nk)$
Matrix vector multiplication $(\hat{m} \times n)$(sparse matrix)	$\mathcal{O}(\hat{m})$
Matrix vector multiplication $(\hat{m} \times n)$ (dense matrix)	$\mathcal{O}(n\hat{m})$

parison is called direct method, or DM. DM uses one of the integration methods presented in Table 2.1 to solve equation (1.1) or (1.2), without the use of any Krylov method.

Assuming that equation (1.1) is used: The energy of DM will be constant with T_s, if an energy preserving method is used. The error will increase linearly with T_s [20]. A goal of this text is to see how the Krylov methods error and energy behave compared to DM.

2.5.6 Number of operations

This section contains a brief discussion about the number of computations for each method presented. A table of computational cost for different mathematical operations is given in Table 2.3.

Table 2.4 shows that the asymptotic cost for KPM and SLM are equal. It is difficult to predict how Windowing compares to this, due to the unknown relation between K and i_r. The difference between the Krylov methods and DM strongly depends on n and i_r. Results are shown in Section 4.7 and 5.5.

Table 2.4: Number of operations needed for the different solving methods when trapezoidal rule is used. i_r is the number of restarts needed for the methods to converge. For windowing $K \cdot k$ is equal to k for the other methods. Windowing with DM is not interesting since it is exactly the same method as DM.

Method	Computational cost for the different methods
KPM	Arnoldi's algorithm, an integration method, and the transformation. $\mathcal{O}((n^2\hat{m} + kn^3 + \hat{m}nk)i_r)$
SLM	Symplectic Lanczos method, an integration method and the transformation. $\mathcal{O}((n^2\hat{m} + kn^3 + \hat{m}nk)i_r)$
DM	An integration method with $n = \hat{m}$. $\mathcal{O}(k\hat{m}^3)$
Windowing (KPM or SLM)	SLM or KPM needs to be run K times. $\mathcal{O}((n^2\hat{m} + kn^3 + \hat{m}nk)i_r K)$

2.6 SLM as a method for eigenvalue problems

It is well known in the literature that the symplectic Lanczos method is mainly a method applied to approximate the eigenvalues of Hamiltonian matrices, see eg. [4],[27], and [16]. This section will explain why this algorithm so attractive for these types of problems.

Eigenvalue problems for large sparse matrices occur in many different areas, eg. control theory, model reduction, and system analysis. SLM is the only method that exploits and preserves these properties. Eigenvalues of Hamiltonian matrices comes in pairs, $\{\pm\lambda\}$, or in quadruples, $\{\pm\lambda, \pm\bar{\lambda}\}$. Since the projected matrix is Hamiltonian, these pairs or quadruples are preserved, and easy to find on the reduced system. Another important property, which is common to all Krylov subspace eigenvalue methods, is that the biggest eigenvalue is found first.

The method is not without flaws. Frequent breakdowns can occur when applying the method to ill conditioned matrices. But much work has been done to overcome the difficulties.The most promising improvements for SLM when applied to eigenvalue problems are based on performing restarts, and there are a number of approaches proposed in the literature, see eg. [4], [27]. This way the numerical estimations can be improved without losing the Hamiltonian structure, and without a too high computational cost. Much work is being done to improve the method further [9], [3].

In this thesis we have considered how SLM can be used to solve linear Hamiltonian ordinary differential equations. Our main comparison method is a similar technique based on the Arnoldi algorithm (KPM). Both methods are implemented with and without restart and their qualitative properties (mainly regarding energy preservation and global error growth) have been investigated.

Chapter 3

Programming

This section explains the test problems, the implementation, and how important results are found.

3.1 Test problems

This section will present the test problems used in the result sections.

Two Hamiltonian matrices are implemented, one is sparse and random, and the other is a discretization of the wave equation. These matrices have two test problems each, one test problem with constant energy, and one test problem with varying energy. All test problems satisfy the condition

$$u(t,0,y) = u(t,1,y) = u(t,x,0) = u(t,x,1) = 0.$$

3.1.1 The wave equation

The first test problem is based on the 2 dimensional wave equation,

$$\frac{\partial^2 \xi}{\partial t^2} = \frac{\partial^2 \xi}{\partial x^2} + \frac{\partial^2 \xi}{\partial y^2} + p(x,y) f(t). \tag{3.1}$$

The spacial domain is the unit square. Each direction is divided in m sub intervals, each of length $h_s = 1/m$, so that $x_i = h_s \cdot i$, $y_j = h_s \cdot j$, with $i, j = 1, \ldots, m-1$. The discretization of the

wave equation can be obtained by approximating $\frac{\partial^2 \xi}{\partial x^2} + \frac{\partial^2 \xi}{\partial y^2}$ with $\tilde{A}$, where $\tilde{A}$ is the five point stencil [22], and approximate $p(x_i, y_j)$ with $\tilde{b}_{i+(m-2)(j-1)}$, where $i, j = 1, \ldots, m-1$. This results in the following system of ODEs:

$$\dot{q}(t) = Iw(t)$$
$$\dot{w}(t) = -\tilde{A}q(t) + \tilde{b}f(t).$$

This can be written as equation (1.1) if $u(t) = [q(t); w(t)]$, $b = [0; \tilde{b}]$, and

$$A = \begin{bmatrix} 0 & I_{\hat{m}} \\ -\tilde{A} & 0 \end{bmatrix},$$

This will be referred to as `wave`, and is a second order approximation of the wave equation. The size of the matrix A is $\hat{m} = 2(m-2)^2$, where m is the number of steps in each spacial direction.

Two test problems are implemented. We consider one test problem with constant energy, which is

$$q(t, x, y) = \sin(\pi x)\sin(2\pi y)\cos(\sqrt{5}\pi t)$$
$$w(t, x, y) = \sin(\pi x)\sin(2\pi y)\sqrt{5}\pi \sin(\sqrt{5}\pi t)$$
$$q_0(x, y) = \sin(\pi x)\sin(2\pi y)$$
$$w_0(x, y) = 0$$
$$f(t, x, y) = 0.$$

We also test the methods on a test problem with varying energy, a non autonomous Hamiltonian system:

$$q(t, x, y) = \sin(\pi x)\, y(y-1)(t^2+1)$$
$$w(t, x, y) = \sin(\pi x)\, y(y-1)(2t)$$
$$q_0(x, y) = \sin(\pi x)\, y(y-1)$$
$$w_0(x, y) = 0$$
$$f(t, x, y) = 2\sin(\pi x)\, y(y-1) - (t^2+1)\sin(\pi x)(2 - \pi^2 y(y-1)).$$

3.1.2 A random test problem

The second implemented Hamiltonian matrix is random, and given by

$$D = \text{rand}(\hat{m}, 1) + 5I_{\hat{m}}$$
$$D1 = \text{rand}(\hat{m} - 1, 1)$$
$$A = J_{\hat{m}}\ \text{gallery('tridiag'}, D1, D, D1)$$

This matrix is referred to as `semirandom`, its size is $\hat{m} = 2(m-2)^2$, which is the same for the wave equation, this is done to easier be able to compare the two methods. The part $5I_{\hat{m}}$ is added to make $J_{\hat{m}}A$ diagonally dominant, this is done to guarantee good convergence of the methods. Our experiments on fully random problems have shown that the convergence is significantly deteriorated.

The test problem is given by

$$u(t, x, y) = \text{unknown}$$
$$u_0(x, y) = \text{rand}(2(m-2)^2, 1)$$
$$f(t, x, y) = 0$$

when the energy is constant, and

$$u(t, x, y) = \text{unknown}$$
$$u_0(x, y) = 0$$
$$f(t, x, y) = \text{rand}(2(m-2)^2, 1) \cdot \text{rand}(1, k),$$

when the energy is varying.

Since we are interested in comparing the different methods to each other, the matrix and the test problems are saved and reused. The analytical solutions to the test problems are in general unknown, but can be computed accurately by an efficient implementation of the matrix exponential and/or of the variation of constants formula. Therefore it is impossible to show convergence in the traditional sense. But there are are some important reasons to use it. It gives a sparse Hamiltonian matrix with much randomness, the randomness makes it difficult for the

Krylov methods to find an approximated solution, which gives more interesting results. Specifically `semirandom` will show more correctly how restarting can improve the solution, compared to `wave` (the discreticatizon of the wave equation given in Section 3.1.1).

The error will be measured as the difference between the solution obtained by a Krylov method and a different method depending on the problem type. The exact solver `diag` will be used if the energy is constant, it will be marked error$^{\text{diag}}$. In the case with varying energy, the exact solvers does not work, in this case the error and energy will be measured as the difference between the Krylov method, and the midpoint rule, and is marked with error$^{\text{comp}}$. This will make the error seem a lot smaller than it really is, but it gives good results.

3.2 Output data

This section will explain how different interesting values are calculated in the program.

All errors are obtained with the same function. At each time step, this function finds the maximum absolute difference between the approximated solution and the exact solution. The resulting vector of absolute errors is then divided by the exact solution, so that the error is relative to the exact solution. This will be marked with "error" on the plots. In the case where the correct solution is unknown, as in `semirandom` (the random test problem implemented in Section 3.1.2), the error is measured as the difference between the numerical approximations produced by `SLM` or `KPM` and `diag` if the energy is constant, or `DM` if the energy is varying. These cases are marked with "error$^{\text{diag}}$" and "error$^{\text{comp}}$", respectively. In some results error$^{\text{diag}}$ and error$^{\text{comp}}$ will also be used with `wave` for comparative reasons.

The energy is found with one dedicated energy function. The energy is calculated for each time step, and then initial energy is subtracted from these values to obtain the absolute energy error. When comparing approximated energy with analytical energy, the energies are calculated independently, and then subtracted from each other. This is done for all energies in Chapter 5. If the energies are compared to the analytical solution they are marked "energy", and "energy$^{\text{comp}}$" if

they are compared to DM.

Other interesting results are the number of restarts, and computation time. The number of restarts is 1 if a Krylov method is used without restart. Any restart is then counted and added to this. The reason for this is that when plotting the number of restarts on a logarithmic scale MATLAB removes all zeros from the plot. The symbol for the number of restarts will be i_r. Computation times are measured as the time it takes to calculate the solution of the test problem. Calculations common to all methods are not included. The symbol for computation time is T_c.

3.3 Figure

All figures are plotted with a dedicated plot tool. This makes it easier to change plots, or uncover programming faults. The program uses the solver to obtain data described in section 3.2. Each run of the solver then gives information about one point on the figure, this means that the solver is run several times for each plot. This removes noisy data, and makes it easy to choose the position of each point.

All labels and legends are generated with a dedicated label tool. On some plots, a black solid line is placed. This line is used to show trends, it is marked with "$\propto \xi^a$", where $a \in \mathbb{N}$ is the increase, and ξ is the data the line is increasing with. SLM and KPM will on pictures be called KPM(n) and SLM(n), where n is the restart variable.

3.4 Implementation

All algorithms and methods are implemented in MATLAB R2014b on an ubuntu 14.04 LTS computer with intel i7 4770 CPU and 16 GB of RAM. The program is divided into several small functions, each function is individually tested. Functions are reused as much as possible. MATLAB's backslash operator is used to solve the linear systems in the trapezoidal rule and the midpoint rule, sparse matrices are used where possible.

There are two ways to implement the number of restarts the Krylov methods should perform. One way is to choose a number of iterations and hope it converges. The other method is to iterate until the change in the solution is below a certain threshold tolerance. Both of these methods are implemented, but mostly the last will be used since this gives more information about how well the solution is approximated. The tolerance will be called ι (iota) since iota is used to describe something small[2].

All codes used to create results in the text can be found on github:
https://github.com/sindreka/Master

Chapter 4

Results for test problems with constant energy

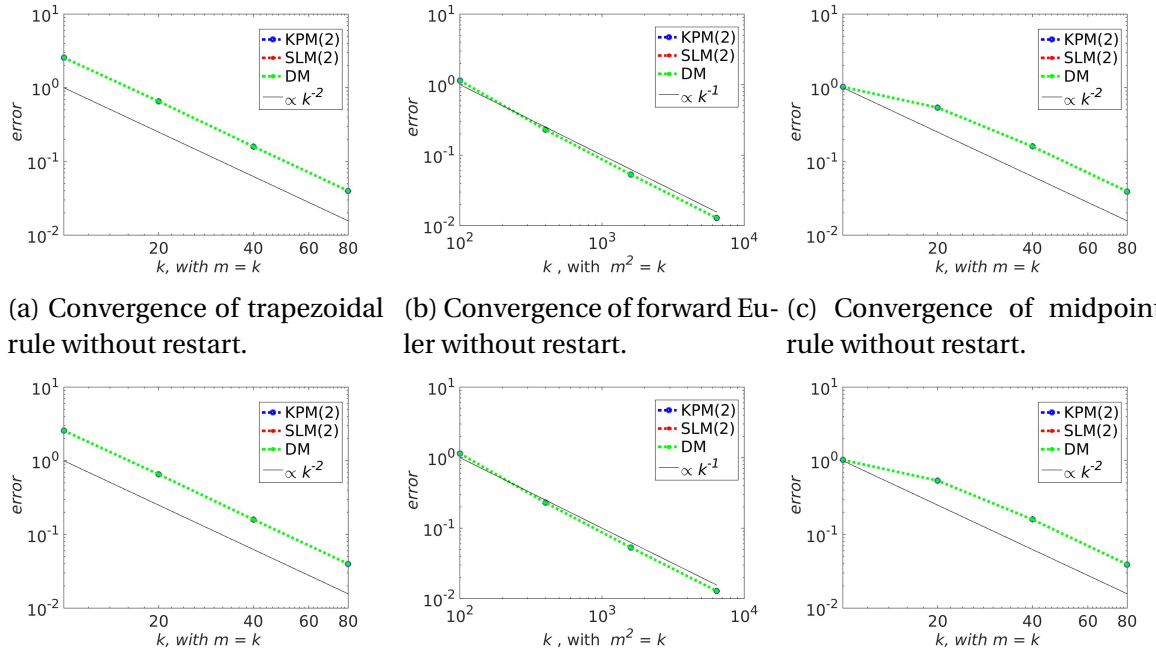
This chapter will show how error, energy and computation times changes with the number of restarts i_r, the tolerance ι, the time domain T_s, the number of steps in time k, the number of points in each spacial direction m and the restart variable n. We are especially interested in seeing how the energy for `SLM` behaves, and how `KPM` and `SLM` differ. The predictions and proofs made in the theory chapter will also be tested.

Due to the computational complexity of finding an exact solution with `diag`, which is needed when calculating `semirandom`'s error, fairly small matrices are used. Unless stated otherwise, assume that $m = 8$, which makes the size of the matrix $\hat{m} = 2(m-2)^2 = 72$, and $k = 20$ per second.

4.1 Convergence

This section shows convergence for the different methods, `wave` is used since it is necessary to know the analytical solution. The time domain, $T_s = 1$ second for all plots in this section.

4.1.1 Convergence with numerical integration



(a) Convergence of trapezoidal rule without restart.

(b) Convergence of forward Euler without restart.

(c) Convergence of midpoint rule without restart.

(d) Convergence of trapezoidal rule with restart.

(e) Convergence of forward Euler with restart.

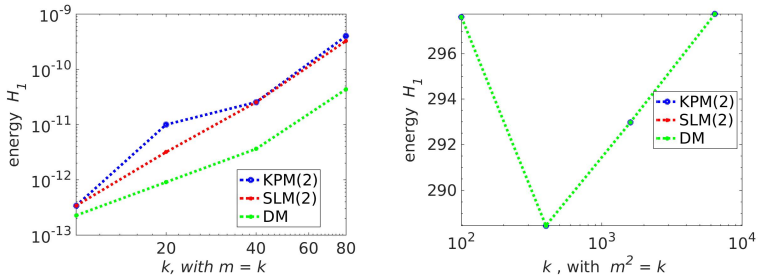
(f) Convergence of midpoint rule with restart.

Figure 4.1: Convergence for the different integration methods. Notice that forward Euler uses k^2 points in time where the other methods use k points in time to obtain the same accuracy. $n = 2$ is kept constant for all plots, since smaller n generally gives poorer convergence. For the figures with restart enabled: the tolerance $\iota = 1e-10$. 1 second is simulated.

Figure 4.1 shows that all methods converge with the expected rate. The trapezoidal rule and the midpoint rule converges quadratically and near identically, the trapezoidal rule and the midpoint coincide on linear constant coefficient problems. So on such Hamiltonian systems they are both symplectic. Since the midpoint rule also is symplectic on problems with non constant coefficients, it will be used when restart is enabled. The trapezoidal rule has a faster run time, and will be used when restart is not enabled due to insignificant differences in error. Both the midpoint and the trapezoidal rule require the solution of one linear system per time step. The linear systems are solved with the function "backslash" in MATLAB.

Forward Euler has a slow computation time due to the larger k needed for the same convergence. It also has a tendency to diverge on longer time intervals and poorly approximates the

energy, which is shown Figure 4.2 for trapezoidal rule and forward Euler. The poorly approximated energy is reason enough not to use forward Euler.

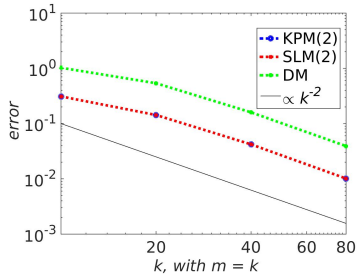


(a) Energy error for trapezoidal rule.

(b) Energy error for forward Euler.

Figure 4.2: A figure showing the difference in energy between trapezoidal rule and forward Euler. 1 second is simulated and restart is not enabled.

4.1.2 Convergence with an exact solver



(a) Convergence of the exact solver without restart.

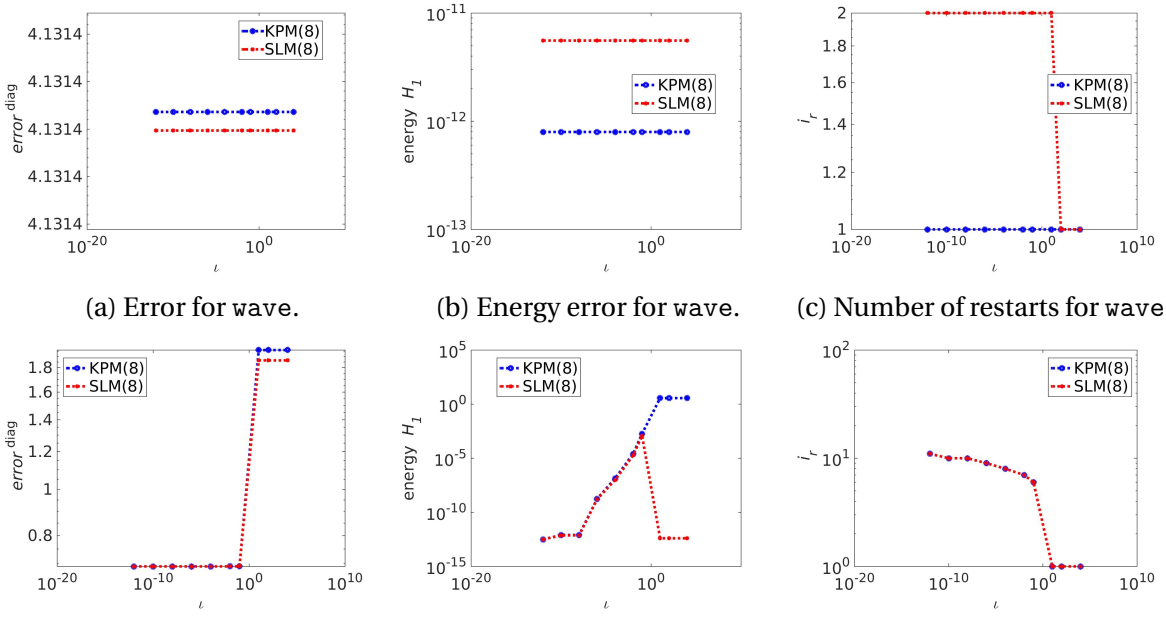
Figure 4.3: Convergence for exact solvers. `DM` uses the trapezoidal rule, and is included to show the difference between this figure, and Figure 4.1. Note that the exactness of the method is for the integration in time, the spacial step-size is kept constant. The expected convergence is therefore still quadratic with m. $n = 2$ because taking the matrix exponential is a costly operation and it is harder for the methods to converge when n is small. 1 second is simulated. The exact solver is defined in Table 2.2.

Figure 4.3 shows that the convergence is quadratic for the exact solver.

4.2 Convergence with the restart

This section will show how error and energy change with the tolerance ι and the number of restarts i_r.

4.2.1 Convergence as a function of ι



(a) Error for `wave`. (b) Energy error for `wave`. (c) Number of restarts for `wave`.

(d) Error for `semirandom`. (e) Energy error for `semirandom`. (f) Number of restarts for `semirandom`.

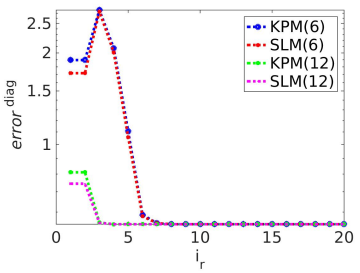
Figure 4.4: These figures show how choosing different tolerances ι affects the solution. The pictures on the top are for `wave` and the pictures at the bottom are for `semirandom`. This plot considers 10 seconds and uses the midpoint rule. Remember that $\text{error}^{\text{diag}}$ is the relative difference between the specified method (eg. `KPM` or `SLM`), and an exact solver.

Figure 4.4 shows that if `wave` is used, there is no reason to use the restart. From Figure 4.1 we conclude that this also is the case for larger m. `wave` will therefore not be used again until Chapter 5. If you are interested in using a Krylov method on the wave equation with constant energy, it can be very well utilized. The largest orthogonal space needed is about $n = 2$ (depending on T_s, see Section 4.3), which gives a very low run time. The rest of this chapter will only consider `semirandom`, since this better shows the difference between using restarts and not.

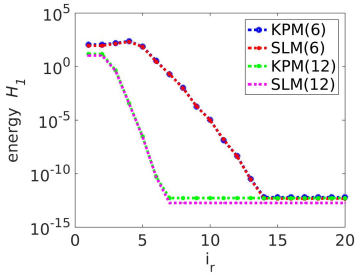
Figure 4.4f shows that the number of restarts are the same for KPM and SLM. Figure 4.4d shows that there is little reason to restart after gaining a certain precision, since the changes will not be visible. Based on Figure 4.4e i conclude that $\iota = 1e-6$ is a suitable tolerance.

Figure 4.4d shows that both KPM and SLM need to restart to gain a low error. But Figure 4.4e shows that the energy for SLM increases the first times it restarts, before it starts sinking again. This shows that the energy can be accurately estimated by SLM without restart, but not the error. This seems to indicate that the restart procedure ruins the energy preservation property of SLM, but many restarts can better the approximation of the energy. The energy of KPM can also be estimated to machine accuracy, even though its initial energy is quite high.

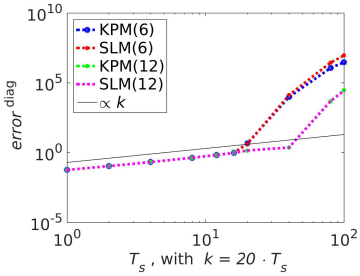
4.2.2 Convergence as a function of i_r



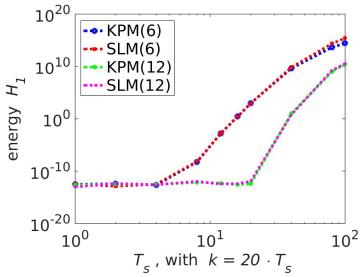
(a) Error as a function of i_r.



(b) Energy error as a function of i_r.



(c) Error as a function of time with constant $i_r = 10$.



(d) Energy error a function of time with constant $i_r = 10$.

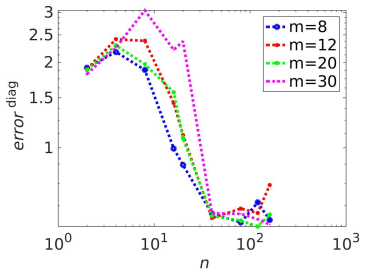
Figure 4.5: The top figures show how the error and energy change as a function of the number of restarts i_r. The bottom pictures show how error and energy behave with increasing time domain, with constant $i_r = 10$, and $T_s = 10$.

Figure 4.5a and 4.5b show that for KPM(6) and SLM(6) it is necessary to perform more than 4 restarts to gain accuracy w.r.t. error. After 8 restarts the change in the solution is too small to observe. This shows why it is wise to use ι and not i_r as convergence criterion. If ι is used, the Krylov methods can perform the exact number of iterations needed to converge, with any n and T_s. If i_r is used as convergence criterion, it needs to be changed with n and T_s. Figure 4.5c shows that the error increases linearly with time for the first few points, as predicted in 2.5.5. Figure 4.5d shows that the energy increases, SLM shows no energy preserving property in this case.

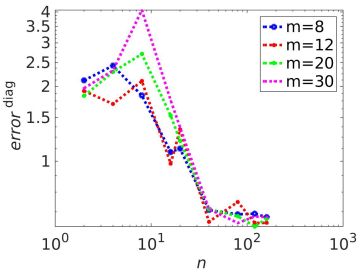
4.3 How to choose the restart variable n

This section will look at how n can be chosen to avoid unnecessary restart, and still give satisfactory results. Figure 4.4d and 4.4e shows that the energy and error behaves quite similarly, except when $i_r = 0$ for SLM. If only energy preservation is important, there is no need to use KPM, or to wisely choose an n. This section will therefore only consider the error. How n should change with the size of the matrix A (which is given by $\hat{m} = 2(m-2)^2$, see Section 3.1.1 for a more thorough explanation), and the length of the time domain T_s, is explored. We assume that all cases tested here are independent.

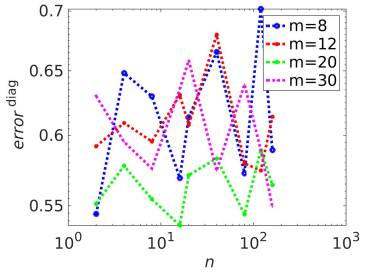
4.3.1 Restart variable as a function of m



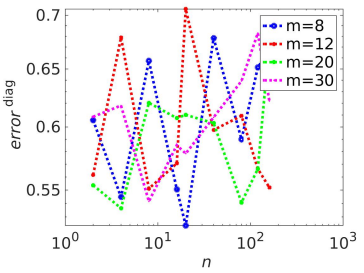
(a) Error for KPM without restart.



(b) Error for SLM without restart.



(c) Error for KPM with restart.

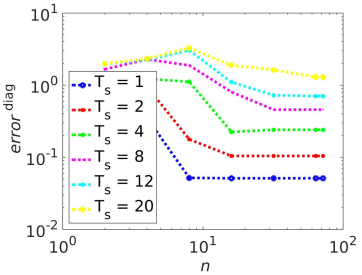


(d) Error for SLM with restart.

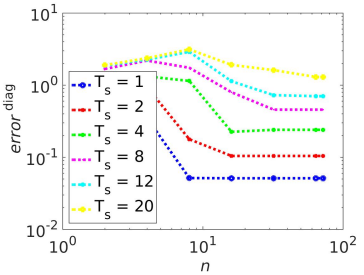
Figure 4.6: The pictures show which n gives convergence for different m, where the size of the matrix A, is given by $\hat{m} = 2(m-2)^2$. The top pictures are without restart, and the bottom pictures are with restart. 10 seconds is simulated.

Figure 4.6a and 4.6b show that n can be choose independently of m, as long as $n \geq 40$ when restart is not enabled. Figure 4.6c and 4.6d show that if restart is enabled, a good approximation of the solution can be found with any n, for any m. The reason for the independence between m and n can be due to the structure of the matrix.

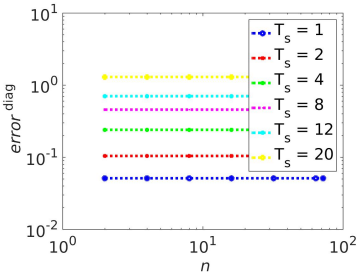
4.3.2 Restart variable as a function of T_s



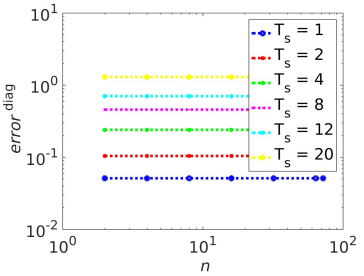
(a) Error for KPM without restart.



(b) Error for SLM without restart.



(c) Error for KPM with restart.



(d) Error for SLM with restart.

Figure 4.7: The pictures show which n gives convergence for different T_s. The top plots are without restart, and the bottom pictures are with restart.

Figure 4.7a and 4.7b show that the length of the time domain is affecting the optimal choice of n if restart is not enabled. In this case n should increase linearly with T_s. n and T_s are independent when restart is enabled. The lines are not overlapping due to the linear increase in error over time.

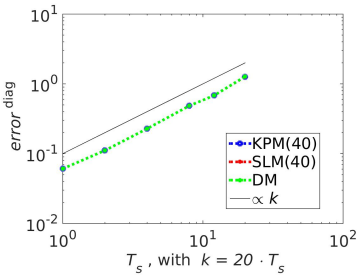
SLM and KPM behave very similarly in all pictures shown in this section, thus the values of n will be the same for the different methods. Based on Figure 4.6 and 4.7, this text will consider $n = 40$ when restart is not enabled, and $n = 8$ when restart is enabled.

4.4 Energy and error

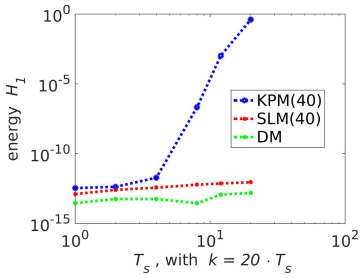
This section will show how the error and energy of SLM, KPM and DM changes as a function of time. Restarts and windowing will also be tested.

4.4.1 Energy and error as a function of time

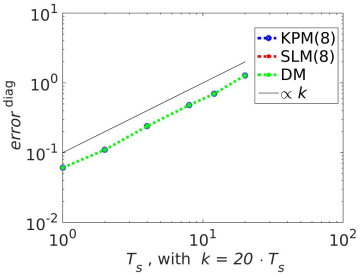
In this case, the midpoint rule is used if the restart is enabled, while the trapezoidal rule is used if the restart is not enabled. Windowing is not considered.



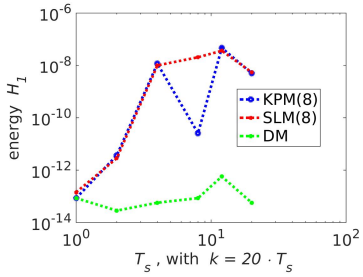
(a) Error without restart.



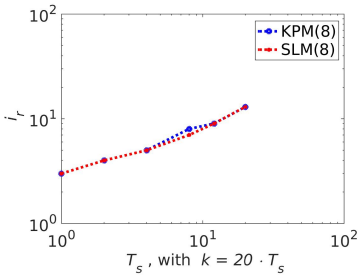
(b) Energy error without restart.



(c) Error with restart.



(d) Energy error with restart.



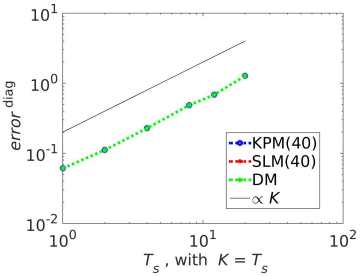
(e) Number of restarts.

Figure 4.8: The figures show how the error and energy changes with as a function of time. The top pictures are without restart, and the bottom pictures are with restart and $\iota = 1e-6$.

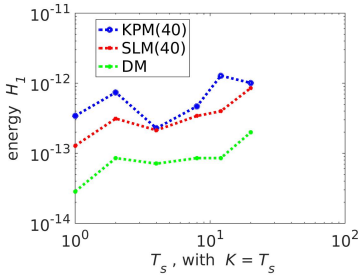
Figure 4.8 shows that the error for all methods are very similar, and increases linearly. Figure 4.8b shows that KPMs energy increases very fast, while SLM and DMs energy remains constant. When restart is enabled, SLM and KPMs energy are very similar, and a little worse than SLMs energy without restart. The number of restarts increases as a function of T_s.

4.4.2 Energy and error as a function of time for windowing.

Windowing (see Section 2.4) is now used together with the Krylov methods. The K windows each have a length 1 second. The window is chosen fairly small based on the energies in Figure 4.8b and 4.8d, in these figures, the energy increases when T_s becomes too large.

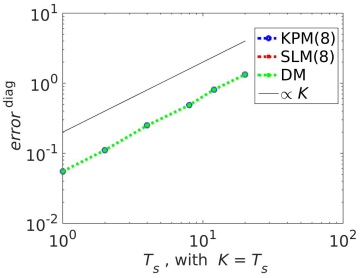


(a) Error without restart.

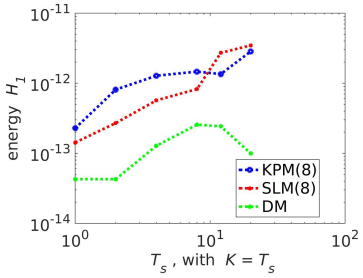


(b) Energy error without restart.

.



(c) Error with restart.



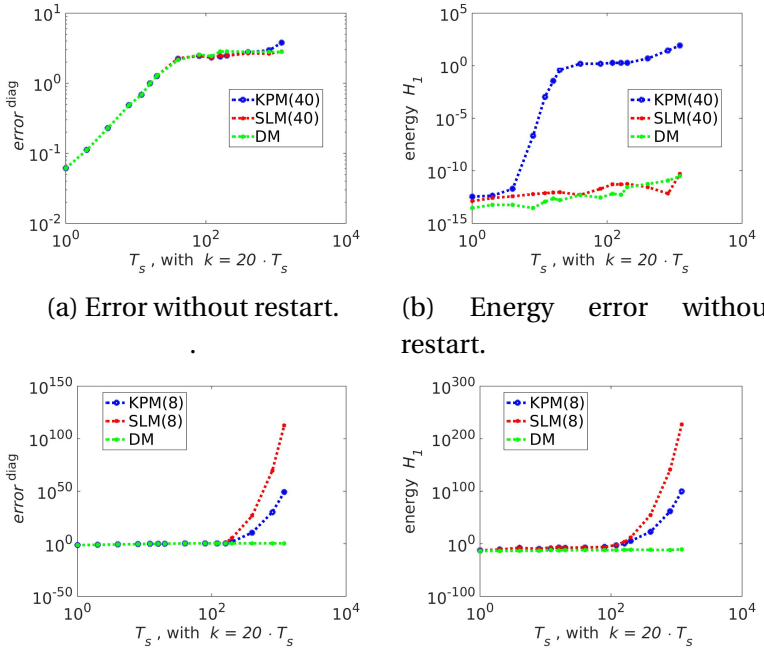
(d) Energy error with restart.

Figure 4.9: The figures show how the error and energy change over time when windowing is used. The top pictures are without restart, and the bottom pictures are with restart and $\iota = 1e-6$.

Figure 4.9 shows that when windowing is enabled, the increasing energy of `KPM` that occurred on Figure 4.8b disappears. If restarting is not enabled, the error increases linearly, and the energy is preserved for all methods. The energy is slowly increasing when restart is enabled. The error and energy with windowing is comparable to Figure 4.8. This makes windowing an interesting idea.

4.4.3 Behavior of energy and error on long time domains

In this section it is shown what happens when the time domain becomes to large for the Krylov methods to converge.



(a) Error without restart.

(b) Energy error without restart.

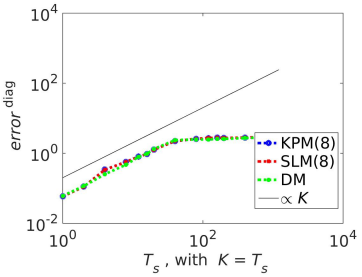
(c) Error with restart.

(d) Energy error with restart.

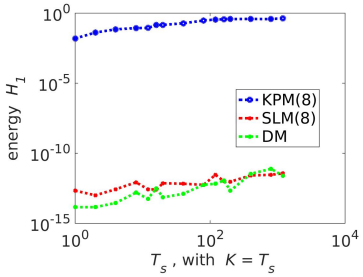
Figure 4.10: This figure shows how different methods perform on a long time domain.

Figure 4.10a shows that, without restart, the increase in error is linear until the numerical value for the errors is about the same size as the numerical value of the exact solution, ei. ~ 1. The energy for `KPM` increase to about 1 before flattening out, while the energy for `SLM` and `DM` is near constant. Actually the energy for `SLM` and `DM` is near constant for all values of T_s, m, and n. Figure 4.10c and 4.10d shows that when restart is enabled, the energy and error for the Krylov methods diverges. This means: Do not use a Krylov method on a long time domain when restart is enabled, without increasing n.

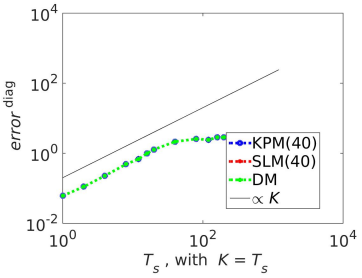
Figure 4.11a and 4.11c show that if windowing is used, the trend of increasing error continues as it did on Figure 4.9, until it becomes constant, around 100 seconds. Figure 4.11a shows that `KPM`s energy is poorly approximated if restart is not enabled. When restart is enabled the energy increases slowly for all methods, suggesting the numerical round off errors are to blame. Windowing can effectively be used to avoid the divergence happening on long time domains when restart is enabled.



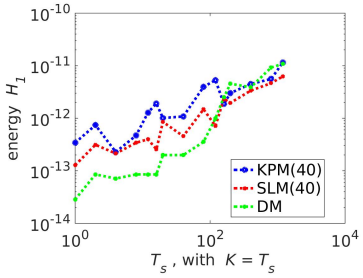
(a) Error for windowing without restart.



(b) Energy error for windowing without restart.



(c) Error for windowing with restart.



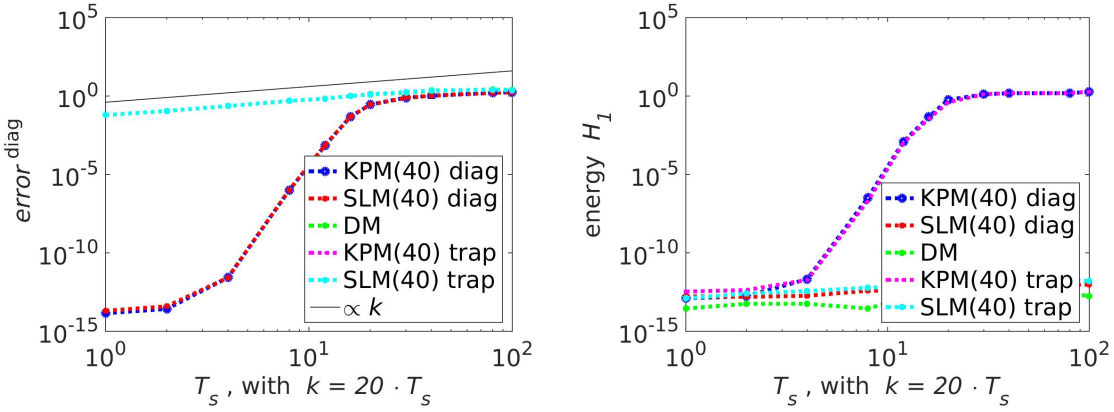
(d) Energy error for windowing with restart.

Figure 4.11: This figure shows how windowing performs on a very long time domain.

4.5 Energy and error with exact solvers

This section will show how using an exact solver can improve the error and energy, compared to the trapezoidal rule. It is also shown how restarting with midpoint rule changes the energy and error after using an exact solver, and how windowing behaves with an exact solver.

4.5.1 Exact solver vs. numerical integration



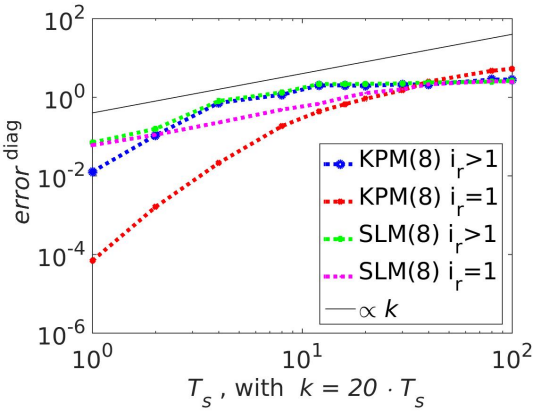
(a) Error as a function of time.

(b) Energy error as a function of time.

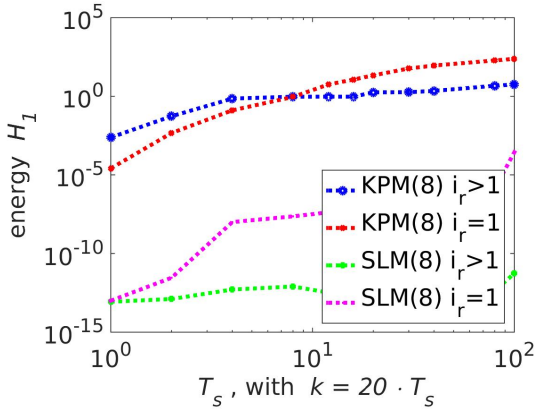
Figure 4.12: A figure showing how the error and energy changes over time when an exact solver (`diag`) is used. Restart is not enabled, and `DM` uses the trapezoidal rule.

Figure 4.12a shows how the Krylov methods compare to each other with an exact solver and with the trapezoidal rule. `diag` makes it possible for the Krylov methods to be within machine accuracy of the exact solution, thou only for a short time. After about 11 second the difference in error between `trap` and `diag` is minimal, in this case the leading error is caused by the Krylov method, and not the exact solver. Figure 4.12b shows that using an exact solver does not change the energy, compared with the trapezoidal rule.

4.5.2 Exact solver with restart



(a) Error with and without restart.

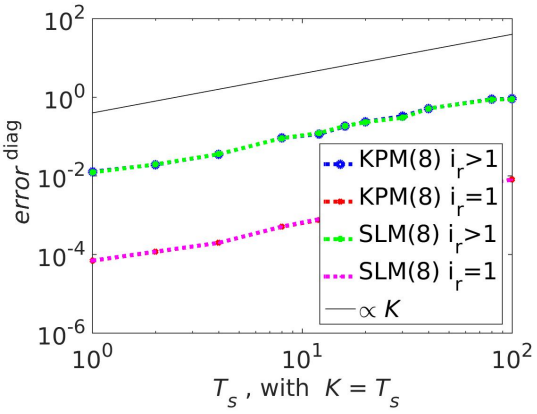


(b) Energy error with and without restart.

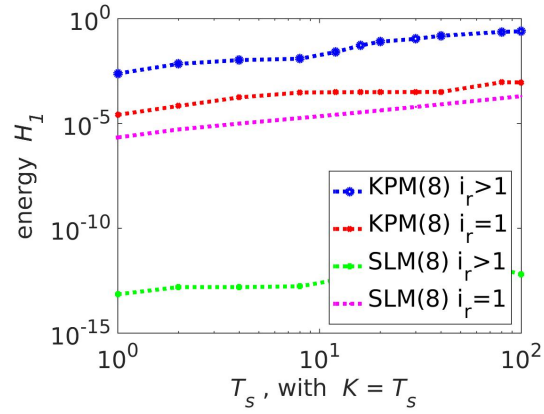
Figure 4.13: A figure showing how restarting after using an exact solver changes the error and energy. $i_r = 1$ means that restart is not used, while $i_r > 1$ means that restart is used.

Figure 4.13 shows that the difference between restarting and not is quite small when an exact solver is used. It seems that restarting gives better results at the end of the time domain, while not restarting gives better results at the beginning of the time domain. However, this effect is not large enough to conclude anything definite.

4.5.3 Exact solver with windowing



(a) Error with and without restart.



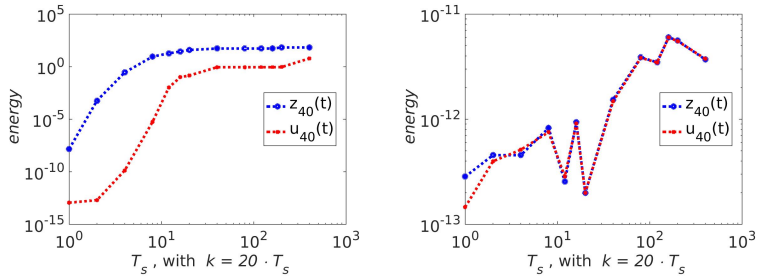
(b) Energy error with and without restart.

Figure 4.14: A figure showing how windowing with an exact solver changes the error and the energy. $i_r = 1$ means that restart is not used, while $i_r > 1$ means that restart is used.

Figure 4.14a shows that the error is linearly increasing, this makes windowing with an exact solver better than just the exact solver after 10 seconds, but before this, the exact solver much better. Figure 4.14b shows that the energy becomes poorly approximated after a few seconds, only SLMs energy with restart is constant and near machine accuracy.

4.6 Energy in the transformations

This section will show how the energy changes when transforming from $z_n(t)$ to $u_n(t)$. For KPM $u_n(t) = V_n z_n(t)$, and for SLM $u_n = S_n z_n(t)$. The difference is the matrix V_n and S_n, where V_n is an orthonormal matrix, and S_n is a symplectic matrix. It is predicted that the transformation with S_n would be energy preserving, no predictions is made regarding the energy for the transformation with V_n. For more information see Section 2.5.1 and 2.5.2.



(a) Energy error for KPM(40). (b) Energy error for SLM(40).

Figure 4.15: The figures show how the transformation between $z(t)$ and $u(t)$ changes the energy. Restart is not enabled.

For KPM there is a huge discrepancy between the energy $\mathscr{H}_2$ for $z_n(t)$, and $\mathscr{H}_1$ $u_n(t)$, see Section 2.2 for more information about the energy. For SLM there is no difference between the two. This shows the transformation with S_n is symplectic. It is worth mentioning that $S_n J_{\hat{m}} S_n - J_n \approx 1e{-}16$ and $V_n^\top V_n - I_n \approx 1e-11$, when $m = 20$ and $n = 200$.

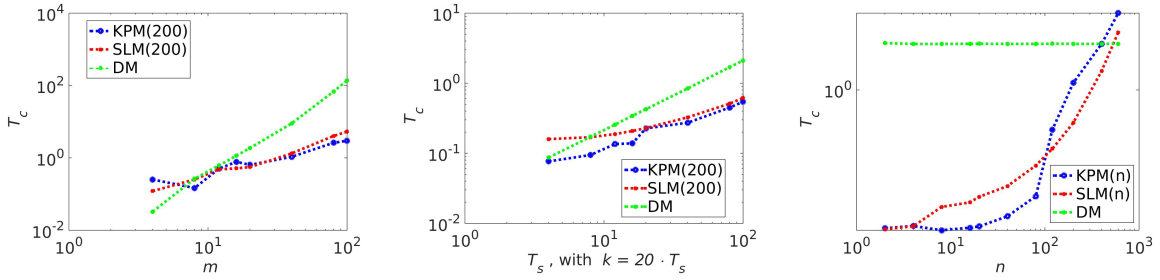
4.7 Computation time

This section will compare computation time for the different methods discussed. Matrices with the size $m = 8$ are far too small to show optimal computation times for the Krylov methods.

Therefore assume that $m = 20$. $k = 20$ is still suitable. The restart variables for this size is found using Section 4.3 with $T_s = 100$, this gives $n = 200$ when restart is not enabled, and $n = 20$ when restart is enabled. An additional reason to pick $n = 20$ when restart is not enabled is that the number of iterations decreases as n increases, as you will see, $n \sim m$ is a good rule for optimal computation times.

4.7.1 Computation time for the numerical integration

In this case, the trapezoidal and the midpoint rule are used, without windowing. Computation time for different m, k and n is shown.



(a) Computation time as a function of m without restart.

(b) Computation time as a function of k without restart.

(c) Computation time as a function of n without restart.

(d) Computation time as a function of m with restart.

(e) Computation time as a function of k with restart.

(f) Computation time as a function of n with restart.

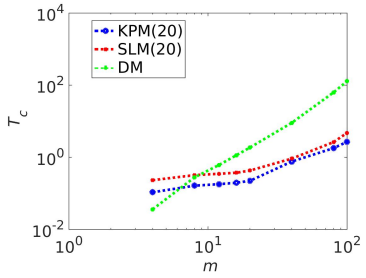
Figure 4.16: A figure of computation times with and without restart for different m, k and n. Assume that $n = 200$, $T_s = 100$, $k = 2000$, and $m = 20$.

Figure 4.16 shows that the computation time for all methods, except `KPM` with restart, increases linearly with k. `KPM`'s computation time seems to be increasing quadratically with k. The computation time for `DM` increases quadratically with m. `SLM` is faster than `DM` if restart is not used, but slower if restart is used. The difference between `KPM` with and without restart is minimal, and is similar to `SLM`'s computation time without restart. Both `KPM` and `SLM` are fastest without

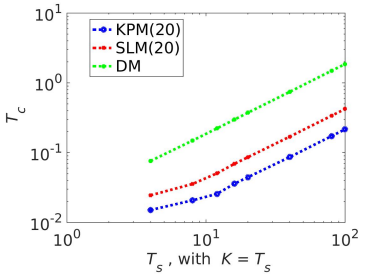
restart. KPM can also be fast with restart if m is large, k is small, and n is well chosen. Figure 4.16f and 4.16c show that choosing n optimal is very important for the Krylov methods to achieve good run times. $n \approx m$ is a good rule when restart is enabled, thou this rule is less important for SLM. When restart is not enabled, smaller n gives lower computation times.

4.7.2 Computation time with windowing

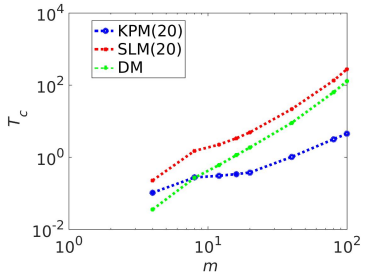
In this case, the trapezoidal and the midpoint rule are used, with windowing. Computation time for different m and k is shown.



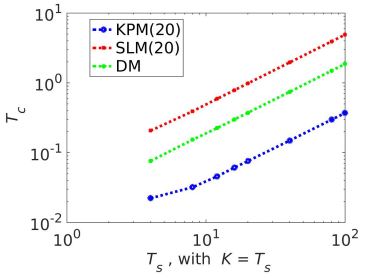
(a) Computation time as a function of m without restart.



(b) Computation time as a function of k without restart.



(c) Computation time as a function of m with restart.



(d) Computation time as a function of k with restart.

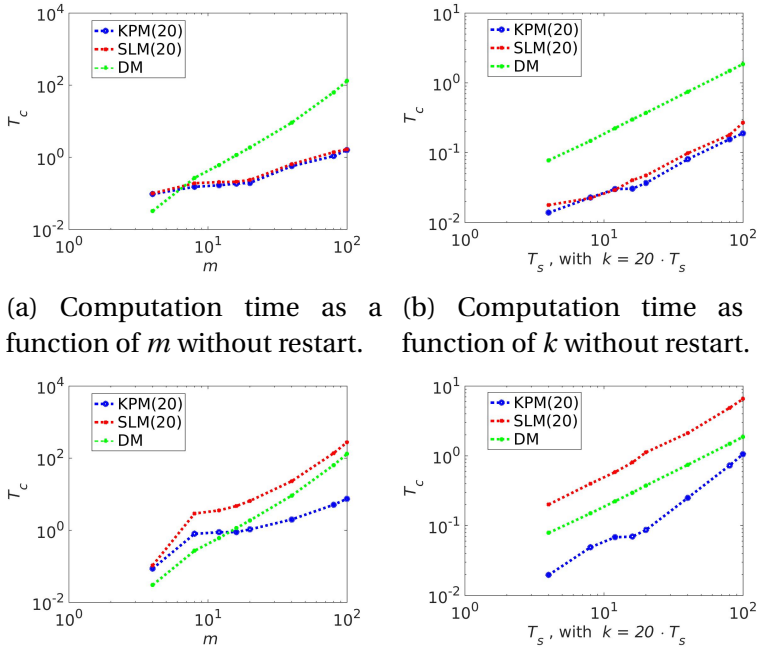
Figure 4.17: A figure of the computation times when windowing is used. Assume that $n = 20$, $T_s = 100$, $k = 20$ per second, and $m = 20$.

Figure 4.17 shows that the computation time for all methods increase linearly with time. There is a big difference between restarting and not for SLM, where SLM should avoid the restart, since it then is slower than DM. For KPM the difference between restarting and not is minimal, and it is always faster than DM.

One great thing with the computation times with windowing for KPM is that, even with restart,

the computation times increase linearly with time, making KPM faster than DM for all values of k and m.

4.7.3 Computation time with diag



(a) Computation time as a function of m without restart.

(b) Computation time as a function of k without restart.

(c) Computation time as a function of m with restart.

(d) Computation time as a function of k with restart.

Figure 4.18: A figure of the computation times with diag. Assume that $n = 20$, $T_s = 100$, $k = 2000$, and $m = 20$.

Figure 4.18 shows that using an exact solver is equally as fast as using any of the other approaches discussed. This, together with the small error, makes exact solvers very desirable on small time domains. SLM with restart has the same problem as in the other cases, where the restart makes it very slow. KPM also has the same problem as before, where computation time increases faster than linear when restart is enabled.

No plots of time consumption with windowing and an exact solver is shown, because the results are very similar to the ones shown in Figure 4.17.

Chapter 5

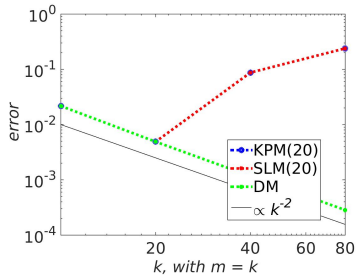
Results for test problems with varying energy

In this chapter we will investigate the performance of `SLM` and `KPM` when applied to non autonomous Hamiltonian systems of type (1.2). The error and energy as a function of time T_s, together with computation time as a function of m and k, is shown. k is the number of steps in time, and m is used to calculate the size of the matrix $\hat{m}$, given by the following relation $\hat{m} = 2(m-2)^2$. We also make a numerical study to find out what the best strategy for choosing the restart variable n is.

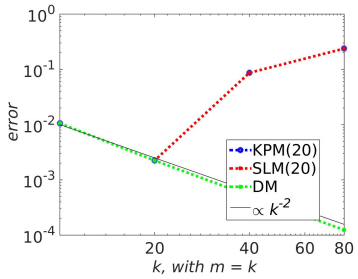
The exact solver in Table 2.2 does not work in this case due to the presence of the time dependent source term in equation (1.2). When `semirandom` is used, the error and energy will be measured as the difference between `DM` and the Krylov method used. This will be marked with error$^{\text{comp}}$ and energy$^{\text{comp}}$. This makes the error seem a lot smaller than it really is. Another approach that could be used is to apply the midpoint rule with a very small step-size, this has not been considered here.

Since it is interesting to compare the results in this chapter to the previous chapter, all values will remain the same, meaning: $m = 8$, $n = 8$ with restart, $n = 40$ without restart, and $k = 20$ per second, unless stated otherwise. The tolerance $\iota = 1e-6$ is used on all figures with restart.

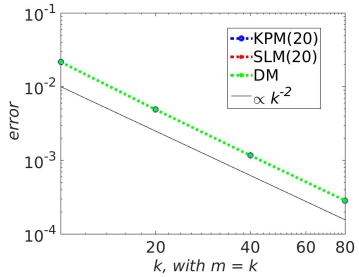
5.1 Convergence with m and k



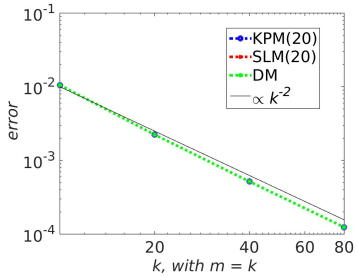
(a) Convergence for `wave` with trapezoidal rule without restart.



(b) Convergence for `wave` with midpoint rule without restart.



(c) Convergence for `wave` with trapezoidal rule and restart.

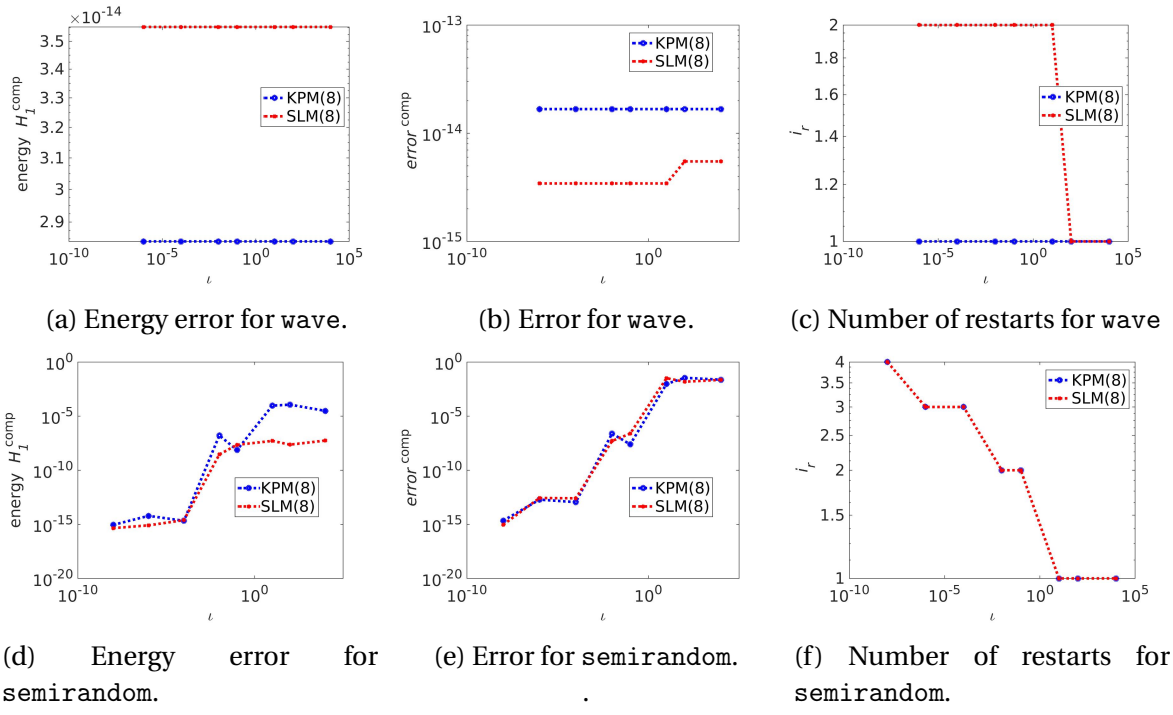


(d) Convergence for `wave` with midpoint rule and restart.

Figure 5.1: Convergence plot with different integrators, simulated over 1 second. The top pictures are without restart, the bottom pictures are with restart and $\iota = 1e-6$.

Figure 5.1 shows that without restart, the methods does not converge with $n = 20$, while they converged with $n = 2$ with constant energy (see Figure 4.1). Clearly convergence is a lot harder to achieve in this case, though the restart can effectively be used to ensure this. With restart, all methods converge quadratically. The midpoint rule performs better than the trapezoidal rule, therefore only the midpoint rule will be used further in this chapter.

5.2 Convergence with ι



(a) Energy error for `wave`.

(b) Error for `wave`.

(c) Number of restarts for `wave`

(d) Energy error for `semirandom`.

(e) Error for `semirandom`. .

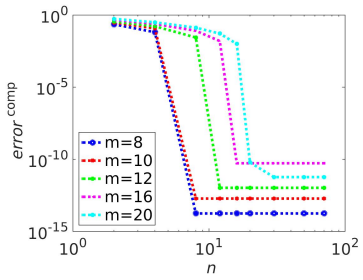
(f) Number of restarts for `semirandom`.

Figure 5.2: These figures show how restarting changes energy and error. The pictures on the top are for `wave` and the pictures on the bottom are for `semirandom`. The time integration is over a time interval of 1 second.

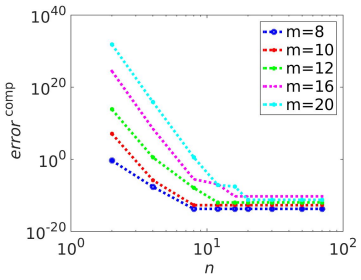
Figure 5.2 looks very similar to Figure 4.4, one important difference is that the energy for `SLM` no longer starts at $1e-15$, suggesting that `SLM` cannot estimate the energy precisely, without decreasing the error. The two Krylov methods behave very similarly.

`wave` has no need for the restart, while `semirandom` can have a great decrease in error and energy. In the interest of showing how the restart can best be utilized, only `semirandom` is used in the rest of this chapter.

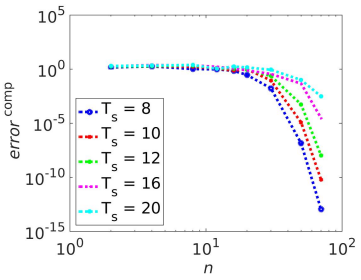
5.3 How to choose the restart variable n



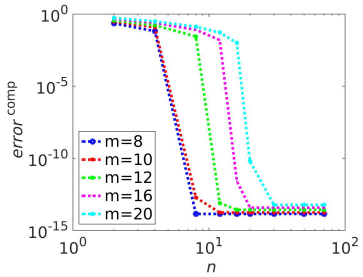
(a) Error for KPM without restart, for different m.



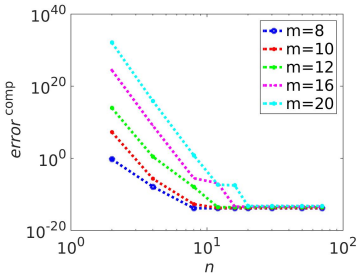
(b) Error for KPM with restart, for different m.



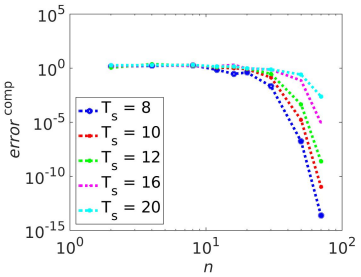
(c) Error for KPM without restart, for different T_s.



(d) Error for SLM without restart, for different m.



(e) Error for SLM with restart, for different m.



(f) Error for SLM without restart, for different T_s.

Figure 5.3: The pictures show which n gives convergence for different m and T_s. The size of the interval of integration is 10 seconds.

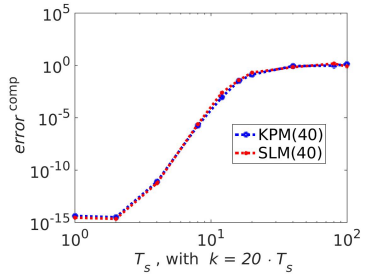
Figure 5.3 shows that n depends linearly on both m and T_s. This explains why the divergence in Section 5.1 occurs: m becomes too large without n increasing. If n is too small, restarting will make the error increase.

The values chosen based on this picture will remain $n = 8$ when restart is not enabled, and $n = 40$ when restart is enabled. The reason for the large n when restart is not enabled is that $T_s = 100$ is considered later.

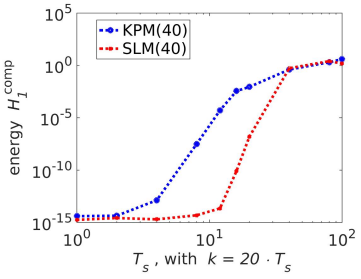
5.4 Energy and error as a function of time

This section will show how error and energy changes as a function of time.

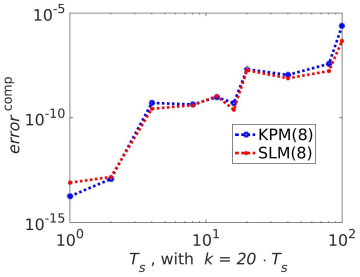
5.4.1 Energy and error without windowing



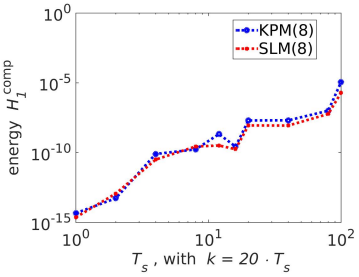
(a) Error without restart.



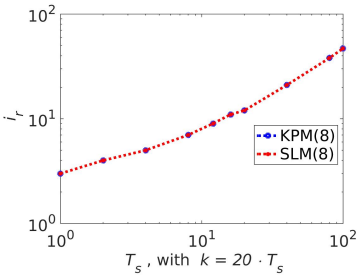
(b) Energy error without restart.



(c) Error with restart.



(d) Energy error with restart.

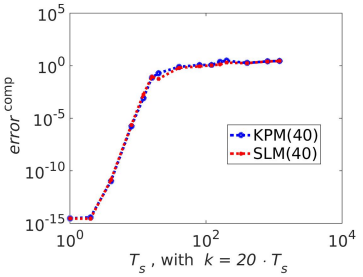


(e) Number of restarts.

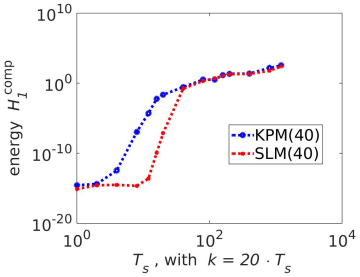
Figure 5.4: The pictures show the change in the error and energy as a function of time. Restart is not enabled for the top pictures, restart is enabled for the bottom pictures.

Figure 5.4 shows that SLM preserves the energy a little longer than KPM when restart is not enabled. Apart from that, the methods are almost identical. These pictures look very similar to Figure 4.12, this is because we only look at the error the Krylov methods create, and not the error made by the time integration method. The restart gives a much better estimate of the error and energy throughout the time domain, than when restart is not enabled.

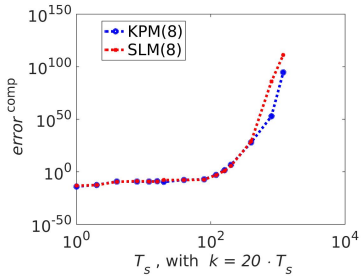
Figure 5.5 shows the divergence occurring on long time domains for the Krylov methods. Figure 5.5e shows the linear increase in the number of restarts.



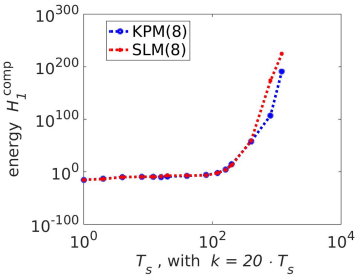
(a) Error without restart.



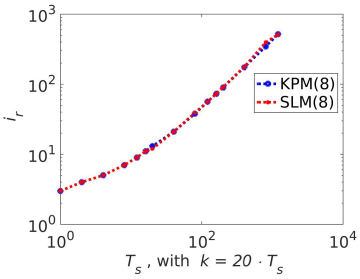
(b) Energy error without restart.



(c) Error with restart.



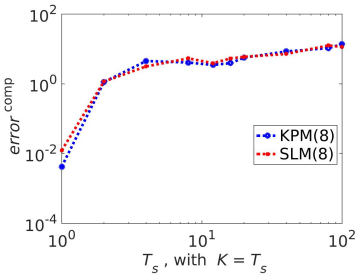
(d) Energy error with restart.



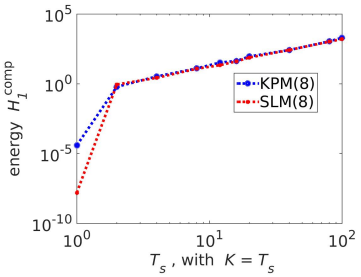
(e) Number of restarts.

Figure 5.5: The pictures show the change in error and energy over a long time domain. Restart is not enabled for the top pictures, restart is enabled for the bottom pictures.

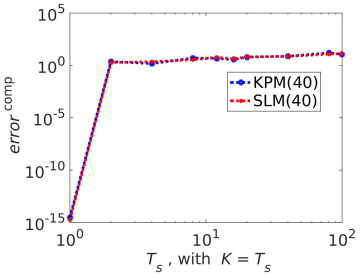
5.4.2 Energy and error with windowing



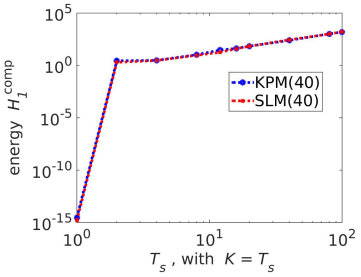
(a) Error without restart.



(b) Energy error without restart.



(c) Error with restart.



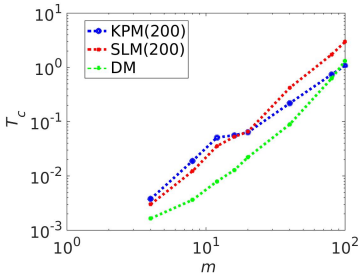
(d) Energy error with restart.

Figure 5.6: The change in error and energy as a function of time with windowing.

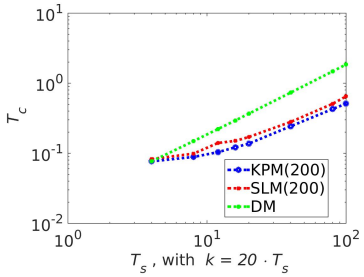
Figure 5.6 shows that windowing does not work with varying energy, thus a very promising solution strategy from the previous chapter has a limiting factor.

5.5 Computation time

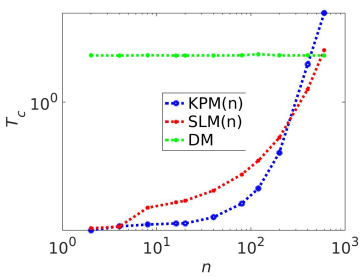
Computation times are somewhat different in this case, compared to the case with constant energy. Windowing is not included since the method did not give the correct answer.



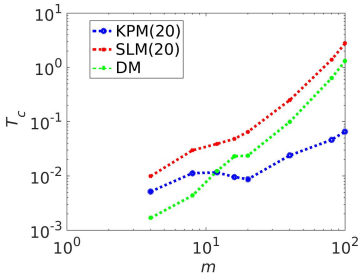
(a) Computation time as a function of m without restart.



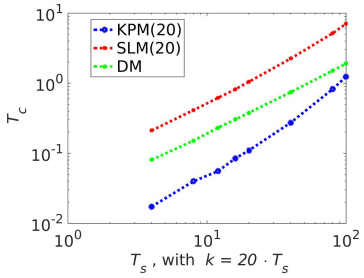
(b) Computation time as a function of k without restart.



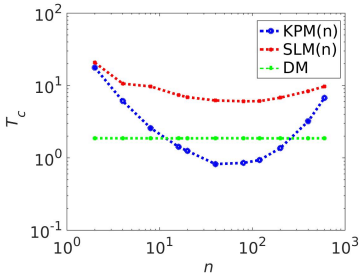
(c) Computation time as a function of n without restart.



(d) Computation time as a function of m with restart.



(e) Computation time as a function of k with restart.



(f) Computation time as a function of n with restart.

Figure 5.7: A figure of the computation times with and without restart. $n = 200$, $T_s = 100$, $k = 2000$, and $m = 20$.

Figure 5.7 shows that when restart is not enabled, both `KPM` and `SLM` are slower than `DM` for $m \geq 100$. `KPM` is faster than `DM` after this, while `SLM` always is slower than `DM`. When restart is enabled, `KPM` can be faster if m is large, and k is small. `SLM` is still slower than `DM`. Figure 5.7f shows that when restart is not enabled, smaller n gives lower computation times.

Chapter 6

Conclusion

The aim with this thesis is was to study two Krylov methods and the restart, and compare them to other popular solution methods in: global error, energy preservation, and computation time. The energy preservation for SLM was also examined, together with derivation of the methods and proof of convergence.

The Krylov methods have been compared to DM (see Section 2.5.5), in error, energy and computation time. Two different Hamiltonian ODEs were used, one was random (See Section 3.1.2), and the other was a discretization of the wave equation (See Section 3.1.1). Two very different cases were also tested: constant energy, and varying energy. These cases were handled differently. In the case with constant energy, the error was measured with an exact solver, see Table 2.2. When the energy was varying, both error and energy was measured against DM. Because of this, the conclusion is divided into two parts, Section 6.1 and 6.2.

The wave equation gave nice results for any n, which made the restart uninteresting. To better show how the restarts can be used to improve the solution, only the random test case was used. If you want to use a Krylov method to solve the wave equation, this is very efficient.

6.1 Constant energy

The restart can be effectively used to improve the error if a small restart variable (see Section 2.5.1) is preferred (see Figure 4.4d).

When restart is not enabled: The error for all methods increase linearly with time (see Figure 4.8a). SLM is energy preserving (see Figure 4.10b). KPMs energy is not preserved, but it is bounded. Using an exact solver instead of the trapezoidal rule gives a smaller error, but does not change the energy (see Figure 4.12). Both SLM and KPM is faster than DM (see Figure 4.16). This makes SLM an excellent solver on linear Hamiltonian ODEs, when restart is not used.

When restart is enabled: SLM and KPM behave very similar. The error increases linearly with time, but diverges on long time domains (see Figure 4.8 and 4.10). The energy also increases slowly with time, and diverges on long time domains. Using windowing, the performance of the methods is improved, see Section 2.4 and Figure 4.11. Using an exact solver instead of the trapezoidal rule gives a smaller error, but does not change the energy (see Figure 4.12). SLM is very slow, making it less useful (see Figure 4.16). KPM, on the other hand, is quite fast, but only if the matrix is big, the number of steps in time is small, and the restart variable is well chosen (see Figure 4.16f). This means that KPM often will be the better alternative of the two methods, when restart is used.

Using windowing and/or an exact solver does not change the computation time, see Figure 4.17 and 4.18.

6.2 Varying energy

For SLM and KPM without restart, the energy and error are bounded, but not constant (see Figure 5.4). When restart is enabled, the energy and error increases slower, but diverges on long time domains (see Figure 5.5). The divergence cannot be removed by windowing (see Figure 5.6). Compared to DM, SLM is slow, while KPM is fast if the size of the matrix is large, the number of time steps is small, and n is well chosen (see Figure 5.7). This makes KPM the better choice if this is

the case, and DM better in the other cases.

6.3 Further work

It could be interesting to see how the error and energy behave with a random Hamiltonian matrix with different structures (eg. full random, sparse 5-diagonal, 1-diagonal), with a known analytical solution (eg. make an accurate eigenvalue/eigenvector decomposition of the matrix and then take the exponential of the diagonal matrix of eigenvalues, or use the method based on the shur decomposition). This could shed more light on when and how to best utilize the Krylov methods.

The Krylov methods ability to create small matrices can make it possible to utilize graphics cards as the processing unit. Graphics cards are designed to be used on small matrices [17]. They consists of thousands of processing units, which could be used to solve non linear Hamiltonian problems in parallel (see Section 2.5.3).

Bibliography

[1] http://se.mathworks.com/help/matlab/ref/expm.html.

[2] Archbishop Richard Bancroft. *King James Bible. New Testament (Matthew 5:18)*. King's Printer Robert Barker, 1604.

[3] Peter Benner and Zvonimir Bujanović. On the solution of large-scale algebraic Riccati equations by using low-dimensional invariant subspaces. *Linear Algebra and its Applications*, 488:430–459, 2016.

[4] Peter Benner and Heike FaBbender. An Implicitly Restarted Symplectic Lanczos Method for the Hamlltonlan Eigenvalue Problem. *LINEAR ALGEBRA AND ITS APPLICATIONS 263:75-111 (1997)*.

[5] Peter Benner and Heike Faßbender. An Implicitly Restarted Symplectic Lanczos Method for the Hamlltonlan Eigenvalue Problem. *LINEAR ALGEBRA AND ITS APPLICATIONS 263:75-111*, 1997.

[6] Peter Benner, Heike Faßbender, and Martin Stoll. A Hamiltonian Krylov–Schur-type method based on the symplectic Lanczos process. *Linear Algebra and its Applications 435 (2011) 578–600*, page 579.

[7] Peter Benner, Heike Faßbender, and Martin Stoll. A Hamiltonian Krylov–Schur-type method based on the symplectic Lanczos process side 581. *Linear Algebra and its Applications 435 (2011) 578–600*, page 581.

[8] Peter Benner, Heike Faßbender, and Martin Stoll. A Hamiltonian Krylov–Schur-type

method based on the symplectic Lanczos process. *Linear Algebra and its Applications*, (435):578–600, 2011.

[9] Peter Benner, Heike Fassbender, and Chao Yang. Some Remarks on the Complex J - Symmetric Eigenproblem. july 2015.

[10] M. A. Botchev. A block Krylov subspace time-exact solution method for linear ordinary differential equation systems. *Numer. Linear Algebra Appl. 2013; 20:557–574*, page 557.

[11] E. Celledoni, V. Grimm, R.I. McLachlan, D.I. McLaren, D. O'Neale, B. Owren, and G.R.W. Quispel. Preserving energy resp. dissipation in numerical PDEs using the ''Average Vector Field'' method. *Journal of Computational Physics 231 (2012) 6770–6789*, page 6772.

[12] E. Celledoni, V. Grimm, R.I. McLachlan, D.I. McLaren, D. O'Neale, B. Owren, and G.R.W. Quispel. Preserving energy resp. dissipation in numerical PDEs using the ''Average Vector Field'' method. *Journal of Computational Physics 231 (2012) 6770–6789*, page 6778.

[13] Celledoni E. and Moret I. A Krylov projection method for system of ODEs. *Applied Numerical Mathematics 23 (1997) 365-378*, 1997.

[14] Celledoni E. and Moret I. A Krylov projection method for system of ODEs. *Applied Numerical Mathematics 23 (1997) 365-378*, page 372, 1997.

[15] Sindre Eskeland. A Krylov projection Method for the heat equation. 2015.

[16] HEIKE FASSBENDER. ERROR ANALYSIS OF THE SYMPLECTIC LANCZOS METHOD FOR THE SYMPLECTIC EIGENVALUE PROBLEM. *BIT 2000, Vol. 40, No. 3, pp. 471–496*.

[17] K. Fatahalian, P. Hanrahan, and J. Sugerman. Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication. *Graphics Hardware*, 2005.

[18] Joel Feldman. Derivation of the Wave Equation. 2000.

[19] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration*, chapter VI.8 Conjugate Symplecticity, page 223. Springer, second edition edition, 2006.

[20] Ernst Hairer, Gerhard Wanner, and Christian Lubich. *Geometric Numerical Integration*, chapter X.3 Linear Error Growth and Near-Preservation of First Integrals, page 413. Springer, second edition edition, 2006.

[21] Cohn Henry, Kleinberg Robert, Szegedy Bal azs, and Umans Christopher. Group-theoretic Algorithms for Matrix Multiplication. *Proceedings of the 46th Annual Symposium on Foundations of Computer Science, 23-25 October 2005, Pittsburgh, PA, IEEE Computer Society, pp. 379-388*, 2005.

[22] Abramowitz Milton and Stegun Irene A. Handbook of Mathematical Functions. Tenth Printing, December 1972. with corrections.

[23] Arup Kumar Nandy and C. S. Jog. Conservation properties of the trapezoidal rule in linear time domain analysis of acoustics and structures. *FRITA Lab,Department of Mechanical Engineering, Indian Institute of Science, Bangalore, India-560012*, januar 2014.

[24] Endre Süli and David F. Mayers. *An Introduction to Numerical Analysis*, chapter 7.2 Newton–Cotes formulae, pages 202–203. Cambridge university press, 2003.

[25] Endre Süli and David F. Mayers. *An Introduction to Numerical Analysis*, chapter 12.2 One-step methods, page 317. Cambridge university press, 2003.

[26] Endre Süli and David F. Mayers. *An Introduction to Numerical Analysis*, chapter Definition 10.1, page 286. Cambridge university press, 2003.

[27] David S. Watkins. On Hamiltonian and symplectic Lanczos processes. *Department of Mathematics, Washington State University, Pullman, WA 99164-3113, USA.*

[28] Saad Yousef. *Iterative Methods for Sparse Linear Systems*, volume SECOND EDITION, chapter 6.3.1, pages 154–155. Siam, 2003. Proposition 6.5.

[29] Saad Yousef. *Iterative Methods for Sparse Linear Systems*, volume SECOND EDITION, chapter 6.3.1, page 154. Siam, 2003. Algorithm 6.1.

[30] Saad Yousef. *Iterative Methods for Sparse Linear Systems*, volume SECOND EDITION, chapter Proposition 6.6., page 155. Siam, 2003.