NTNU
Department of Mathematical Sciences

Krylov methods for linear Hamiltonian ODE's

Sindre Eskeland

February 28, 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 Trygve for helping me with proofs; to Ane for not crashing my computer; and to Harald for distracting me with silly computer games.

Summary and Conclusions

Krylov methods are projection methods that can transform big linear ODE's to a smaller linear ODE's with similar properties. Two such methods (the symplectic Lanczos method (SLM) and the Krylov projection method (KPM)) are compared with each other on linear Hamiltonian differential equations. The behavior of global error, energy as a function of time is examined. Computation time for the different methods are also shown. The projection methods are also compared to each other on non autonomous linear Hamiltonian differential equation. Energy preservation for the symplectic Lanczos method is proved under some assumptions, along with convergence for both projection methods. SLM has no advantage over KPM when considering error. On autonomous systems the computation times for SLM and KPM are similar, and both methods can be well utilized. On non autonomous systems KPM is faster than SLM. Since it has comparable error and energy preservation KPM might be considered a better choice in this case.

Contents

Chapter 1

Introduction

Hamiltonian differential equations is 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 projection 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 [7]

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

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 matrix, 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], [27] and [5]. Since SLM preserves the Hamiltonian structure of the matrix, and the projected problem has the same energy 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 case where A is Hamiltonian. The Krylov methods can utilize restarts to improve the solutions via iterative refinement. This text will look at these two projection 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 and error behaves in this case will be examined.

In addition to the case with Hamiltonian linear ODE's 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 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 projection 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. 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 $u(t_j)$ will be 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. Let the discretization of $f(t_j)$ be called F_j. The integration methods considered in this text are trapezoidal rule, forward Euler, and 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 a 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 in two integration methods: forward Euler, and backwards Euler. By first performing a

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 than for the other methods. 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: Methods for exact integration in time. Since they are very computationally demanding they will only be used on small projected matrices. They 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. These build in function are explained in MATLAB's docmentation: [1].

Eigenvalue and diagonalization (`diag`)	$[V,D] = \texttt{eig}(A)$ $U_i = V\cdot\texttt{diag}\left(\texttt{exp}\left(\texttt{diag}(D\cdot t_i)\right)\right)/V\cdot b - b$
MATLAB's `expm` function (`expm`)	$U_i = \texttt{expm}(A\cdot t_i)\cdot b - b$

backward Euler 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, some exact solvers are used for comparison. They are 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 projection 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

There are two Krylov methods that will be discussed 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 a restart approach. This is done using the projection 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 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, $e_{\hat{m}}$ is the $\hat{m}$-th canonical vector in $\mathbb{R}^{\hat{m}}$, and n is the number of iterations performed with Arnoldi's algorithm.

We conduct the approximations

$$u_n(t) = V_n z_n(t)$$

for the solution of (2.5) by requiring that

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

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}, $h_{n+1,n}$.

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 equation (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 $n \leq \hat{m}$ ($V_{\hat{m}} h_{\hat{m}+1,\hat{m}} v_{\hat{m}+1} = 0$ by construction [30]), the procedure will converge toward 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 (2.8) from (2.5)

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 iterations of KPM to approximated numerically $\epsilon_n(t)$ 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 ODE's:

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

The numerical solution after i restarts with restart 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 an arbitrary constant of the true solution. 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.

The proof of convergence for the restart can be found in [14].

2.5.2 Symplectic Lanczos method

SLM requires the matrix A in (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 [9], with reortogonalization from [6].

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**
  $v = Av_j$
  $\delta_j = v_j^\top v$
  $\tilde{w} = v - \delta_j v_j$
  $\kappa_j = v_j^\top J_{\tilde{m}} v$
  $w_j = \frac{1}{\kappa_j} \tilde{w}_j$
  $w = Aw^j$
  $\tilde{S}_{j-1} = [v_1, v_2, \ldots, v_{j-1}, w_1, w_2, \ldots, w_{j-1}]$
  $w_j = w_j + \tilde{S}_{j-1} J_{j-1} \tilde{S}_{j-1}^\top J_{\tilde{m}} w_j$
  $\beta = -w_j^\top J_{\tilde{m}} w$
  $\tilde{v}_{j+1} = w - \zeta_j v_{j-1} - \beta_j v_j + \delta_j v_j$
  $\zeta_{j+1} = \|\tilde{v}_{j+1}\|_2$
  $v_{j+1} = \frac{1}{\zeta_{j+1}} \tilde{v}_{j+1}$
  $\tilde{S}_j = [v_1, v_2, \ldots, v_j, w_1, w_2, \ldots, w_j]$
  $v_{j+1} = v_{j+1} + \tilde{S}_j J_j \tilde{S}_j^\top J_{\tilde{m}} v_{j+1}$
end for
$S_n = [v_1, v_2, \ldots, v_{\tilde{n}}, w_1, w_2, \ldots, w_{\tilde{n}}]$
$H_n = \begin{bmatrix} \mathrm{diag}\left([\delta_j]_{j=1}^{\tilde{n}}\right) & \mathrm{tridiag}\left([\zeta_j]_{j=2}^{\tilde{n}}, [\beta_j]_{j=1}^{\tilde{n}}, [\zeta_j]_{j=2}^{\tilde{n}}\right) \\ \mathrm{diag}\left([\kappa_j]_{j=1}^{\tilde{n}}\right) & \mathrm{diag}\left([-\delta_j]_{j=1}^{\tilde{n}}\right) \end{bmatrix}$
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 helps ensure symplecticity of the method.

To derive the method, transform the problem in equation (2.5) with $u_n(t) = S_n z_n(t)$, by requiring that

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

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

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

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 $n \leq \hat{m}$ [8], the method converges.

A restart can be performed if a small, fixed n is needed. This 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 solution with restart 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 of convergence and other interesting results for this method can be found in [2].

Proof that `SLM` without restart is energy preserving

This section will show that if equation (2.12) is solved by an energy preserving method, eg. trapezoidal rule, the energy of $u_n(t)$ will be preserved [12]. It is well known that the Hamiltonian ODE (1.1), solved with an energy preserving method has constant energy. Since `SLM`'s projected matrix 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 $u_n(t)$ also is energy preserving.

The energy of equation (2.12) is

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

While the energy of the original problem is

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

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

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

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

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

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 the method itself is not energy preserving. The other reason is that the transformation with V_n is not symplectic, thus the transformation will change the energy. How this holds in practice is shown in Section 4.4.4.

No proof known to me has predicted anything about the energy preserving property of `SLM` with restart.

2.5.3 Linearity of the methods

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

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

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

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

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 suffer. It is not advised to use any Krylov method if this is the case [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. Since initial conditions cannot be time dependent, they must be shifted to act as a source term.

2.5.5 Direct method

Until now, the methods presented are special in some way, because they do not solve the original problem, but some transformed problem, and therefore behaves differently. This makes it natural to compare them to more usual solution approaches. The method used for this is, in this

text, 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.

The energy of DM will be constant with T_s if an energy preserving method is used, while the error will increase linearly with T_s [20] if equation (1.1) is solved. A goal of this text is to see how the Krylov method's error and energy behaves compared to DM.

The proof that DM is the natural method to compare with will be shown here with trapezoidal rule for KPM on equation (1.1).

We start by discretizing equation (2.7) with the trapezoidal rule:

$$\Big(I - \frac{H_n h}{2}\Big) Z_{i+1} = Z_i + \frac{h}{2}\big(H_n Z_i + \|b\|_2 e_1\big).$$

Here Z_i is the time discretized version of $z_n(t_i)$. By using equation (2.6) and the transformation $\tilde{U}_i = V_n Z_i$, this can be written as

$$\Big(I - \frac{Ah}{2}\Big) \tilde{U}_{i+1} = \tilde{U}_i + \frac{h}{2}\big(A\tilde{U}_i + b\big).$$

Equation (1.1) discretized with trapezoidal rule is given by

$$\Big(I - \frac{Ah}{2}\Big) U_{i+1} = U_i + \frac{h}{2}\big(AU_i + b\big).$$

It is shown in Section 2.5 that $u_n(t)$ will converge towards $u(t)$ when n increases. Since this also holds for $\tilde{U}_i$ and U_i, KPM will converge towards DM as n increases.

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 2.3: 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 costs 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)	$\mathcal{O}(n\hat{m})$

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	Explonation and cost
`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. $\mathcal{O}((n^2\hat{m}+kn^3+\hat{m}nk)i_rK)$

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. This makes it impossible to conclude anything without actually doing it, results are shown in Section 4.6 and 5.5.

2.6 `SLM` and its eigenvalue solving properties

A quick internet search for "symplectic Lanczos method" gives several articles about the symplectic Lanczos method's ability to approximate the eigenvalues of Hamiltonian matrices, eg. [4],[27], and [16]. This section will explain why this algorithm so attractive for these types of problems.

The eigenvalue problem 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 is that the biggest eigenvalue is found first.

The method is not without flaws. Frequent breakdowns due to ill conditioned matrices occur. But due to its large potential, much work has been done to overcome the difficulties. The most promising improvements are different types of restarts. 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.

Many of these papers also compare SLM and KPM.

Chapter 3

Practice

This section will be concerned with test problems, implementation, and calculation of out data.

3.1 Test problems

This section will present the test problems used to obtain 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 satisfies 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 pieces, each with 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 $i, j = 1, \ldots, m-1$. The wave equation can then be written as a system of ODE's:

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

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.

Two test problems are used to check the integrity of the solver. In the case with constant energy, the test problem is

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

For varying energy it is

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

Both q and w are used when measuring the error.

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`. The part $5I_{\hat{m}}$ is added to make $J_{\hat{m}}A$ diagonally dominant, since a fully random problem will not converge in general. The matrix is simulated as a 2 dimensional system, to make implementations easier. Since the projection methods are only usable with sparse matrices, it is J-tridiagonal.

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 unknown, therefore it is impossible to show convergence in the traditional sense. A larger m does not give a better approximation of some equation, it gives a new matrix, with no relation to any matrix with different m. This test problem might therefore seam uninteresting, but there 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 error will be measured as the difference between the solution obtained by a Krylov method and `DM`, it will be marked error$^{\text{comp}}$. This will make the error seam a lot smaller than it really is, but as shown in Section 2.5.5, it is correct to compare the solutions in this manner. When the energy is constant there is no need to compare it to anything, but test problems with varying energy will be compared with its non-projected equivalent, and be marked with energy$^{\text{comp}}$.

3.2 Outdata

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 correct solution. The resulting vector of absolute errors is then divided by the correct solution, so that the error is relative to the correct solution. This will be marked with "error" on the plots. In the case where the correct solution is unknown, as in `semirandom`, the error is measured as the difference between the projected solution and `DM`. This case is marked with "error$^{\text{comp}}$". In some results error$^{\text{comp}}$ will also be used with `wave`.

The energy is found with one dedicated energy function. The exceptions are $\mathscr{H}_3$ and $\mathscr{H}_4$ which are calculated by a different function. The energy is calculated for each time step, and then initial energy is subtracted from this. In the case where the energy is constant, the energy is calculated without comparing it to anything. This is because the correct solution would sometimes have more energy than the approximation, making it difficult to draw any conclusions. This is done for all energies in Chapter 4. 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 DM and a projection method is compared.

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 in the computation time. The symbol for computation time is T_c.

3.3 Figure

All figure 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 legend 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 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[3], and both ϵ and δ were unavailable.

All code 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

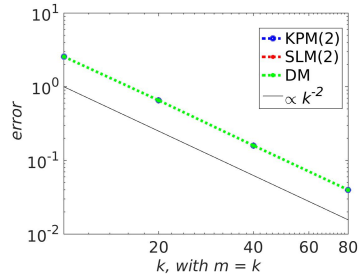
This chapter will show how error, energy and computation time changes with i_r, ι, T_s, m and n. There are special interest in seeing how the energy for `SLM` behaves, how `KPM` and `SLM` differs, and how occurring problems might be handled. The predictions and proofs made in the theory chapter will also be tested.

Suitable values for the matrices and times are $m = 20$, and $k = 20$ per second. These are chosen so since they show interesting results, without being too computationally demanding.

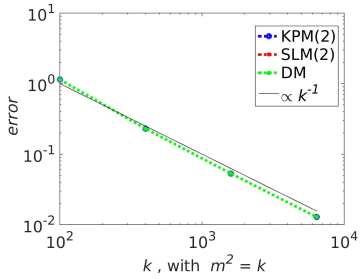
4.1 Convergence

This section will show convergence with the wave equation with all the integration methods presented in Table 2.1 and 2.2. The reason for only using the wave equation is that the analytical solution is unknown for `semirandom`. $T_s = 1$ second for all plots in this section.

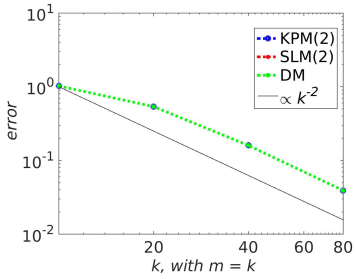
4.1.1 With Numerical integrations



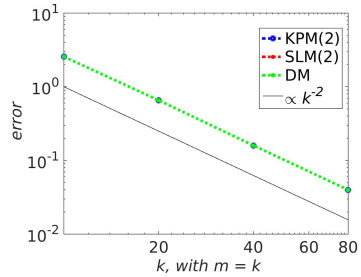
(a) Convergence of trapezoidal rule without restart.



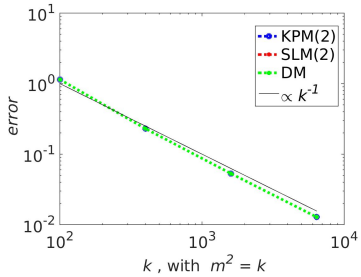
(b) Convergence of forward Euler without restart.



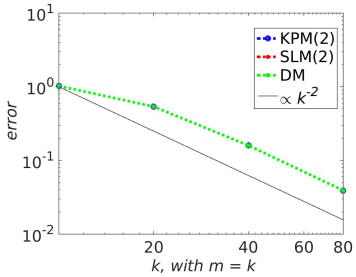
(c) Convergence of midpoint rule without restart.



(d) Convergence of trapezoidal rule with restart.



(e) Convergence of forward Euler with restart.



(f) Convergence of midpoint rule with restart.

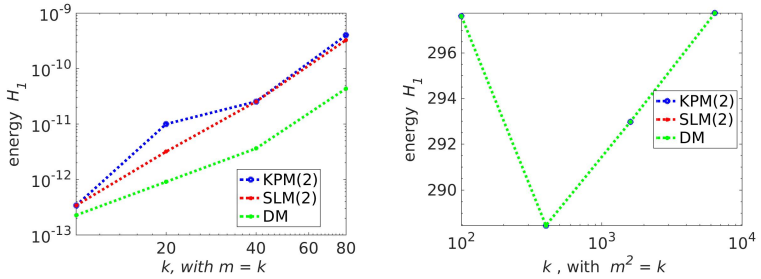
Figure 4.1: Convergence for the different integration methods. Notice that forward Euler uses k^2 points in time where the other methods uses k points in time to obtain the same accuracy. $n = 2$ is kept constant for all plots, since smaller n generally gives poorer convergence. For the figures with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

All methods converge with the expected rate.

Forward Euler has a worse run time than the other methods, due to the larger k needed for the same convergence. It also has a tendency to diverge on longer time intervals and is in general not suited for the job. Figure 4.2 shows the difference in energy between forward Euler (Figure 4.2a) and trapezoidal rule (Figure 4.2b). This difference in energy is reason enough not to use forward Euler any more.

Trapezoidal rule and midpoint rule converges quadratically and near identically, as they should. Since midpoint rule is symplectic, it will be used when restart is enabled. Trapezoidal rule has a faster run time, and will be used when restart is not enabled due to insignificant differences in

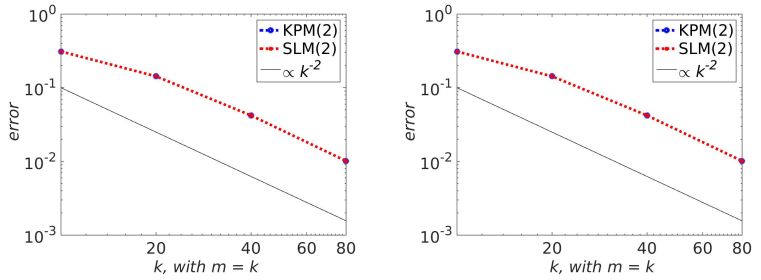
error.



(a) Energy for trapezoidal rule. (b) Energy 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 With exact solvers



(a) Convergence of the exact solver using `diag` without restart.

(b) Convergence of the exact solver using `expm` without restart.

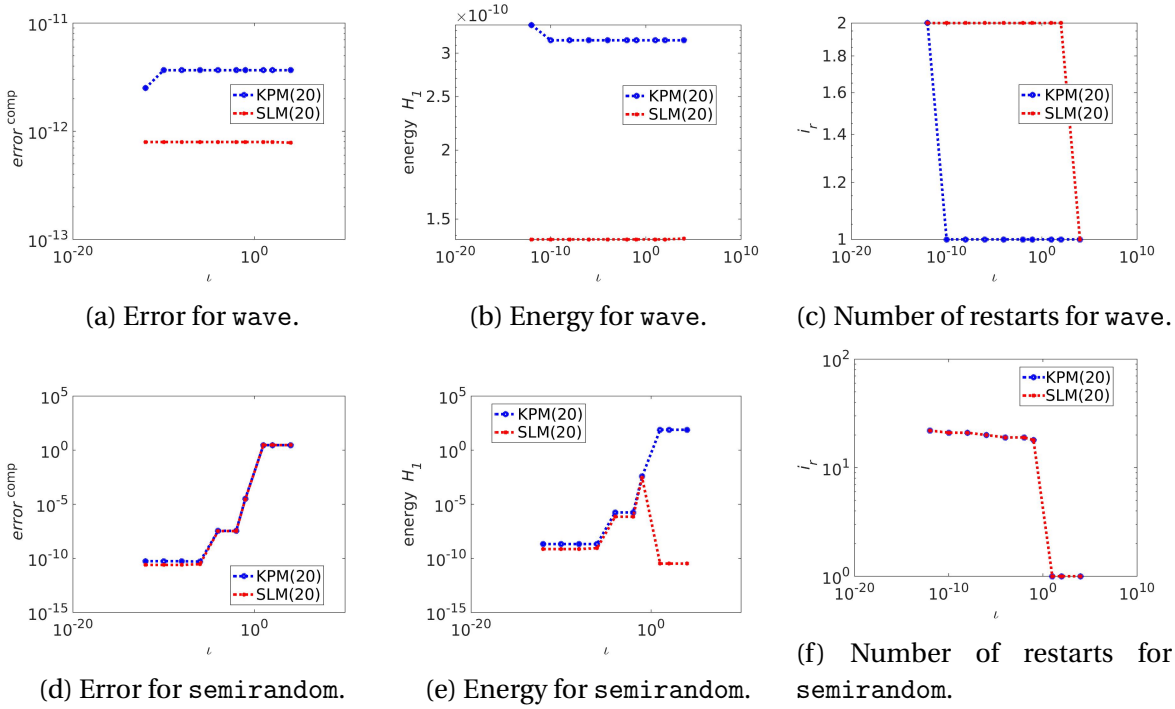
Figure 4.3: Convergence for exact solvers, with trapezoidal rule. Note that the exactness of the methods are for the integration in time, and not the spacial discretization. The expected convergence is therefore still quadratic with m. $n = 2$ because taking the matrix exponential is a costly operation and it is harder for the methods to converge when n is small. 1 second is simulated.

The convergence is quadratic and identical for both integration methods. It might seem strange to include two different exact solvers which look so similar, but an important difference between the two is shown in Section 4.5.1.

4.2 Convergence with the restart

This section will show how error and energy changes with ι and i_r.

4.2.1 As a function of ι



(a) Error for `wave`.

(b) Energy for `wave`.

(c) Number of restarts for `wave`.

(d) Error for `semirandom`.

(e) Energy for `semirandom`.

(f) Number of restarts for `semirandom`.

Figure 4.4: These figures show how choosing different ι affects the solution. The pictures on the top are for `wave` and the pictures at the bottom are for `semirandom`. This plot considers 100 seconds, with $k = 2000$, $n = m = 20$ and midpoint rule.

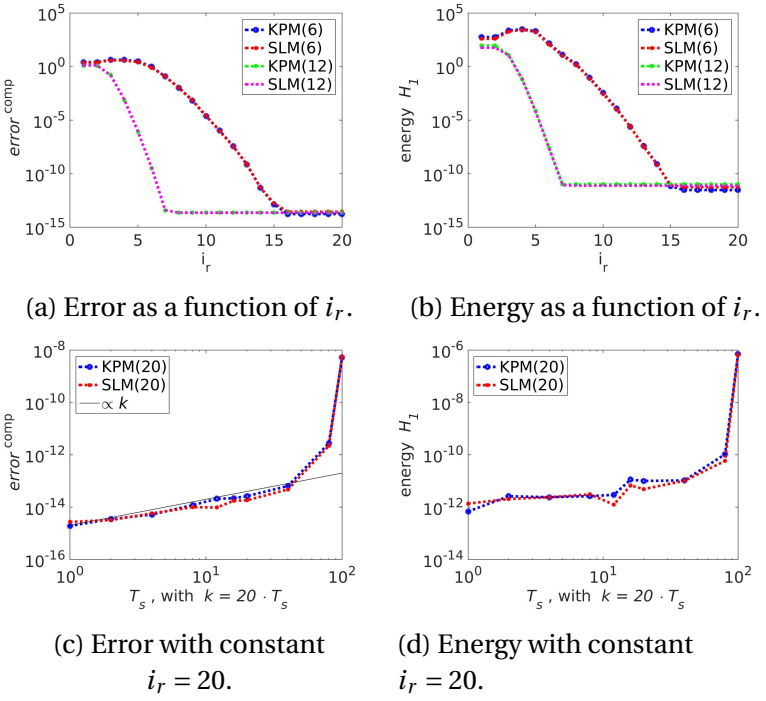
Figure 4.4 shows that if `wave` is used there is no reason to use the restart. From Figure 4.1 it also seams be the case for larger m. `wave` will therefore not be used again until Section 4.5. If you are only interested in using a projection method on the wave equation with constant energy this is definitely a smart thing. The largest orthogonal space needed is about $n = 2$ (depending on T_s), which gives an incredible run time.

The rest of this chapter will only consider `semirandom` since this is a more interesting case.

The number of restarts are the same for `KPM` and `SLM`. Another ting to notice is that there is little reason to restart after gaining a certain precision, since the changes will not be visible. ι should be chosen a few orders smaller than the accuracy of `DM`. Based on the pictures $\iota = 1e-6$ is suitable.

Some interesting results from 4.4e and 4.4d is that both KPM and SLM needs to restart to gain a low error, but 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 (necessarily) the error. Thus it is clear that SLM looses its energy preserving property when it restarts. For KPM it is much simpler, more restart means better approximations, for both error and energy.

4.2.2 As a function of i_r



(a) Error as a function of i_r.

(b) Energy as a function of i_r.

(c) Error with constant $i_r = 20$.

(d) Energy with constant $i_r = 20$.

Figure 4.5: The top figures shows how the error and energy changes as a function of i_r. $T_s = 10$ is chosen to better show the decrease. The bottom pictures show how error and energy behaves with increasing time domain, with constant $i_r = 20$. $m = 20$.

Figure 4.5 shows that a few restarts are needed before a better approximation is made. For KPM(6) and SLM(6) it is necessary to perform more than 5 restarts to gain any accuracy. After 17 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 projection methods can perform the exact number of iteration needed to converge, with any n and T_s that allows convergence. If i_r is used as convergence criterion, it needs to be changed with n and T_s. The increase in error is linear (except for

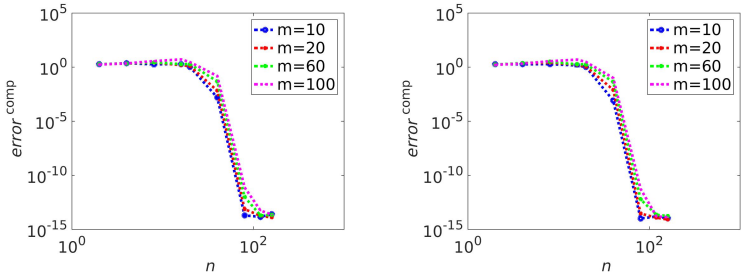
the last few points) for Figure 4.5c, which is what is predicted for these methods.

The increase in energy is sublinear for Figure 4.5d (except for the last few points). It was predicted that it would be constant for SLM. Why the increase happens will be examined later in this chapter, together with why the increase at the last few points happens.

4.3 How to choose n

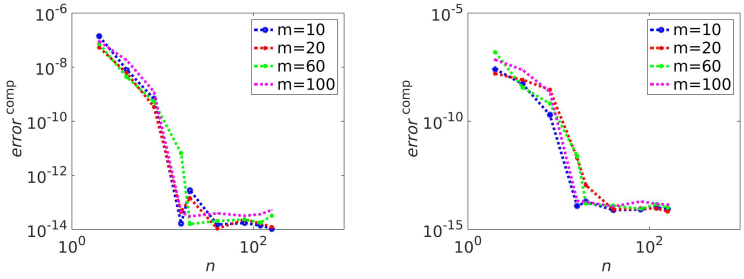
This section will look at how to choose n to avoid unnecessary restart and still obtain satisfactory results. Last section show that error and energy behaved quite similarly. Thus this section will only show error, since this is the most important property of the restart. We assume that all cases tested here are independent.

4.3.1 With different 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.

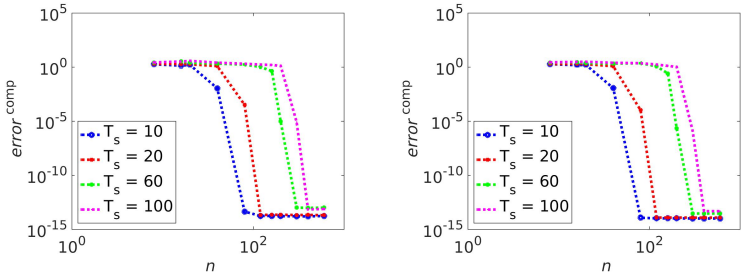
Figure 4.6: The pictures shows which n gives convergence for different m. Restart is not enabled for the top pictures, restart is enabled for the bottom pictures. $k = 200$ over 10 seconds.

Figure 4.6a and 4.6b shows that n can be choose independent of m, as long as $n \geq 100$ when restart is not enabled.

Figure 4.6c and 4.6d shows that if restart is enabled, a good approximation of the solution can be found with $n \geq 20$, for any m.

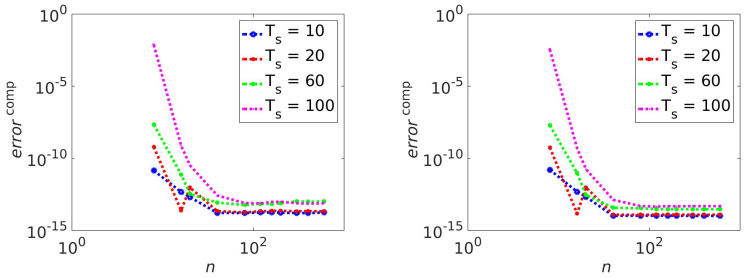
The reason for the independence between m and n can be due to the structure of the matrix, and is not a rule for general Hamiltonian matrices.

4.3.2 With different 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 shows which n gives convergence for different T_s, with $k = 20 \cdot T_s$ and $m = 20$. The top plots are with restart, and the bottom are without restart.

Figure 4.7 shows that the length of the time domain is affecting the optimal choice for n. It seams that if T_s is doubled, then n needs to be doubled, though this rule seams to be less important when restart is enabled.

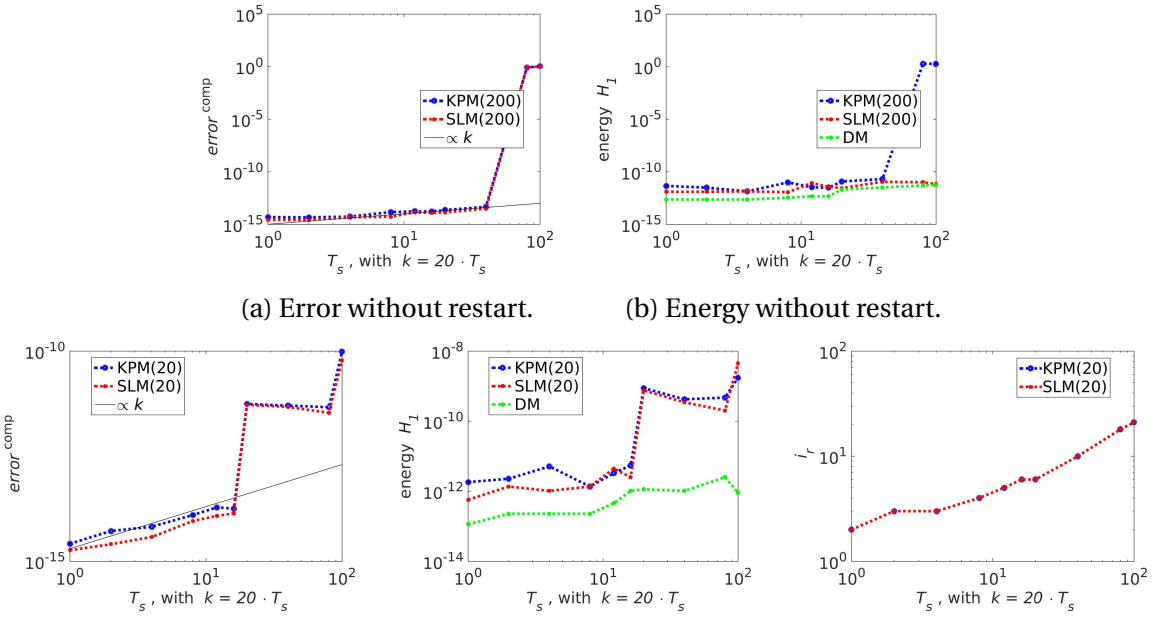
SLM and KPM behaves very similarly in all pictures shown in this section.

4.4 Energy and error

The convergence section shows that increasing m and k will decrease the error of the solution. This has been done for short time (1 s). Convergence for the restart has been shown both for different ι and i_r. There has also been a discussion about how to chose n. The values of these will be kept at $n = 200$ when restart is not used. $n = 20$ and $\iota = 1e-6$ when restart is used. Remember that $m = 20$ except where stated.

This section will look at how error and energy change as a function of time, and how restarting and windowing interacts with this.

4.4.1 Energy and error as a function of time.



(a) Error without restart. (b) Energy without restart.

(c) Error with restart. (d) Energy with restart. (e) Number of restarts.

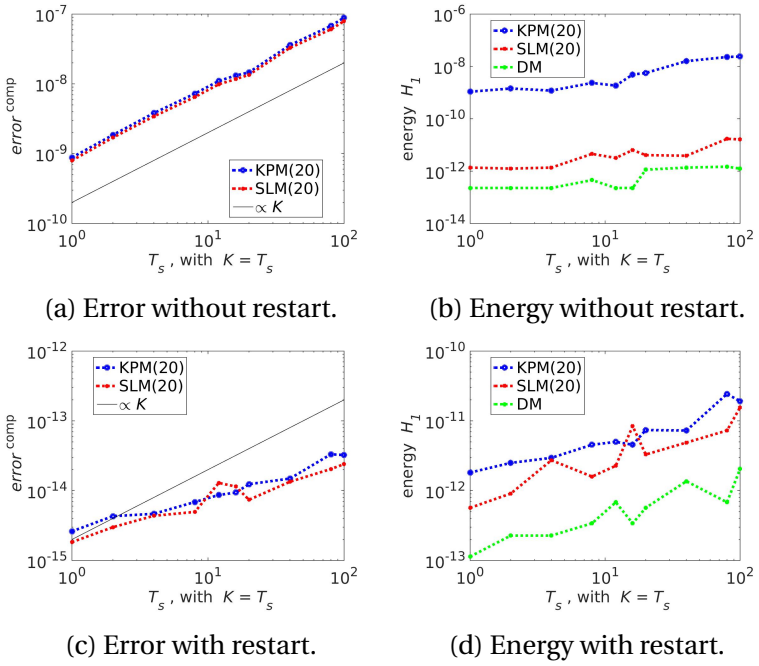
Figure 4.8: The figures show how error and energy changes with as a function of time. $m = 20$. The top pictures are without restart, and the bottom pictures are with restart.

The energy for all methods increase, but slower than linear, and might be explained by small rounding errors that cannot be avoided when dealing with such small numbers. This is supported by the fact that the energy for DM increases equally fast as for the other methods. Picture 4.8b shows that SLM preserves the energy, while KPM does not.
The error increases linearly until a sudden jump at the last points. This shows the instability of the projection methods. If the time domain was a little larger the projection methods would diverge.

The top pictures and bottom pictures in Figure 4.8 are quite similar, except for the last points. This means that restarting can help with convergence.

The number of restarts increases linearly after the time domain has become sufficiently large. Larger time domains might therefor give a larger growth in computation time than expected.

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



(a) Error without restart.

(b) Energy without restart.

(c) Error with restart.

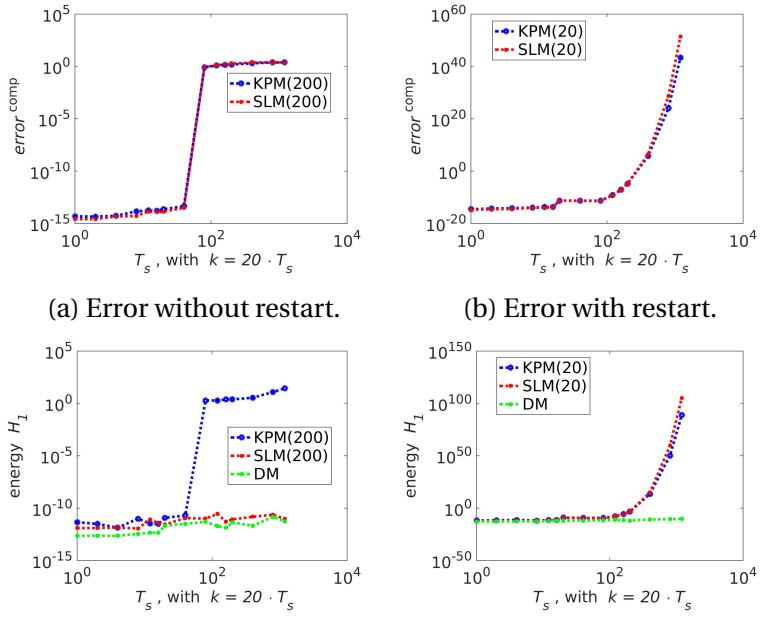
(d) Energy with restart.

Figure 4.9: The figures shows how the error and energy changes over time. $m = 20$, $k = 20$. The top pictures are with restart, and the bottom pictures are without restart.

On Figure 4.9, the divergence problems with the last points seams to be gone. Restarting makes the error a few orders smaller. The growth in error is linear with T_s when restart is not enabled, and sublinear when restart is enabled. The linear increase is a little worrying since it is in comparison to another method that should also have linearly increasing error, this means that windowing without restart can have quadratically increasing error. This will unfortunately not be tested due to the limitations of the test problems. The energy is increasing equally fast for all methods. Both error and energy with windowing is comparable to the other cases. This makes windowing an interesting idea.

4.4.3 Behavior on long time domains

In this section it is shown what will happen when the time domain becomes to large for the projection methods to converge.



(a) Error without restart.

(b) Error with restart.

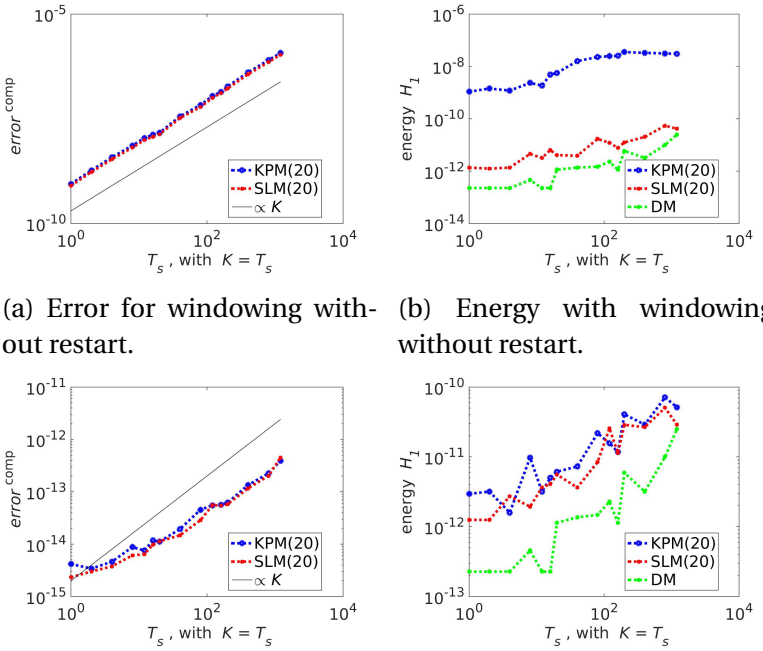
(c) Energy without restart.

(d) Energy with restart.

Figure 4.10: This figure shows how different methods perform on a very long time domain. $m = 20$, $k = 20$ unless stated.

This figure shows something very interesting, namely that with restart or not, the error will diverge on long time domains if n is kept constant. For `SLM` without restart the energy is always preserved (no matter what n, m, T_s and k is used), in all other cases `KPM` and `SLM` perform equally

bad. There is an important difference between restarting and not. If restart is not enabled, the method will at one point not work, and the error will just be noise. But the numeric value of the approximated solution will be close to the numeric value of DM. With restart, on the other hand, everything blows up exponentially. Figure 4.11 shows that if windowing is used the problem can be worked around. In this case the trend of increasing energy and error continues as it did for 100 seconds.



(a) Error for windowing without restart.

(b) Energy with windowing without restart.

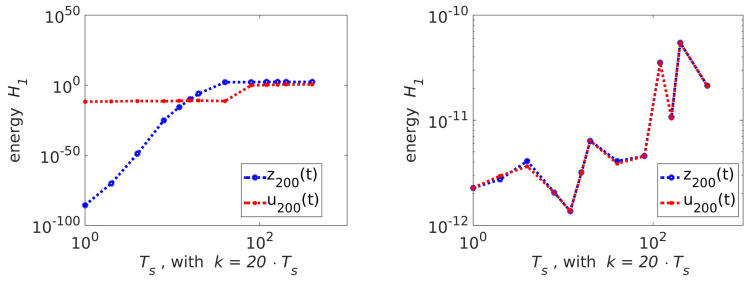
(c) Error for windowing with restart.

(d) Energy with windowing with restart.

Figure 4.11: This figure shows how windowing perform on a very long time domain. $m = 20$, $k = 20$ unless stated.

4.4.4 Energy in the transformations

This section will show how the energy changes when transforming from $z_n(t)$ to $u_n(t)$.



(a) Energy for KPM(200). (b) Energy for SLM(200).

Figure 4.12: The figures shows how the transformation between $z(t)$ and $u(t)$ changes the energy. Restart is not enabled and $m = 20$.

For KPM there is a huge discrepancy between the energy of $z_n(t)$ and $u_n(t)$. For SLM there is no difference between the two. This shows the symplectic transformation of SLM. 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 $n = 200$.

4.4.5 Residual energy

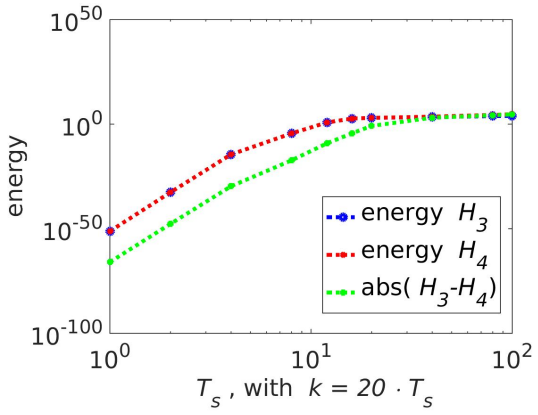


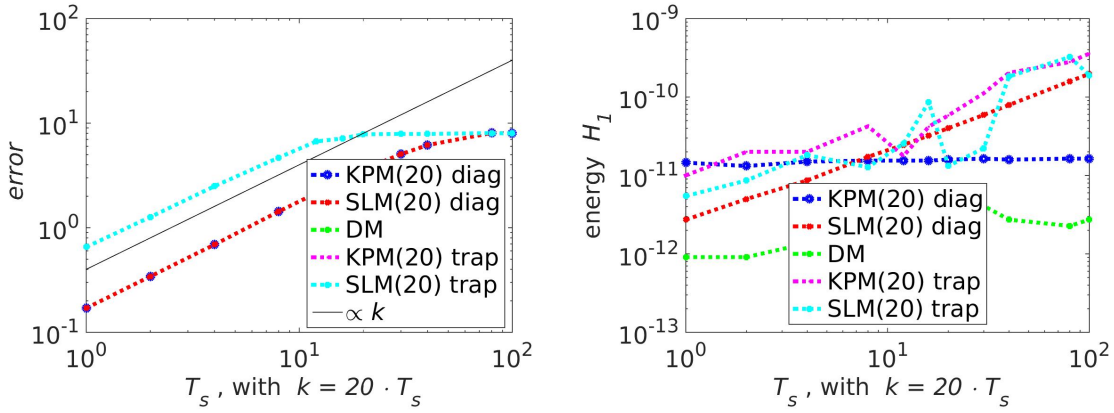
Figure 4.13: A plot of $\mathscr{H}_3$ and $\mathscr{H}_4$ for different time domains.

$\mathscr{H}_3$ and $\mathscr{H}_4$ are very similar, as is predicted in the theory section. The difference between the two is machine accuracy until T_s gets to big (around 10 seconds).

4.5 Energy and error with exact solvers

This section will see how using an exact solver changes error and energy, compared to trapezoidal rule.

Unfortunately the test problems has some severe limitations. `wave` does not require a restart to be well approximated, and the analytical solution is unknown for `semirandom`. If `semirandom` is used there will be no way of knowing if `DM` or the projection method gives the best approximation. If `wave` is used the question about whether to use restart or not with exact solvers will remain unanswered. Since the latter limitation is smaller, `wave` will be used. Restart will not be used since it is already known that the method will not restart.



(a) Error.

(b) Energy.

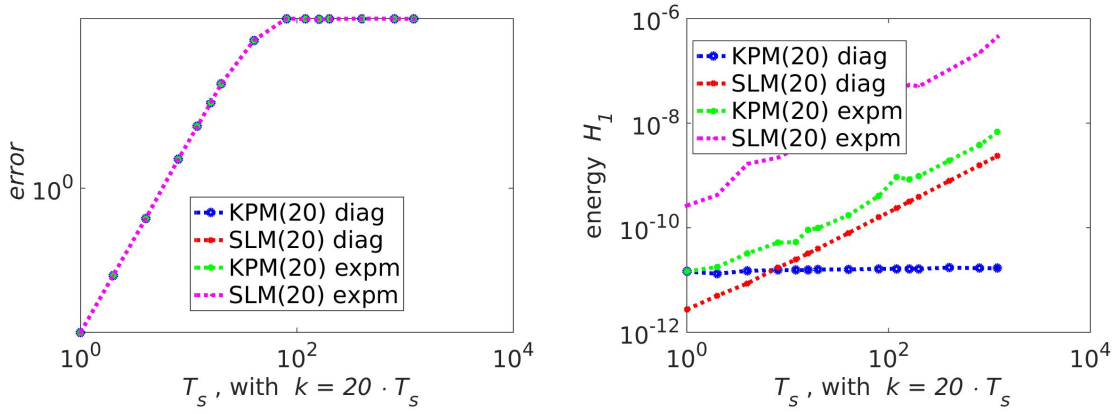
Figure 4.14: A figure showing how the error and energy changes over time when an exact solver (`diag`) is used. $n = 20$, $m = 20$, restart is not enabled. `DM` uses trapezoidal rule.

The error for the projection methods with an exact solver is smaller than the error for the methods that uses trapezoidal rule, but all errors are increasing identically and linearly.
The energy for `SLM` with `diag` increases linearly, why this happens is unknown, but a reasonable explanation is rounding errors. For both `KPM` and `DM` with `diag`, the energy is constant.

$n = 20$ is not large enough to have convergence when `semirandom` is used, this section may therefore give a artificially good result for the exact solver.

4.5.1 MATLABs `expm` function

This section will explain the reason for using `diag` instead of `expm`.



(a) Error.

(b) Energy.

Figure 4.15: A figure showing the difference in error and energy for the different exact integration methods. Restart is not enabled, $m = 20$.

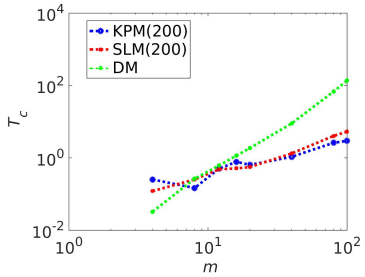
The error for the two exact solver are identical, but there is a big difference in energy. `KPM` has a constant energy with `diag`, while the energy increases linearly when `expm` is used.
It seams that the energy preserving property of `SLM` is lost when used with an exact solver, because of the linear increase in energy. `diag` should still be used since it has a better initial approximation than `expm`.

4.6 Computation time

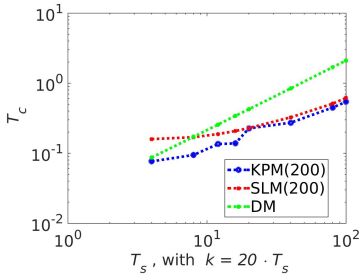
This section will compare computation time for the different methods discussed.

4.6.1 Naïve implementation

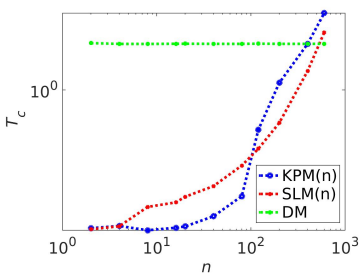
In this case trapezoidal and midpoint rule are used, without windowing. Computation time for different m, k and n is shown.



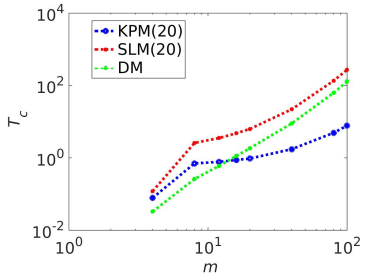
(a) Computation time as a function of m without restart.



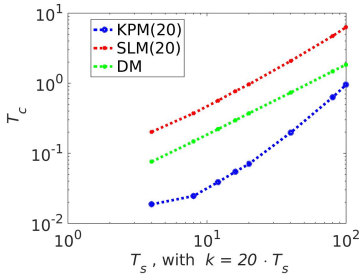
(b) Computation time as a function of k without restart.



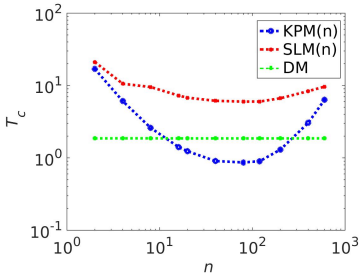
(c) Computation time as a function of k without restart.



(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.

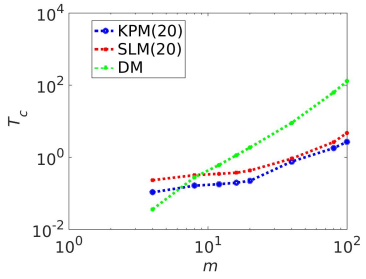
Figure 4.16: A figure of computation times with and without restart for different m, k and n. $n = 200$, $T_s = 100$, $k = 2000$, $m = 20$ unless stated.

Figure 4.16 shows that the computation time for all methods, except `KPM` with restart, increases linearly with k. The reason for the faster increase for `KPM` is the additional restarts needed for convergence on longer time domains.

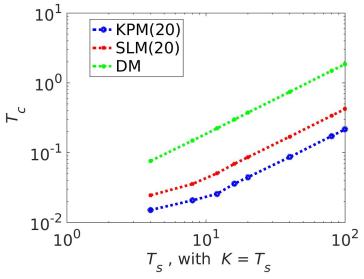
The computation time for `DM` increases quadratically with m. `SLM` is faster than `DM` if restart is not used, but slower if restart is used. The difference between `KPM` with and without restart is minimal, and similar to `SLM` without restart.

Both `KPM` and `SLM` are fastest without restart. `KPM` can also be fast with restart if m is large, T_s is small, and n is well chosen.

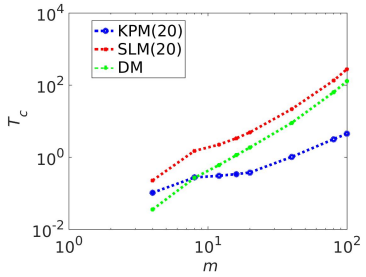
4.6.2 Windowing



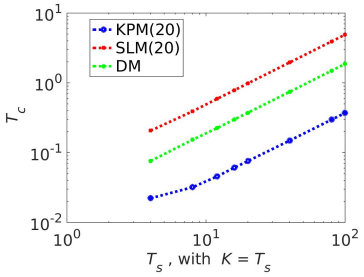
(a) Computation time as a function of m without restart.



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



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



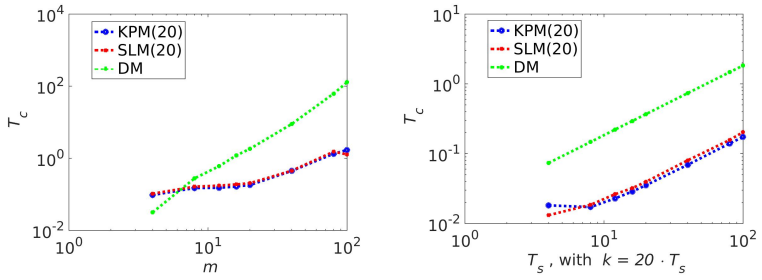
(d) Computation time as a function of k with restart.

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

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. For `KPM` the difference between restarting and not is minimal, and it is always faster than `DM`.

Even though the results here are great for windowing without restart, remember that the error and energy is a few orders bigger than the methods without windowing.

4.6.3 With `diag`



(a) Computation time as a function of m.

(b) Computation time as a function of k.

Figure 4.18: A figure of the computation times with `diag`. $n = 20$, $T_s = 100$, $k = 2000$, $m = 20$ unless stated.

Figure 4.18 shows that the exact solver without restart is the fastest method. This, together with the small error, makes exact solver very desirable. But this result is not showing the entire truth, as it is done with $n = 20$ and without restart. This will not give convergence with `semirandom`. The reason for the small n is that this is what is used in section 4.5.

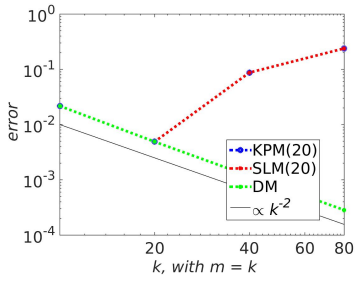
Chapter 5

Results for test problems with varying energy

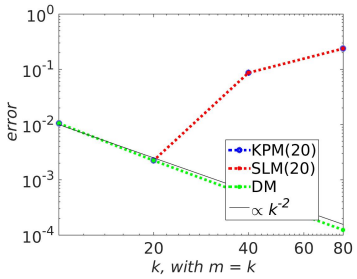
This chapter will look at many of the same elements that was discussed in Chapter 4, but not all. The excluded results are mostly theoretical results that needed verification, which has already been done in Chapter 4. This chapter will try to find out if there is any reason to use `SLM` instead of `KPM` on non autonomous Hamiltonian systems.

The exact solvers does not work due to the time dependent source term in equation (1.2), thus there will be no discussion about that in this chapter.

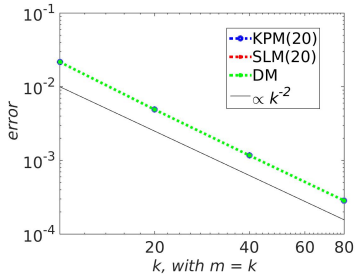
5.1 Convergence with m and k



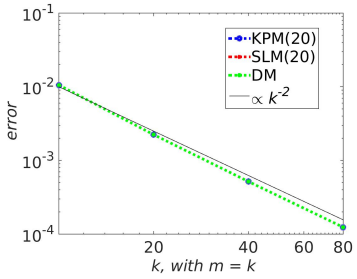
(a) Convergence for `wave` with trapezoidal rule without restart.



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



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

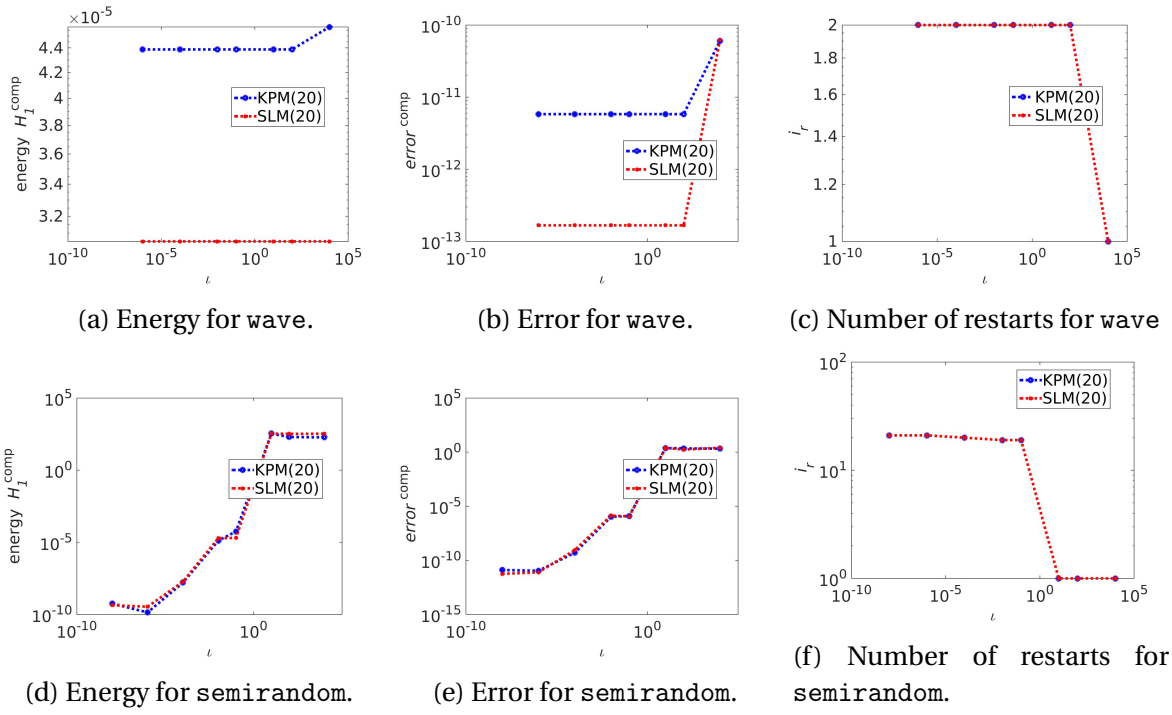


(d) Convergence for `wave` with midpoint rule rule and restart.

Figure 5.1: Convergence plot with different integrators, simulated over 1 second.

Midpoint rule performs better than trapezoidal rule. It is interesting to see that without restart the methods does not converge with $n = 20$, while they converged with $n = 2$ when the energy was constant. Clearly convergence is a lot harder to achieve in this case. Only midpoint rule will be used further in this chapter.

5.2 Convergence with ι



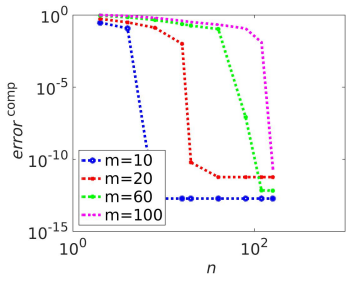
(a) Energy for `wave`. (b) Error for `wave`. (c) Number of restarts for `wave`

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

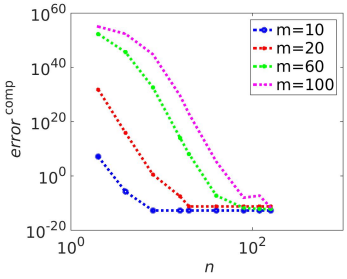
Figure 5.2: These figures show how restarting can improve the solution. The pictures on the top are for `wave` and on the bottom are for `semirandom`. This plot considers 100 seconds, with $n = m = 20$, $k = 2000$, $m = 20$, and midpoint rule.

These figures look very similar to the figures in Section 4.4, one important difference is that the energy for `SLM` no longer starts at $1e-15$.

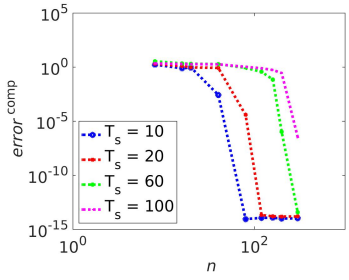
5.3 How to choose n



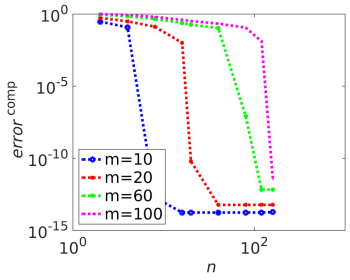
(a) Error for KPM without restart.



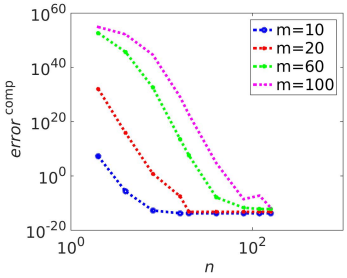
(b) Error for KPM with restart.



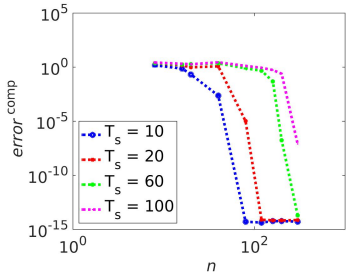
(c) Error for KPM without restart.



(d) Error for SLM without restart.



(e) Error for SLM with restart.



(f) Error for SLM without restart.

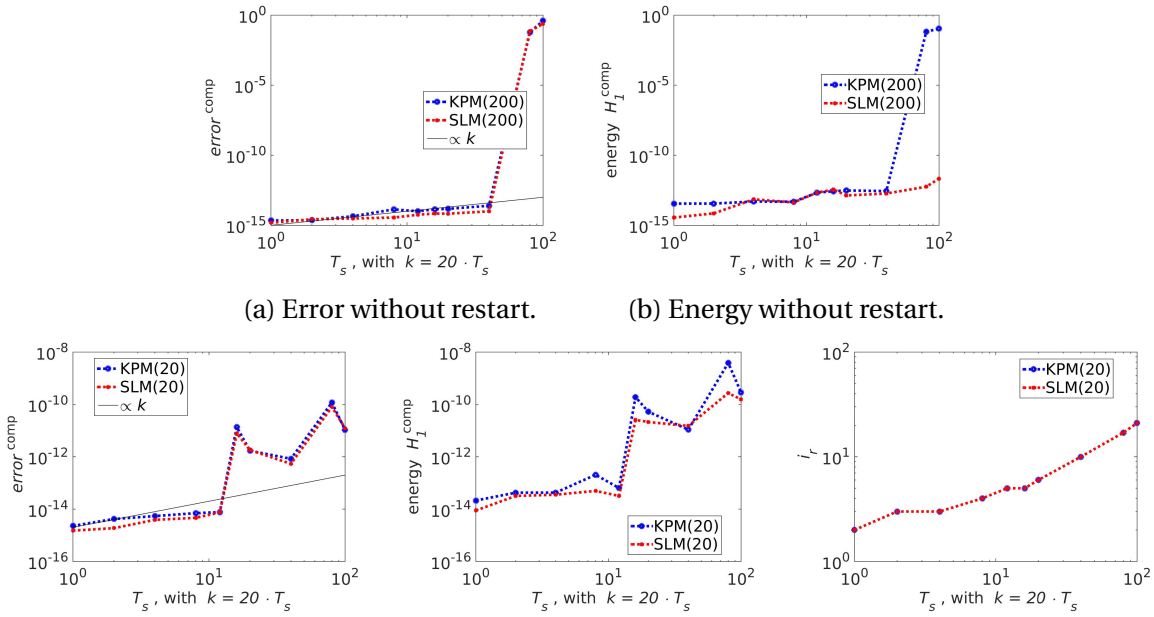
Figure 5.3: The pictures shows which n gives convergence for different m and T_s. $k = 200$ over 10 seconds unless stated.

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

5.4 Energy and error as a function of time

Suitable n is the same in Section 5.3 as in Section 4.3 for $m = 20$, thus n will be kept at the same value as in Chapter 4. This means that $n = 200$ when restart is not enabled, $n = 20$ and $\iota = 1e-6$ when restart is enabled.

5.4.1 For `semirandom`



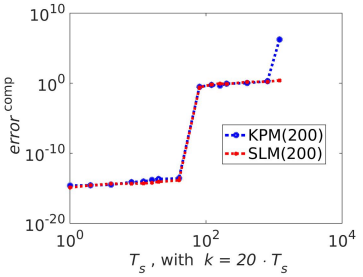
(a) Error without restart. (b) Energy without restart.

(c) Error with restart. (d) Energy with restart. (e) Number of restarts.

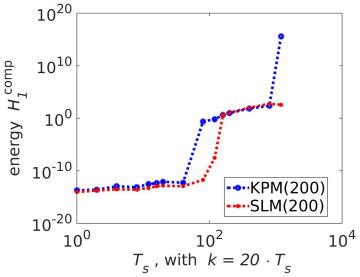
Figure 5.4: The pictures show the change in error and energy as a function of time. Restart is not enabled for the top pictures, restart is enabled for the bottom pictures, $m = 20$.

Figure 5.4 is very similar to Figure 4.8. The only difference is the more powerful divergence occurring at the last points. The restart makes `SLM` and `KPM` behave very similar. The restart also gives a much better estimate of error and energy than not restarting. This makes restarting a better idea here than in Chapter 4.

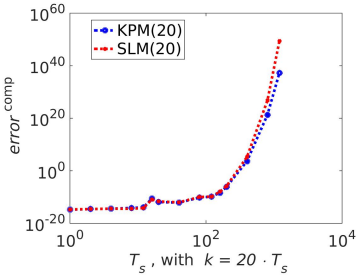
Figure 5.5 shows the divergence for the projection methods. The figure also shows the linear increase in the number of restarts, and that `SLM`'s energy is preserved a little longer than the energy for `KPM`.



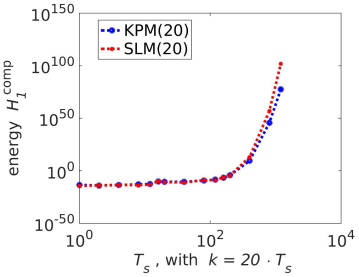
(a) Error without restart.



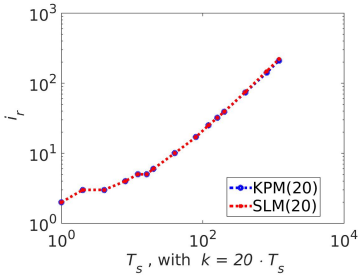
(b) Energy without restart.



(c) Error with restart.



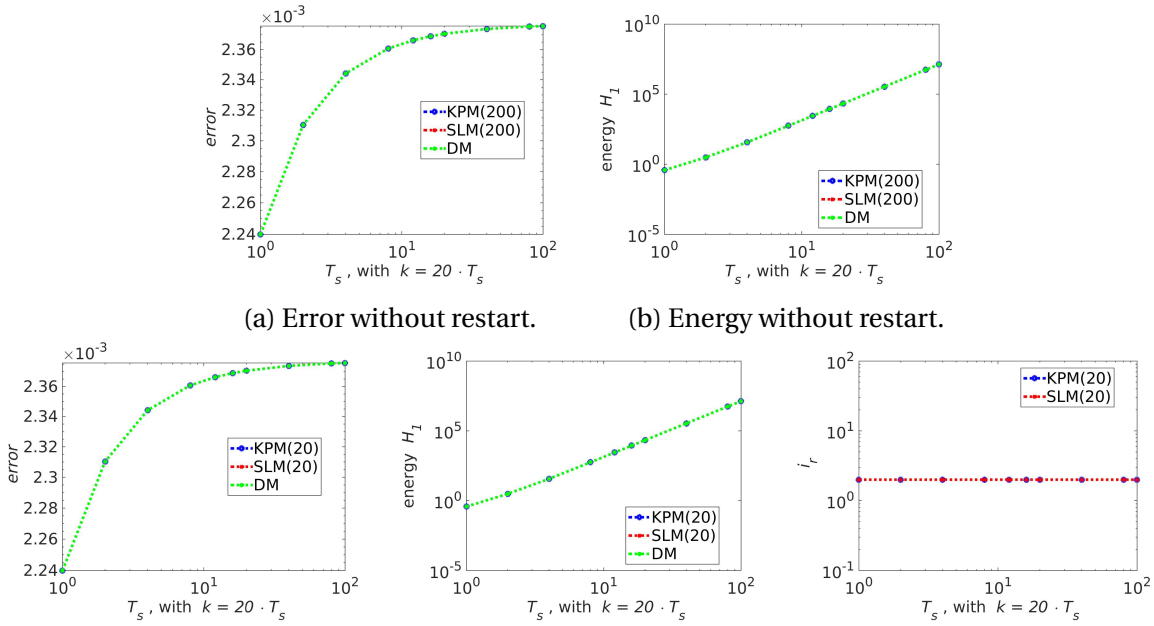
(d) Energy with restart.



(e) Number of restarts.

Figure 5.5: The pictures show the change in error and energy over a very long time domain. $m = 20$, restart is not enabled for the top pictures, restart is enabled for the bottom pictures.

5.4.2 For `wave`



(a) Error without restart.

(b) Energy without restart.

(c) Error with restart.

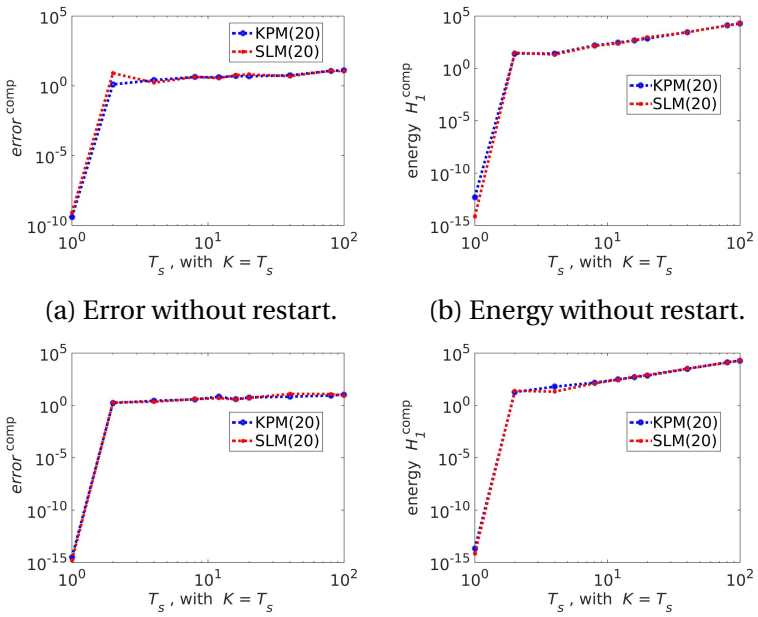
(d) Energy with restart.

(e) Number of restarts.

Figure 5.6: The change in error and energy as a function of time. $m = 20$, restart is not enabled for the top pictures, restart is enabled for the bottom pictures. Since `wave` is used, error and energy are compared to the analytical solution.

All methods in Figure 5.6 performs very similarly. With `wave` and $m = 20$ there is no need to use a particularly big n or restart. This is the reason why `wave` is so little used in this chapter. It also shows that it is not possible to see the difference between `DM` and the projection methods.

5.4.3 Windowing



(a) Error without restart.

(b) Energy without restart.

(c) Error with restart.

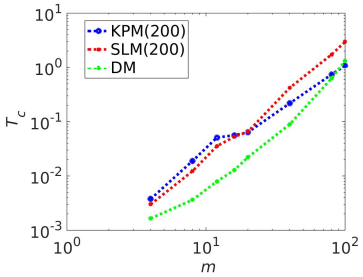
(d) Energy with restart.

Figure 5.7: Windowing with $m = 20$, $k = 20$.

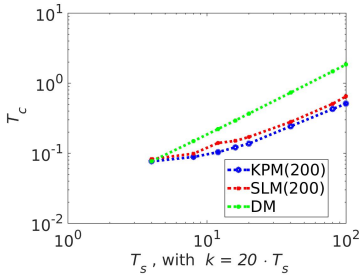
Figure 5.7 shows that windowing does not work with varying energy, thus a very promising solution strategy from the previous chapter has a limiting factor.

5.5 Computation time

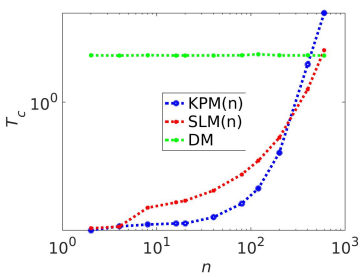
Computation times are somewhat different from the case with constant energy due to the difficulties with convergence.



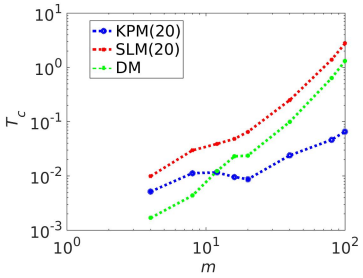
(a) Computation time as a function of m without restart.



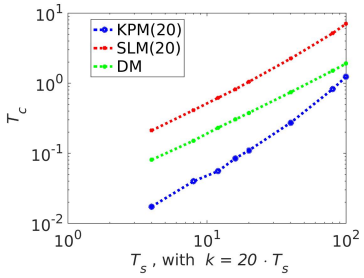
(b) Computation time as a function of k without restart.



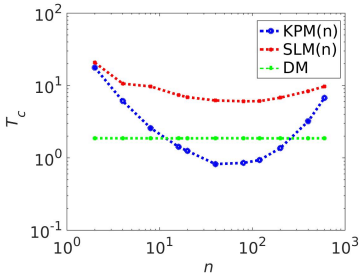
(c) Computation time as a function of ns without restart.



(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 5.8: A figure of the computation times with and without restart. $n = 200$, $T_s = 100$, $k = 2000$, and $m = 20$ unless stated.

Figure 5.8 shows that the computation times here are quite similar to the computation times in Section 4.6.1. The exceptions are shown in Figure 5.8a, in this case both `KPM` and `SLM` is slower than `DM`. This means that `SLM` is never faster than `DM`. `KPM` is only faster if m is large, T_s is small, n is well chosen, and restart is enabled.

Chapter 6

Conclusion

The conclusion is, for simplicity, divided in the same two cases that the results were divided into.

!!!!!!!!!!!!!!!!!!Et par ting!!!!!!!!!!!!!!!!!!!!!!!

- teksten i begynnelsen av denne seksjonene skal forbedres, slik at man kan lese sammendrag, introduksjon og deretter forstå alt som står i dette kapitelet.
- Innføre ODe45 som en løsemetoden i oppgaven min for å sammenligne metodene.
- Lese nøye gjennom SLM og Praktical (og så videre)!!!!!!!!!!!!
- Fikse resultatene mine, de er dårlige!!!!!!!!!!!!!!!
- Legg til en tabell om hvor gode metodene mine er i konklusjonen?
- referer windowing!!!

6.1 Constant energy

The error for both `KPM` and `SLM` is found to increase linearly, when a suitable n is used. On large time domains or with small n it was found a sudden increase in error. If restart is enabled the sudden increase is unbounded and exponential, while it is bounded when restart is not enabled.

The sudden increase vanishes when windowing is used.

The energy for both projection methods is increasing very slowly with the length of the time domain, and about as fast as for DM. This suggests that the increase is actually a result of rounding errors, and not a fault in the methods. The energy for SLM is always preserved if restart is not enabled.

Because of SLM's energy preserving properties it was predicted that it would massively outperform KPM. This has proven to only be partially true, as the error will be equally big for both projection methods. If a small error is necessary there are two ways this can be done: Either by restarting or with a larger n. If restarting is used, SLM looses its energy preserving property, and behaves very similarly to KPM. If n is chosen larger, SLM's error will decrease, but so will KPM's error and energy. The two methods will again perform similarly. Thus, in practice SLM's energy preserving property only gives SLM a small advantage over KPM.

Suitable n is found to be independent of the size of the matrix, and increase linearly with the length of the time domain. If restart is used, n can be about a tenth of the n used without restart.

SLM is near its fastest performance if restart is not enabled. In this case it performed about equal to KPM under the same conditions. KPM is equally fast if n is chosen optimal, and restarting is used. The optimal value for all m and k is not explored thoroughly enough in this text to give a general rule. Both projection methods can be faster than DM.

If an exact solver is used it is possible to achieve better accuracy with small computation time. The energy for SLM increases linearly in this case, and is constant for KPM. This case is unfortunately not directly comparable to the case where the numerical integrators are used because of the limitations of the test problems.

I conclude that SLM is better when error is no concern. If the error should be small, KPM's error and energy is just as small as SLM's, while being faster. The divergence problem happening on

large time domains can be solved with windowing, without hurting computation time.

6.2 Varying energy

SLM has a slight advantage over KPM regarding energy estimation, but only when restart is not enabled. Windowing and the exact solvers does not work. Thus, some of the important reasons to use the projection methods are removed. Both SLM and KPM manages to get error and energy close to DM on small time domains. On large time domains a sudden increase happens, much the same as with constant energy. Since windowing is not possible, this makes the projection methods useless on time domains over a certain size. This size will depend on nm this relationship has not been thoroughly enough explored. This restriction might not be too big since the linearly increasing error of DM will make any approximation useless when time domains are above a certain size.

Suitable n is found to be depending linearly on the length of the time domain, and on m.

Convergence is more difficult for the projection methods in this case than when the energy was constant. This means that either a larger n, or more restarts are necessary to achieve convergence. This results in longer computation times, making SLM consistently run slower than DM. This eliminates the reason to use SLM in the first place. KPM is barely faster in some cases. Since KPM has a comparable error and energy it may, under some assumptions, be more desirable than DM.

6.3 Further work

It could be interesting to see how the error and energy behave with a random Hamiltonian matrix, with a known analytical solution. Specifically with an exact solver for the initial problem, and trapezoidal rule for the restarts. The relation between n, k, and m could also be explored more thoroughly.

Graphics cards are designed to be used on small matrices in parallel [17]. When the energy is constant, windowing can be used to ensure convergence. This could then be used to solve non linear Hamiltonian problems in parallel under optimized conditions.

Bibliography

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

[2] Peter Benner a, 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*.

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

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

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

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

[7] 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.

[8] 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.

[9] 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.

[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.