NTNU
Department of Mathematical Sciences

Projection methods for linear Hamiltonian systems

Sindre Eskeland

February 22, 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

Two projection methods (the symplectic Lanczos method and the Krylov projection method) are compared to each other on linear Hamiltonian systems. Energy, error and computation time is the primary concern. Energy preservation for the symplectic Lanczos method is proved and shown under some assumptions, along with convergence for both projection methods.
The projection methods are also compered to each other on non-autonomous linear Hamiltonian systems.

Contents

Chapter 1

Introduction

A matrix A is said to be Hamiltonian if [8]

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

where

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

and I_j is the $j \times j$ identity matrix. Hamiltonian systems on the form

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

is found in several branches of physics and mathematics, eg. planetary movement, particle physics and control theory. The matrix A is in theses cases large sparse and Hamiltonian. Acquiring good approximations can be computationally costly. A projection method that exploits the property of these matrices is the symplectic Lanczos method (SLM). SLM projects a large matrix A onto a smaller dimensional Hamiltonian problem, making further calculations less computationally demanding. This property has made SLM a popular choice when finding eigenvalues of large Hamiltonian matrices, see eg. [5], [30] and [6]. Since SLM preserves the Hamiltonian structure of the matrix, energy of the system (1.1) is preserved.

Arnoldi's algorithm, together with the Krylov projection method (KPM) is another much used

projection method. KPM has no structure preserving properties, which makes it faster at the cost of energy preservation.

This text will look at how these two projection methods compare to alternatives without projecting: in error approximation, energy preservation, and computation time.

Equation (1.1) has the well known analytical solution

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

Taking the matrix exponential is a very costly operation, but can be a great way to exploit the smaller matrices produced by the projection methods. This will also be looked at closely.

In addition to the case with Hamiltonian systems and constant energy, the projection methods will be tested against each other when the energy is nonconstant as in equation (1.2).

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

where p is a vector, and $f(t)$ is a scalar time dependent function. Matlab notation will be used where applicable.

Chapter 2

Theory

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

2.1 Zero initial condition

For both `KPM` and `SLM` 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 [11]:
Start by shifting the solution

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

then rewrite the original equation as

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

The equation above solves the shifted problem. The original problem can be solved by shifting it back with

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

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 as a collective term to describe the product Au_0, or

any constant vector occurring on the right hand side of equation (1.1).

2.2 Energy

It is well known that the energy of a system on the form of equation (1.1) can be expressed as [13]

$$\mathscr{H}_1(u) = \frac{1}{2} u^\top(t) J A u(t)$$

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

$$\mathscr{H}_2(\hat{u}) = \frac{1}{2} \hat{u}^\top(t) J A \hat{u}(t) + \hat{u}^\top(t) J b. \tag{2.2}$$

In theoretical derivations $\mathscr{H}_2$ will be used, as the shifted problem is the one actually solved. In figures however, $\mathscr{H}_1$ is used since the unshifted solution is what is sought.

There will be a discussion about different types of test problems. The ones marked "constant energy" will be problems on the form of equation (1.1). The other type is problems marked "varying energy", which is on the form of equation (1.2). Test problems will be presented in section 3.3.

2.3 Integration methods

The time domain $[0, T_s]$ will be divided in k pieces, so that the step size is $h_t = T_s/k$. The time discretized solution of $u(t_j)$ will be called U_j, with $t_j = j h_t$ 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 definition and the iteration scheme of the different methods are given in table 2.1.

Trapezoidal rule and midpoint rule are the same methods if the energy is constant. When F_j is not constant, midpoint rule should have an advantage because it is symplectic [29].

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 needs to be saved for midpoint rule as for the other methods. Trapezoidal and midpoint rule have quadratic convergence rates, while forward Euler has linear convergence. To compare the methods it is therefore necessary to use the squared number of points for forward Euler as for the other methods. g is the right hand side of equation (1.1) or (1.2).

Trapezoidal rule (`trap`) [26]	$\frac{U_{i+1}-U_i}{h_t} = g\left(\frac{1}{2}(t_i + t_{i+1}), \frac{1}{2}(U_i + U_{i+1})\right)$ $U_{i+1} = (I - \frac{Ah_t}{2})^{-1}\left(U_i + \frac{h_t}{2}\left(AU_i + (F_{i+1} + F_i)\right)\right)$
Forward Euler (`Euler`) [27]	$\frac{U_{i+1}-U_i}{h_t} = g(t_i, U_i)$ $U_{i+1} = U_i + h_t\left(AU_i + F_i\right)$
Midpoint rule (`mid`) [28]	$\frac{U_{i+1}-U_i}{h_t} = g\left(t_{i+\frac{1}{2}}, \frac{1}{2}(U_i + U_{i+1})\right)$ $U_{i+\frac{1}{2}} = U_i + \frac{h_t}{2}(AU_i + F_{\frac{1}{2}})$ $U_{i+1} = (I - \frac{Ah_t}{2})^{-1}(U_{i+\frac{1}{2}} + \frac{h_t}{2}F_{i+\frac{1}{2}})$

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 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`)	$U_i = \texttt{expm}(A \cdot t_i) \cdot b - b$

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

In addition to the iteration schemes in table 2.1, some exact solvers are used. They are presented in table 2.2.

2.3.1 Energy conservation for trapezoidal rule

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

$$\begin{aligned} \dot{u}(t) &= Au + b \\ u(0) &= 0. \end{aligned} \tag{2.3}$$

Note that b can be any constant vector, not just Au_0. The proof is based on [25]. 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 iterations scheme for 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 (2.3) gives

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

Substituting $\frac{U_{j+1}+U_j}{2}$ for u gives the gradient of the energy for this function:

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

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

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

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

$$\mathscr{H}_1(U_{j+1}) - \mathscr{H}_1(U_j) = 0.$$

Hence trapezoidal rule conserves the energy for functions with constant energy. Since the initial conditions are satisfied the energy will have the correct value at all times.

2.4 Windowing

The projection methods has a tendency to stop working when the time domain is too large. A fix to this might be to divide the time domain in smaller pieces, and solve each piece individually. How this is done is described in the next paragraph.

Let T_s denote simulated time, K be the number of subintervals T_s is divide into, and k be the number of pieces each the K is divided into. Each of the K subintervals is then solved as separate problems, with the initial conditions updated. This is explained in a more precise manner in algorithm 1. This method will be called "windowing", as this is the name my supervisor Elena Celledoni [2] used to describe the method.

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 $j = 1, 2, \cdots, K$ **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 orthogonalisation 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). The derivation of the methods, together with a proof of energy perseverance for SLM, is presented in this section.

Projection methods are ways to make approximated solutions from a subset of the original problem by projecting a problem of several dimensions onto a smaller dimensional problem. One great feature of these methods is that a smaller system of equations can be used to obtain solutions, making further computations less demanding. However finding this projected system can be time consuming. Another drawback is that the projected solutions are approximations. The approximated solution can be improved by something called restarting, meaning using the projection method again, to solve an equation for the difference between the projected solution and the unprojected solution. This can be done repeatedly to obtain the desired accuracy.

Assume that the equations are on the form

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

when these methods are used. Note that the initial values are zero otherwise it is difficult to know how well the initial value is approximated. It is also important that A is a Hamiltonian matrix when using `SLM`.

Note that the relation between $\hat{m}$, $\tilde{m}$ and m used in the algorithms is given by $\hat{m} = 2\tilde{m} = 2(m-2)^2$ and that $n = 2\tilde{n}$. $\hat{m}$ is the dimension of the original system and n is the size of the orthogonal system. n is called the restart variable. Don't worry to much about these details, it is just a way to simplify the expressions. The reason to simplify them this way will be described in section 3.3.

When talking about the true solution, it is the meant as the unprojected solution (the same problem, solved exactly the same way, except without any projection method), and will later be referred to as the direct method(`DM`).

2.5.1 Arnoldi's Algorithm and the Krylov projection method

This section is loosely based around the derivation of the method done in [15] and [17].

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 containing all elements $(h_{i,j})_{i,j=1,\cdots,n}$. Then the following holds [31]:

$$\begin{aligned} AV_n &= V_n H_n + h_{n+1,n} v_{n+1} e_n^\top \\ V_n^\top A V_n &= H_n \\ V_n^\top V_n &= I_n. \end{aligned} \tag{2.5}$$

Here, e_n is the nth canonical vector in $\mathbb{R}^n$. n is the number of iterations performed by Arnoldi.

Algorithm 2 Arnoldi's algorithm[32]

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 $j = 1, 2, \cdots, n$ **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}, $h_{n+1,n}$.

By using the transformation $u(t) \approx V_n z_n(t)$, equation (2.4) can, with the help of equation (2.5), be written as

$$\begin{aligned} \dot{z}_n(t) &= H_n z_n(t) + \|b\|_2 e_1 f(t) \\ z_n(0) &= 0. \end{aligned} \tag{2.6}$$

The approximated solution of the original problem can be attained by the relation

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

where u_n is the approximated solution found with n as a restart variable.

Convergence for the method can be shown by finding an expression for the residual, which can be written as

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

This can be rewritten with $u_n(t) = V_n z_n(t)$, to get

$$r_n(t) = v f(t) - V_n \dot{z}_n(t) + A V_n z_n(t).$$

By equation (2.5), it can be simplified to

$$r_n(t) = h_{n+1,n} e_n^\top z_n(t) v_{n+1}. \tag{2.7}$$

Since $h_{n+1,n}$ is zero for some $n \leq \hat{m}$ ($V_{\hat{m}} h_{n+1,n} v_{n+1} = 0$ by construction), 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 drawback is also that there is no way of knowing how well the solution is approximated in advance. The size of $h_{n+1,n}$ does say something about how well the solution is approximated (smaller $h_{n+1,n}$ means a smaller error), though it is only available after solving the problem. If the approximation is not sufficient the solution must be recalculated with larger n, unless you perform a restart.

A restart considers the difference $\epsilon_n^{(i)}(t) = u(t) - u_n^{(i)}$ given by:

$$\dot{\epsilon}_n^{(i)}(t) = A\epsilon_n^{(i)}(t) - r_n^{(i)}, \tag{2.8}$$

where

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

The expression for $r_n^{(i)}$ is exactly the same as in equation (2.7), except that $z_n(t)$ is replaced with $\epsilon_n(t)$, and the counter i is present. $u_n^{(i)}$ is the solution obtained after i restarts with n as a restart variable. Equation (2.8) can be simplified by using equation (2.5), and writing $\epsilon_n^{(i)}(t) = V_n \delta_n^{(i)}(t)$, to obtain the following:

$$\dot{\delta}_n^{(i)}(t) = H_n^{(i)} \delta_n^{(i)}(t) + e_1 h_{n+1,n}^{(i-1)} e_n^\top \delta_n^{(i-1)}(t), \quad i \geq 1. \tag{2.9}$$

The solution 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 (2.6). H_n, v_{n+1}, $h_{n+1,n}$ and V_n without counting variables will always be used together with equation (2.6).

Repeatedly solving this equation can increase the accuracy of the approximated solution within an arbitrary constant of the desired 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 [16].

2.5.2 Symplectic Lanczos method

SLM and KPM are very similar, which is easy to see when comparing equation (2.5) and (2.10). The main difference is that orthonormality in Arnoldi is replaced by symplecticity in SLM. This makes the derivation quite similar.

Let $S_n = [v_1, v_2, \dots v_{\frac{n}{2}}, w_1, w_2, \dots w_{\frac{n}{2}}]$ be a set of J-orthogonal vectors satisfying the following equations [18]:

$$\begin{aligned} AS_n &= S_n H_n + \zeta_{n+1} v_{n+1} e_{\hat{m}}^\top \\ J_n^{-1} S_n^\top J_{\hat{m}} A S_n &= H_n \\ S_n^\top J_{\hat{m}} S_n &= J_n. \end{aligned} \tag{2.10}$$

Here H_n is an $n \times n$ matrix. $\tilde{n} = \frac{n}{2}$ is the number of iterations the algorithm performed, since the algorithm makes two vectors per iterations: v_j and w_j.

The process of making the vectors and the matrix is a little more involved than for Arnoldi's algorithm, so there will be no thorough explanation of how it works, except for in Algorithm 3.

Transform the problem in equation (2.4) with $u(t) \approx S_n z_n(t)$, multiply with $J_n^{-1} S_n^\top J_{\hat{m}}$, and simplify with equation (2.10) to obtain

$$\dot{z}(t) = H_n z_n(t) + J_n^{-1} S_n^\top J_{\hat{m}} b f(t).$$

Algorithm 3 Symplectic Lanczos method [10], with reortogonalization from [7].

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

This can be simplified when writing $b = S_n e_1 \|b\|_2$ and using equation (2.10):

$$\dot{z}(t) = H_n z_n(t) + \|b\|_2 e_1 f(t). \tag{2.11}$$

Equation (2.11) and (2.6) are identical, except for the underlying assumptions.

Since $S_{\hat{m}}^\top J_{\hat{m}} \zeta_{n+1} v_{n+1} = 0$ by construction [9], the proof of convergence is very similar to the proof for `KPM`.
The residual of the method can be written as

$$r_n(t) = v f(t) - \dot{u}_n(t) A u_n(t).$$

This can rewritten with $u_n = S_n z_n(t)$.

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

By equation (2.10) it becomes

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

Since ζ_{n+1} will approach zero as n grows, r_n will approach zero, and the method will converge.

A restart can also here be performed if the accuracy of the solution is not satisfactory. This can be derived by looking at the difference $\epsilon_n = u(t) - S_n z_n(t)$:

$$\dot{\epsilon}_n^{(i)}(t) = A\epsilon_n^{(i)}(t) + r_n^{(i)}(t), \tag{2.12}$$

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 $\epsilon_n^{(i)}(t) = V_n \delta_n^{(i)}(t)$ to obtain

$$\dot{\delta}_n^{(i)} = H_n^{(i)} \delta_n^{(i)} + J_n^{-1} {S_n^{(i)}}^\top J_{\hat{m}} \zeta_{n+1}^{(i-1)} v_{n+1}^{(i-1)} e_n^\top \delta_n^{(i-1)}.$$

This can be further simplified to

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

The solution 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 (2.11). H_n, v_{n+1}, ζ_{n+1} and S_n without counting variables refers to equation (2.11).

Proof of convergence and other interesting results for this method can be found in [3].

Proof that `SLM` without restart is energy preserving

If equation (2.11) is solved by an energy preserving method, eg. trapezoidal rule, the energy will be preserved [14]. The energy of this equation is

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

While the energy of the original problem is

$$\mathscr{H}(u_n) = \frac{1}{2}u_n(t)^\top JAu_n(t) + u_n(t)^\top Jb.$$

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

$$\mathscr{H}(u_n) = \frac{1}{2}z_n(t)^\top S_n^\top JAS_n z_n(t) + z_n(t)^\top S_n^\top Jb.$$

Use that $b = S_n e_1\|b\|_2$, and simplify with equation (2.10):

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

This results in:

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

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 V_n is not symplectic.
SLM with restart is also not energy preserving since the error equation is depending on time, which ruins the energy preserving properties.

Residual energy

Equation (2.14) and (2.15) are solely used to describe the residual energy of SLM. $\mathscr{H}_3$ describes the residual energy of the projected method in u_n space, that is, after transforming back from z_n space. $\mathscr{H}_4$ is the residual energy in z_n space. In theory these energies should be equal and zero. Section 4.5.6 shows how this holds up in practice. They are presented in the equation below, and can easily be derived from (2.12) and (2.13).

$$\mathscr{H}_3(t) = \frac{1}{2}\epsilon^{(1)^\top}(t)J_m A\epsilon^{(1)} + \epsilon^{(1)^\top} J_m h^{(1)}_{n+1,n} v^{(1)}_{n+1} e_n^\top z(t) \tag{2.14}$$

$$\mathscr{H}_4(t) = \frac{1}{2}\delta^{(1)^\top}(t)J_n H^{(2)}_n \delta^{(1)} + \delta^{(1)^\top} S^{(2)^\top}_n J_m h^{(1)}_{n+1,n} v^{(1)}_{n+1} e_n^\top z(t) \tag{2.15}$$

The iteration variables are present since it considers the residual with one restart.

2.5.3 A comment on the orhogonalisation methods

The derivation of the methods above shows the many similarities between SLM and KPM. The only practical difference between the two methods is the orthogonalisation methods. This makes implementation of the methods very similar.

How to obtain the correct number of restarts to get a good estimate of the solution will be discussed in section 3.4 and 4.4.

2.5.4 Direct method

It is important to have some method to compare these projection methods with. This will be done by solving the problem exactly the same way, but without using any projection method. This will be called direct method, or DM.

The energy of DM will be constant if an energy preserving method is used, while the error will increase linearly with T_s [22]. A goal of this text is to see how KPM and SLM's error and energy behaves compared to this.
The proof that DM is the natural method to compare with will be shown here with trapezoidal rule for KPM. The proof is very similar for SLM.

We start by stating the desired equation discretized with the trapezoidal rule:

$$\left(I - \frac{Ah_t}{2}\right)U_{i+1} = \left(U_i + \frac{h_t}{2}\left(AU_i + F_{i+1} + F_i\right)\right.$$

Applying the transformation $U_i = V_n Z_i(t)$ gives

$$\left(V_n - \frac{AV_n h_t}{2}\right)Z_{i+1} = \left(V_n Z_i + \frac{h_t}{2}\left(AV_n Z_i + F_{i+1} + F_i\right)\right).$$

Using equation (2.5) gives

$$\left(V_n - h_t\frac{V_n H_n + h_{n+1,n}v_{n+1}e_n^\top}{2}\right)Z_{i+1} = \left(V_n Z_i + \frac{h_t}{2}\left((V_n H_n + h_{n+1,n}v_{n+1}e_n^\top)Z_i + F_{i+1} + F_i\right)\right).$$

When using the trapezoidal rule directly on (2.6), the result is

$$\left(V_n - h_t\frac{V_n H_n}{2}\right)Z_{i+1} = \left(V_n Z_i + \frac{h_t}{2}\left(V_n H_n Z_i + F_{i+1} + F_i\right)\right).$$

If we now let n grow, $h_{n+1,n}$ will become zero and the projected method will converge towards the unprojected method.

2.5.5 Linearity of the methods

The projection 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. $b + F(t)$, or (more illustrative)

$$\dot{u}(t) = Au(t) + bF(t).$$

In this case the projection method needs to be used two times, one time to solve $\dot{u}_1(t) = Au_1 + b$, and another to solve $\dot{u}_2(t) = Au_2 + F(t)$. $u_1(t)$ 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 $\tilde{b}\tilde{F}(t) = b + F(t)$.

An even bigger problem arises when a differential equation has a source term that is not separable in time and space, read more about this in [15] and [17]. An equation that is separable in time and space can be written as $p(t,x,y) = g(x,y) \cdot F(t)$. If $p(t,x,y)$ is not separable, this is impossible. As an example, consider the wave equation [21]:

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

This equation cannot be discretized with a single vector b. If this is discretized to be on the form of (1.1) and solved with a projection method, it would look like:

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

where $\hat{m}$ is the size of the matrix, and $F_i(t)$ is a time dependent function after spacial discretization. The reason for this is that every different point, x_i and y_i, gives a different time dependent function.This means that equation (2.5.5) needs to be solved $\hat{m}$ times to obtain the solution. This gives the projection methods a terrible run time. It is not advised to use any projection method if this is the case [17].

Table 2.3: Computational cost of some mathematical operations. n is restart variable, $\hat{m}$ is the size of the large matrix, and k is the number of steps in time. Numbers from this table is obtained from [23].

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 methods with trapezoidal rule. i_r is the number of restarts needed for the method to converge. For windowing $K\cdot k$ is equal to k for the other methods. Windowing for DM is not interesting since it has exactly the same run time, and is exactly the same method.

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. $\mathcal{O}((n\hat{m}^2+kn^3+\hat{m}nk)i_r)$
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. $\mathcal{O}((n\hat{m}^2+kn^3+\hat{m}nk)i_rK)$

2.5.6 Number of operations

This section contains a brief discussion about computation times for the different methods presented.

A table of computational cost for different operations is given in Table 2.3.

Surprisingly the asymptotic cost for KPM and SLM is the same. A lot of small additional costs for SLM is hidden here, meaning that KPM probably has a smaller computation time, but the computation times are increasing similarly. The windowing is also a wildcard a since lot of costly operations are hidden. The difference between the projection methods and DM strongly depends on n and r_n, making it impossible to conclude anything without actually doing it, except that all methods are linear with k.

2.6 SLM and its eigenvalue solving properties

If you do a quick internet search for "symplectic Lanczos method", most articles you find are in relation to symplectic Lanczos method's ability to approximate eigenvalues of Hamiltonian matrices. In this section I will try to explain what makes this algorithm so attractive for these types of problems. (This section is based on reading [5],[30], [6], and [19].)

The eigenvalue problem is found in many different areas, eg. control theory, model reduction, system analysis. No other algorithm exploits the Hamiltonian and sparse structure of these matrices.

SLM is a relatively fast algorithm for transforming a big sparse Hamiltonian matrix to a smaller sparse Hamiltonian matrix with similar properties. Eigenvalues of Hamiltonian matrices comes in pairs, $\{\pm\lambda\}$, or in quadruples, $\{\pm\lambda, \pm\bar{\lambda}\}$. Since the projected matrix is Hamiltonian, it is easier to find these pairs or quadruples. An other important property is that the biggest, and therefore often most important eigenvalue, is found first.

The method is not without flaws. Frequent breakdown 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 properties, and without a too high cost. Unfortunately not all are very efficient, and there is still much ongoing research on the topic.

An other method based on SLM is the SR - algorithm which has similarities with the QR - algorithm. SR applies symplectic operations to a Hamiltonian matrix instead of applying orthogonal operations as in QR.

Not all the papers are interested in improving the method, [19] shows under which assumptions the method converges.

Interestingly, many of these papers also compare `SLM` to `KPM`.

Chapter 3

Practice

This section will be concerned with programming choices, implementation, and how important data was calculated.

3.1 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 there is no correct solution, as in `semirandom`, the error is measured as the difference between the projected solution and `DM`. This case is marked with $\text{error}^{\text{comp}}$. In some results $\text{error}^{\text{comp}}$ will also be used with `wave`.

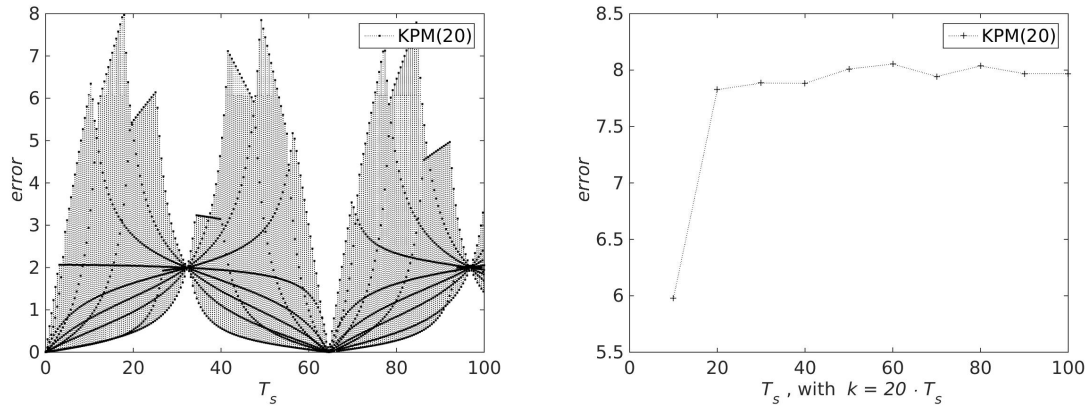
The energy is found with one dedicated energy function. The exceptions are $\mathscr{H}_3$ and $\mathscr{H}_4$ which are calculated by different functions. The energy is calculated for each time step, and initial energy is subtracted. In the case where the energy is supposedly constant the energy is 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 section 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 section 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 performed and computation time.
Number of restarts are 0 if the projection method is not used, and 1 if the projection 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 negative numbers and 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. Initial computations or plots are not counted, as these are common for both the projection methods and `DM`. The symbol for computation time is T_c.

3.2 Pictures

All pictures are plotted with a dedicated plot tool. This makes it easier to change plots, or uncover programming faults. The greatest absolute values of energy or error are the points used in the plots. This is done by running the solving program once for each point. This removes noisy data from the figures, and makes it possible to make the points look equally spaced on logarithmic plots. In figure 3.1 the difference between the two is shown.



(a) A plot made by a single run with the solution program.

(b) A plot made by running the solving program one time for each point.

Figure 3.1: These pictures show how removing useless data can make the pictures easier to understand. The picture on the left shows the error for all points in time. The picture on the right shows the maximum error from start to each point.

`SLM` and `KPM` will on pictures be called `KPM`(n) and `SLM`(n) where n is the restart variable.

3.3 Test problems

Equation (1.1) can be the result of several discretized equations. Since `SLM` needs a Hamiltonian matrix this will be the main focus. Two different matrices are implemented. These matrices also have two test problems each. All test problems satisfies the condition

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

The discretization is done by dividing each spacial direction, $[0,1]$ in m pieces, with step size $h_s = 1/m$, so that $y_i = ih_s$, $x_i = ih_s$ with $i = 1,2,\ldots,m-1$.

3.3.1 The wave equation

The first test problem is the 2 dimensional wave equation,

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

To obtain the matrix, approximate $\frac{\partial^2 \xi}{\partial x^2} + \frac{\partial^2 \xi}{\partial y^2}$ with $\tilde{A}$, where $\tilde{A}$ is the five point stencil[24], and approximate $p(x_i, y_j)$ with $\tilde{b}_{i+(m-1)(j-1)}$, where $i, j = 1, \ldots, m-1$. Then write it as a system of first order (in time) differential equations

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

Now, let $u(t) = [q(t); w(t)]$, $b = [0; \tilde{b}]$, and

$$\begin{aligned} \tilde{A} &= \frac{2}{h_s^2}\,\text{gallery}('\text{poisson}', m-2) \\ A &= \begin{bmatrix} 0 & I_{\hat{m}} \\ -\tilde{A} & 0 \end{bmatrix} \end{aligned},$$

where A is a Hamiltonian matrix. This matrix will be referred to as `wave`, and is a second order approximation of the wave equation. Equation (3.1) can now be written as equation (1.2).

Two test problems is used to check the integrity of the solving program. One with constant energy, and one with varying energy.

In the case when the energy is constant the test problem is

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

For varying energy it is

$$\begin{aligned} q(t,x,y) &= \sin(\pi x)\, y(y-1)(t^2+1) \\ w(t,x,y) &= \sin(\pi x)\, y(y-1)(2t) \\ q_0(x,y) &= \sin(\pi x)\, y(y-1) \\ w_0(x,y) &= 0 \\ f(t,x,y) &= 2\sin(\pi x)\, y(y-1) - (t^2+1)\sin(\pi x)(2-\pi^2 y(y-1)). \end{aligned}$$

Both q and w are used when measuring the error.

3.3.2 A random test problem

The second implemented Hamiltonian matrix is random, and given by

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

Since we are interested in comparing the different projection methods to each other, the matrix is saved and reused. 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. Since the projection methods are only usable with sparse matrices, it is tridiagonal.

The test problem is given by

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

when the energy is constant, and

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

when the energy is varying.

Since the solutions to the test problems are unknown, 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 reasons to use it. It gives a tridiagonal Hamiltonian matrix with much randomness, making it difficult for the projection methods to find an approximated solution. With `wave` the matrix is always the same. This case will also show more correctly how restarting can be used to improve the solution than `wave` will.

The error will in this case be measured as the difference between the projection 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 2.5.4, it is correct to compare the solutions in this manner. When the energy is constant there is no need to do anything different, but varying energy will be compared with its non-projected equivalent, and be marked with energy$^{\text{comp}}$.

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 was divided into several small functions that were tested individually. Functions are reused as much as possible. Matlab's backslash operator is used to solve the linear systems in trapezoidal rule and midpoint rule.

There are two ways to implement the number of restarts the projection methods should per-

form. 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 it is used to describe something small[4], and both ϵ and δ were unavailable.

Chapter 4

Results for test problems with constant energy

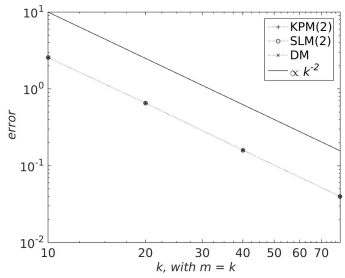
This chapter will show how error, energy and computation time changes with i_r, ι, T_s, m and n. There are special interest in seeing how the energy for `SLM` behaves, how `KPM` and `SLM` differs, and how occurring problems might be handled. The predictions and proofs made in the theory chapter will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

For all sections except the first assume that $m = 20$, and $k = 20$ per second.

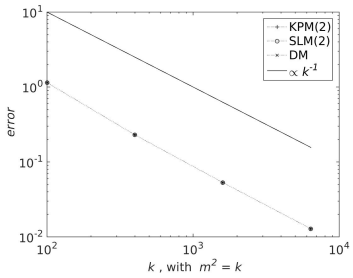
4.1 Convergence

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.

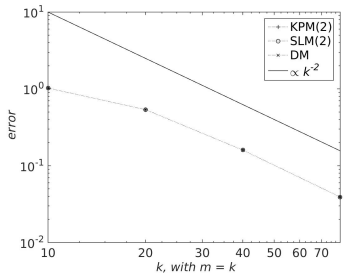
4.1.1 Numerical integration



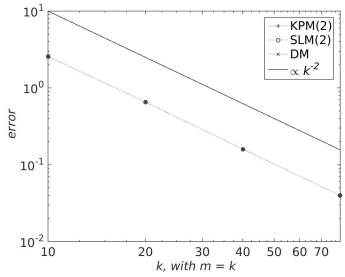
(a) Convergence of trapezoidal rule without restart.



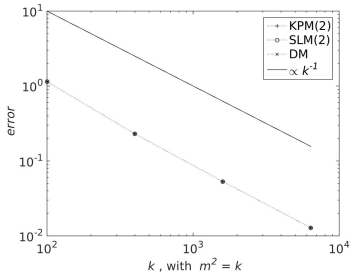
(b) Convergence of forward Euler without restart.



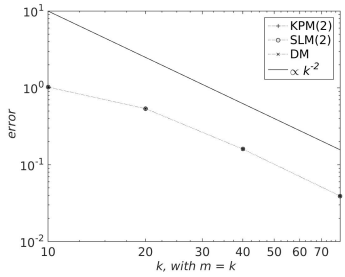
(c) Convergence of midpoint rule without restart.



(d) Convergence of trapezoidal rule with restart.



(e) Convergence of forward Euler with restart.



(f) Convergence of midpoint rule with restart.

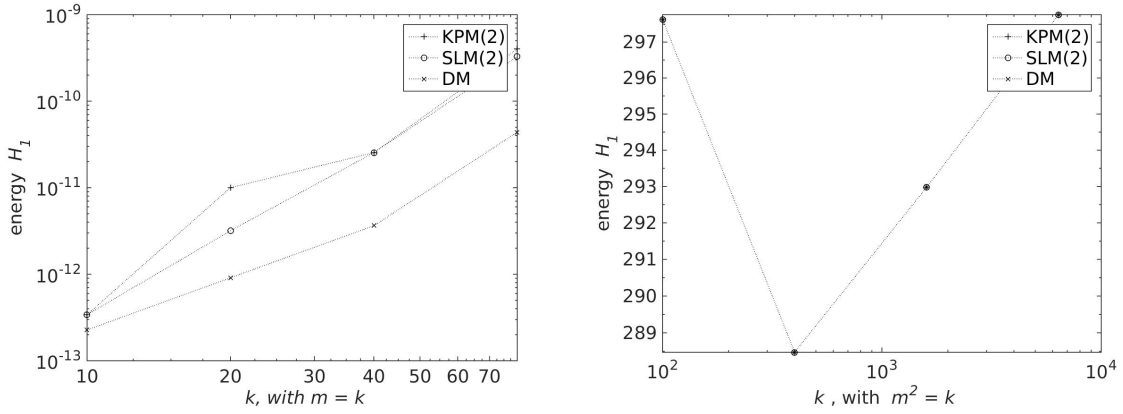
Figure 4.1: Convergence for the different integration methods. Notice that forward Euler uses k^2 points in time where the other methods uses k points in time to obtain the same accuracy. $n = 2$ is kept constant for all plots, since smaller n generally gives poorer convergence. For the figures with 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(4.2a) and trapezoidal rule (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.

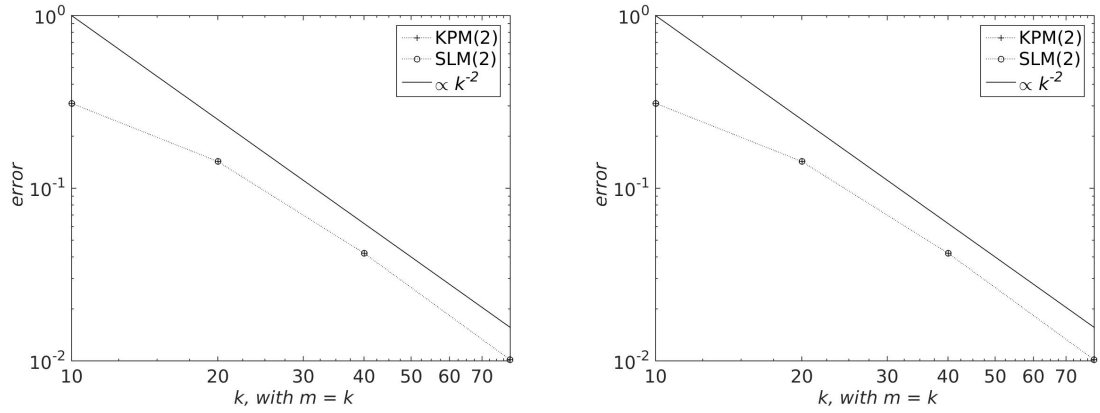


(a) Energy for trapezoidal rule.

(b) Energy for forward Euler.

Figure 4.2: A figure showing the difference in energy between trapezoidal rule and forward Euler when 1 second is simulated.

4.1.2 Exact solvers



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

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

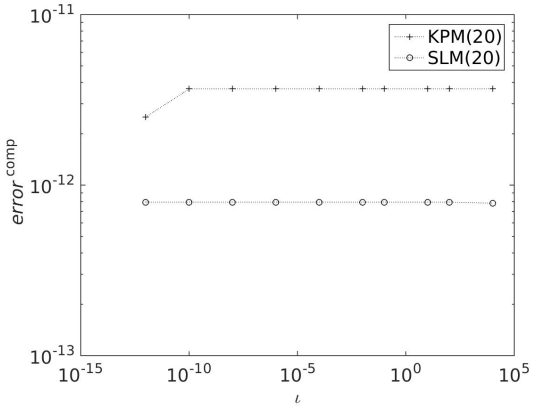
Figure 4.3: Convergence for exact solvers, with trapezoidal rule. Note that the exactness of the methods are for the integration in time, and not the spacial discretization. The expected convergence is therefore still quadratic with m. $n = 2$ because taking the matrix exponential is a costly operation 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 explored in section 4.6.1.

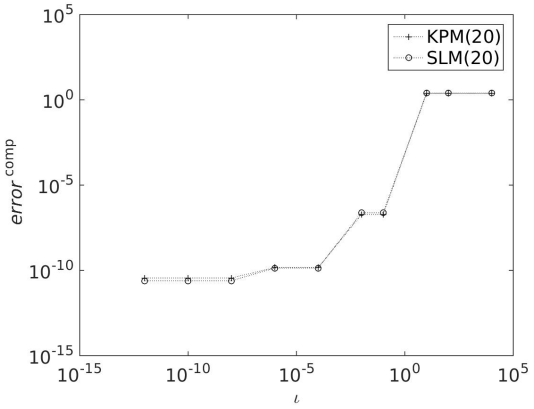
4.2 Convergence of the restart

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

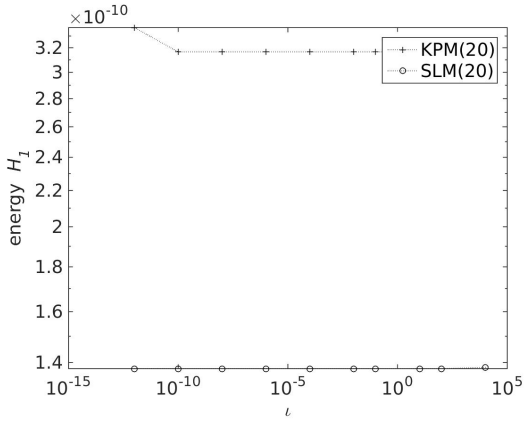
4.2.1 Convergence as a function of ι



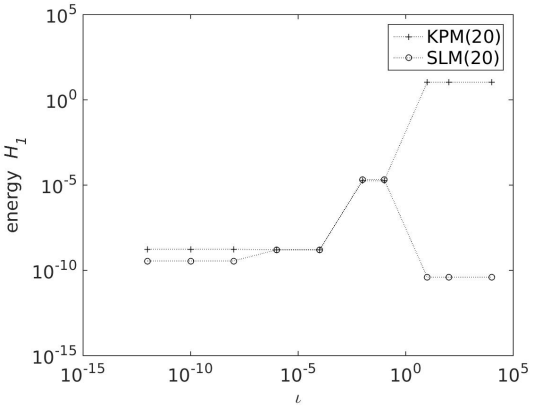
(a) Error for `wave`.



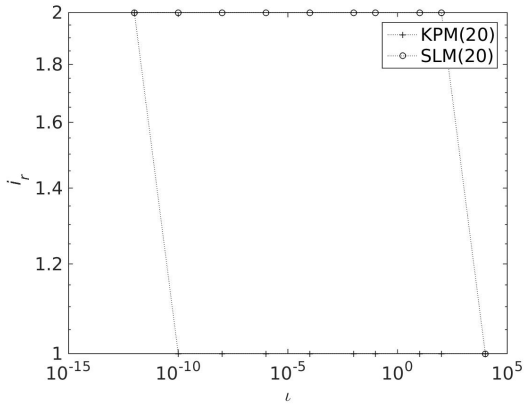
(b) Error for `semirandom`.



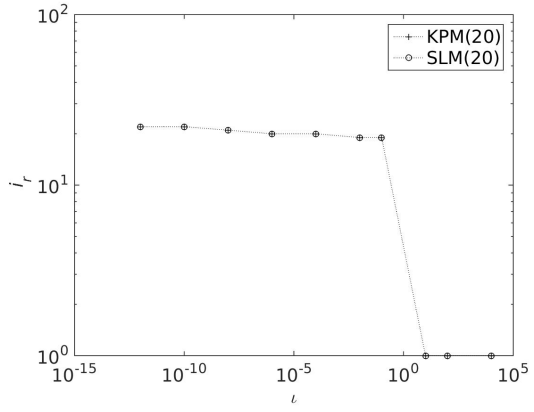
(c) Energy for `wave`.



(d) Energy for `semirandom`.



(e) Number of restarts for `wave`.



(f) Number of restarts for `semirandom`.

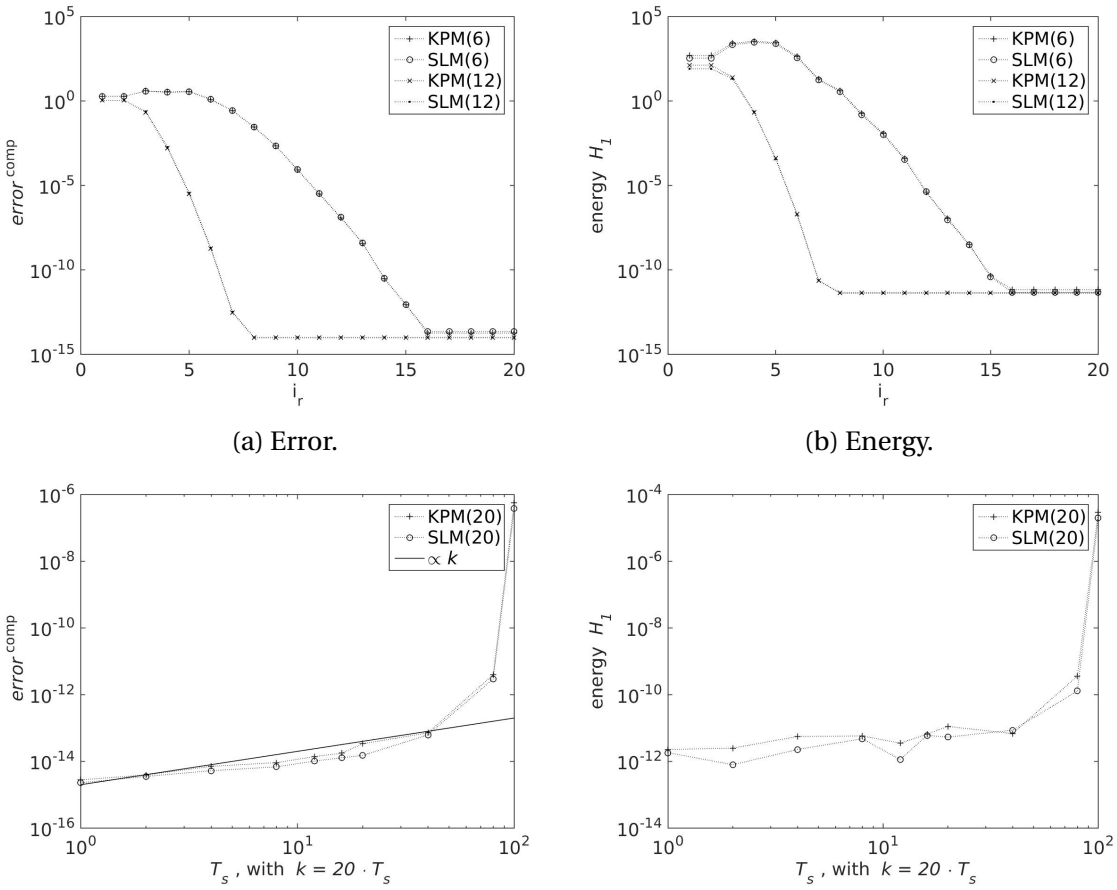
Figure 4.4: These figures show how choosing different ι affects the solution. The pictures on the left are for `wave` and on the right are for `semirandom`. This plot considers 100 seconds, with $k = 2000$, $n = m = 20$ and midpoint rule.

Figure 4.4 shows that if `wave` is used there is no reason to use the restart. From figure 4.1 it seams to also be the case for larger m. This will therefore be the end of the discussion about `wave` in this chapter. 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 4.4d and 4.4b 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. If only the energy is of interest, the restart is not necessary. For `KPM` it is much simpler, more restart means better approximations, for both error and energy.

4.3 Convergence with i_r



(a) Error.

(b) Energy.

(c) Error.

(d) Energy.

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 $i_r = 20$.

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 a certain number of restarts the change in the solution is too small to observe.

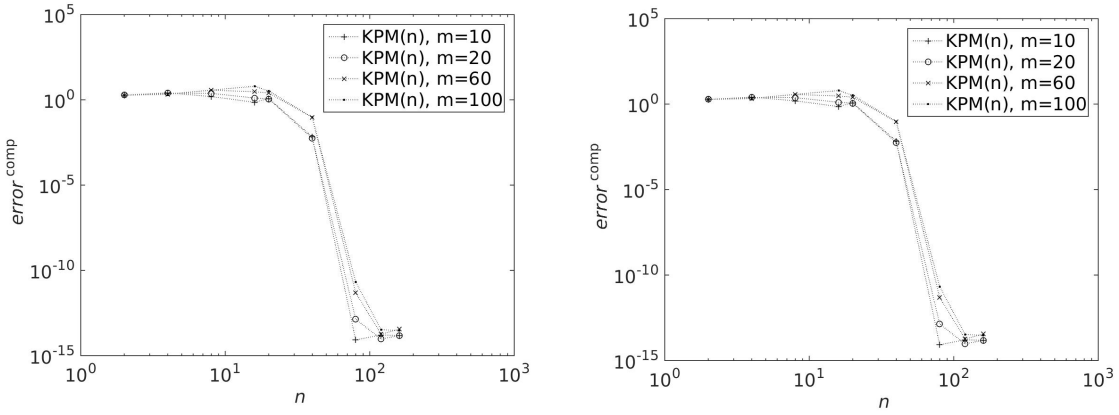
The increase in error is linear for 4.5c, which is what is predicted for these methods.

The increase in energy is sublinear. It was predicted that it would be constant for SLM. Why the increase happens will be examined later in this chapter.

4.4 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.

4.4.1 Without restart



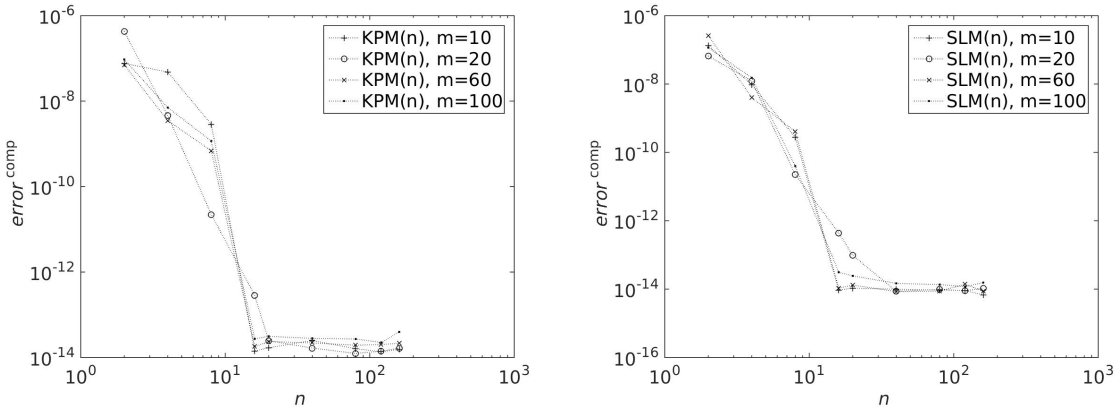
(a) Error for KPM.

(b) Energy for SLM.

Figure 4.6: The pictures shows which n gives convergence for different m when restart is not enabled. This plot is made with trapezoidal rule with $k = 200$ over 10 seconds.

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

4.4.2 With restart



(a) Error for `KPM`.

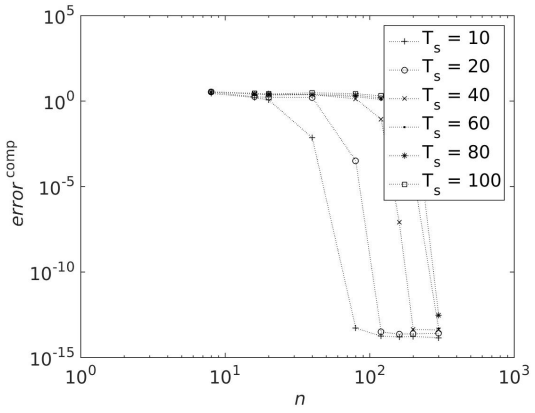
(b) Error for `SLM`.

Figure 4.7: The pictures shows which n gives convergence for different m when restart is enabled. This plot is made with midpoint rule and $k = 200$ over 10 seconds.

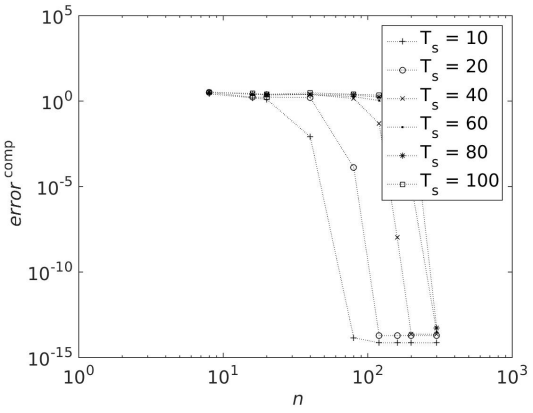
Figure 4.7 shows that if restart is enabled, a good approximation of the solution can be found with $n \geq 20$, for any m.

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

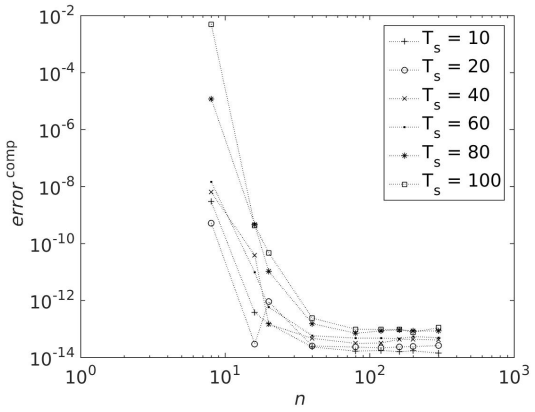
4.4.3 For time domains



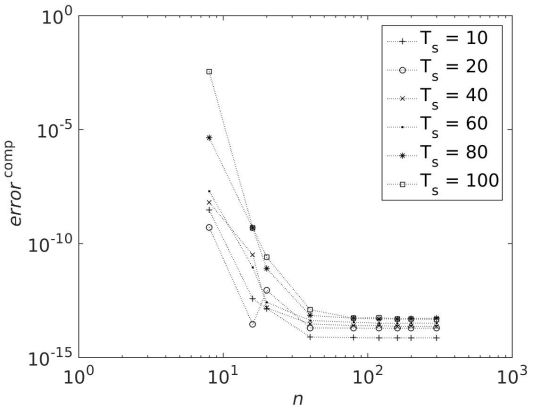
(a) Error for KPM without restart.



(b) Error for SLM without restart.



(c) Error for KPM with restart.



(d) Error for SLM with restart.

Figure 4.8: The pictures shows which n gives convergence for different T_s, with $k = 20 \cdot T_s$ and $m = 20$. The top plots are without restart, and the bottom are with restart.

Figure 4.8 shows that the length of the time domain is affecting the choice of 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.

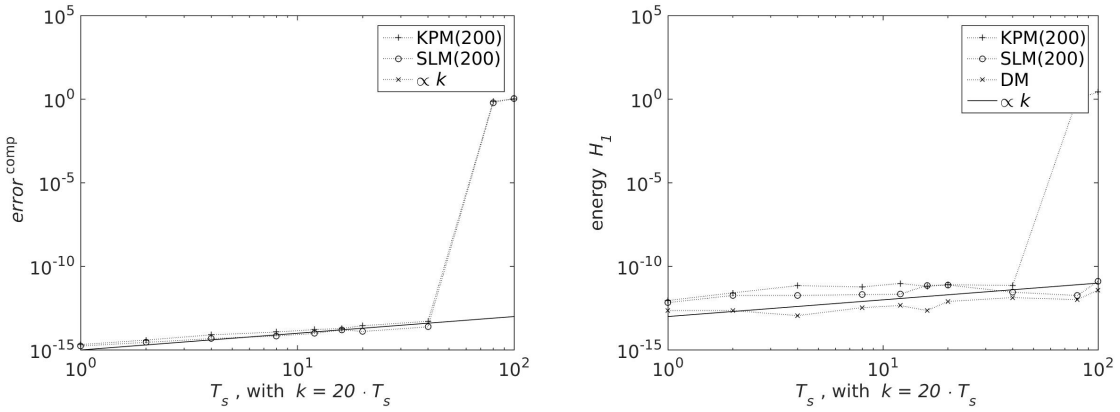
SLM and KPM behaves very similarly in all pictures shown in this section.

4.5 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 the restarting and windowing interacts with this.

4.5.1 Without restart



(a) Error.

(b) Energy.

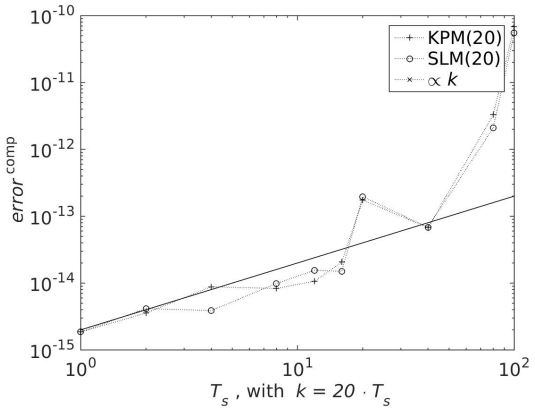
Figure 4.9: The figures show how error and energy changes with larger time domains. $n = 200$, $m = 20$, and trapezoidal rule is used.

The energy and error 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.

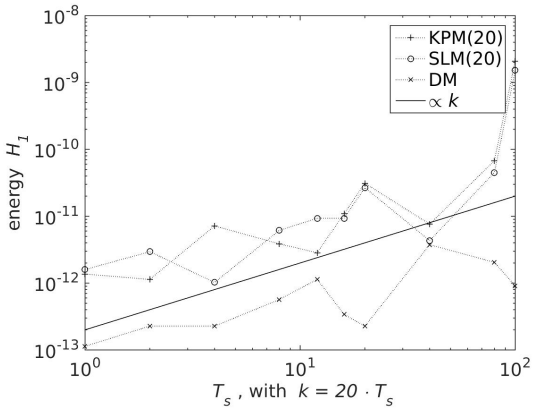
The suddenly increase in error at the last point in figure 4.9a shows the instability of the projection methods, this also happens when restart is enabled, and will be explained further in section 4.5.4. If the time domain was a little larger the projection methods would diverge.

Picture 4.9b shows that `SLM` preserves the energy, while `KPM` does not.

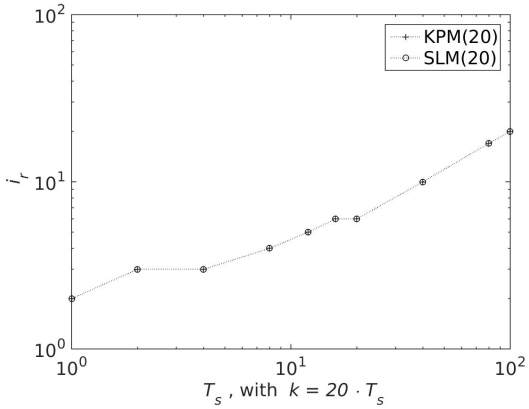
4.5.2 With restart



(a) Error.



(b) Energy.



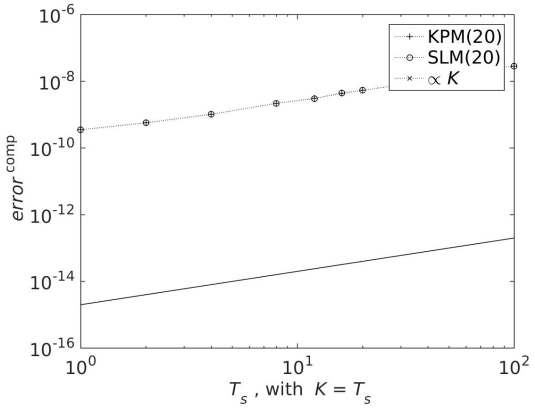
(c) Number of restarts.

Figure 4.10: The figures show how error and energy changes with larger time domains when restart is enabled with $\iota = 1e-6$. $n = 20$, $m = 20$, and midpoint rule is used.

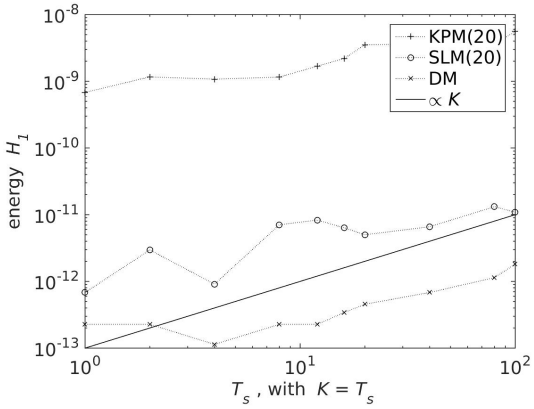
Figure 4.10 and 4.9 are quite similar, except for the last points. In this case the restarts can help with the convergence. If the last points are disregarded these methods are quite similar. Which method is most desirable will not become apparent before computation time is discussed.

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.

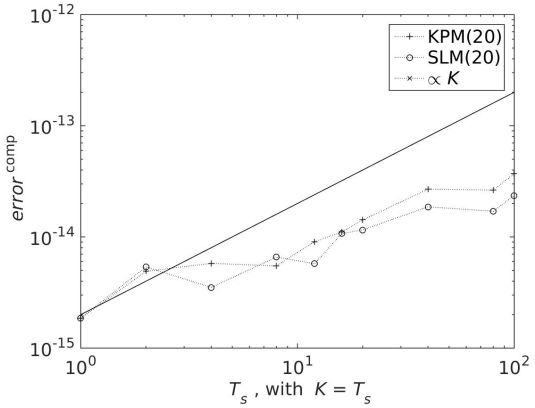
4.5.3 Windowing



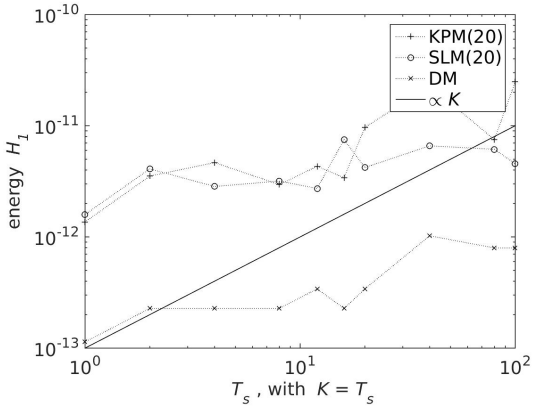
(a) Error without restart.



(b) Energy without restart.



(c) Error with restart.



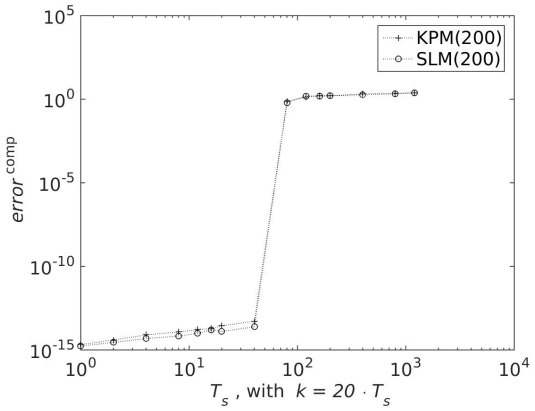
(d) Energy with restart.

Figure 4.11: The figures shows how windowing with and without restart changes the error and energy over time. $m = 20$, $k = 20$ and $\iota = 1 - 6$ when restart is enabled.

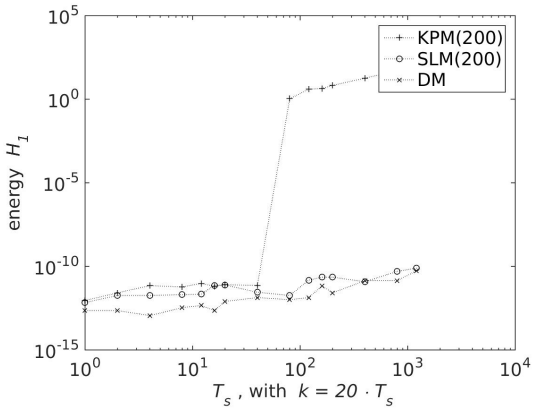
The divergence problems with the last points seams to be gone. Restarting makes the error a few orders smaller, but the growth is linear with T_s for both cases. Since it is interesting to see the difference between `SLM` and `KPM`, restarting will not be used with windowing. Both error and energy with windowing is comparable to the other cases. This makes windowing an interesting idea.

4.5.4 Very long time

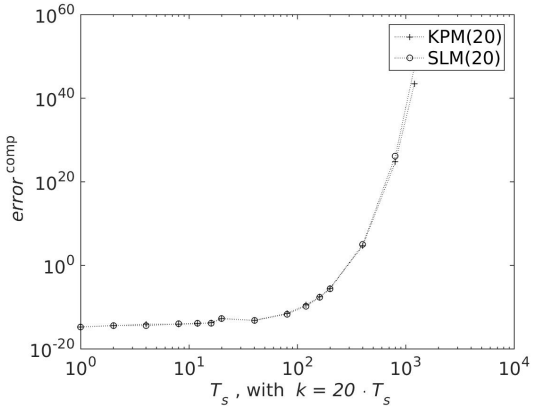
In this section it is shown what will happen when the time domain becomes to large for the methods to converge.



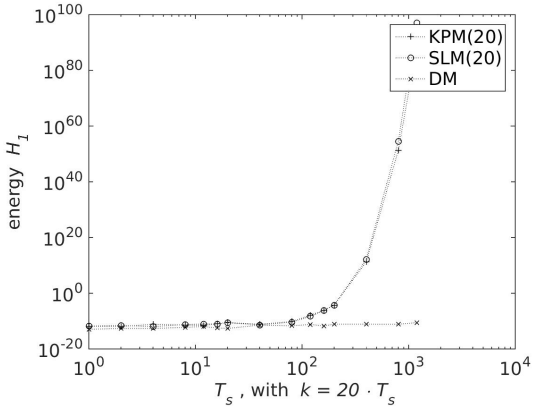
(a) Error without restart.



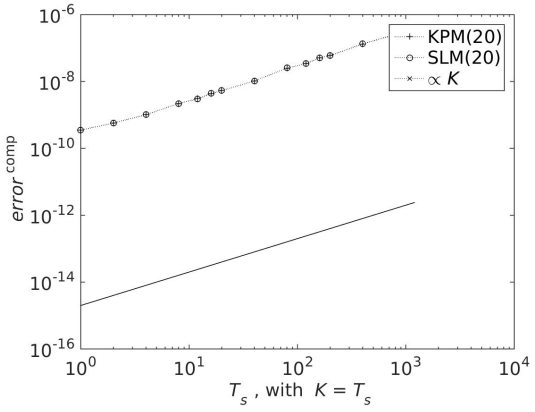
(b) Energy without restart.



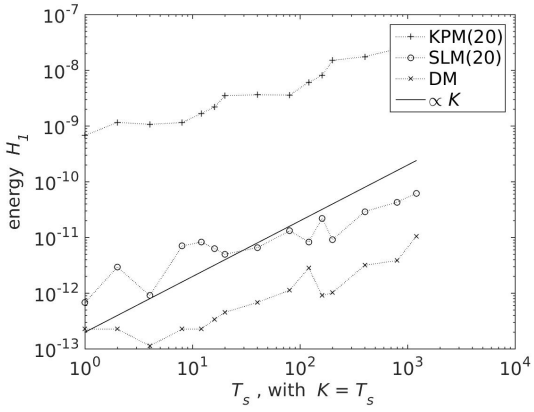
(c) Error with restart.



(d) Energy without restart.



(e) Error for windowing without restart.



(f) Energy with windowing without restart.

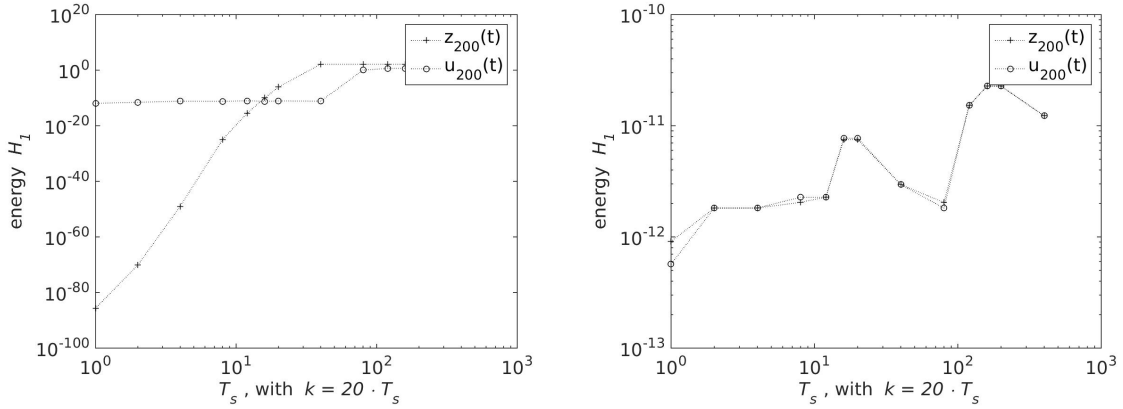
Figure 4.12: This figure shows how different methods cope with a very long time domain. Windowing does not use restart.

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. When 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.

4.5.5 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 4.13: The figures shows how the transformation between $z(t)$ and $u(t)$ changes the energy. Restart is not enabled, $m = 20$, and trapezoidal rule is used.

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 energy preserving properties of SLM. It is worth mentioning that $S_n J_{\hat{m}} S_n - J_n \sim 1e-16$ and $V_n^\top V_n - I_n \sim 1e-11$.

4.5.6 Residual energy

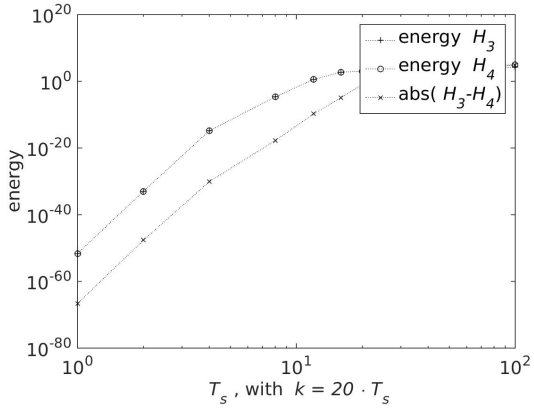


Figure 4.14: A plot of $\mathscr{H}_3$ and $\mathscr{H}_4$ for different time domains.

$\mathscr{H}_3$ 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).

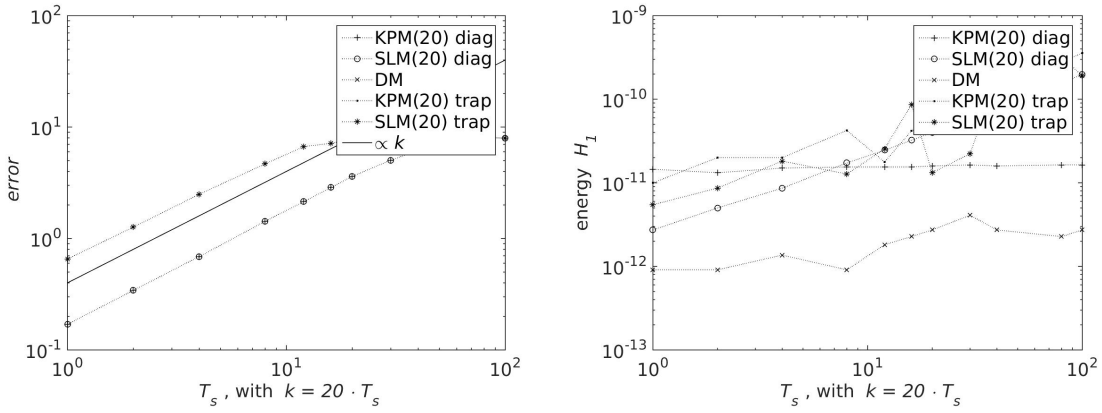
4.6 A different idea

This section will see how using an exact solver can improve 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 4.15: A figure showing how the error and energy changes when an exact solver (`diag`) is used. $n = 20$, $m = 20$, restart is not enabled. `DM` uses trapezoidal rule.

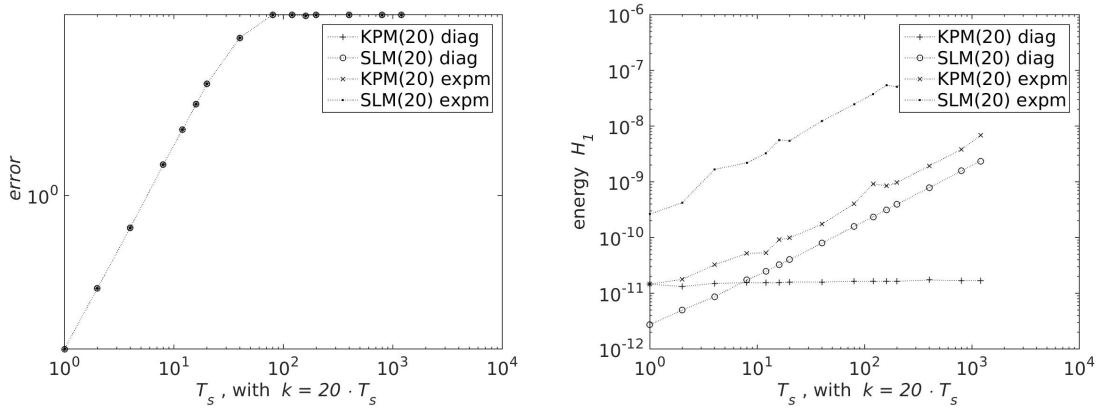
The error for the projection methods with an exact solver is smaller than the error for `DM` with trapezoidal rule.

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` the energy is constant. It seams that `KPM` is better than `SLM` on long time domains in this setting.

The pictures also show that all errors are increasing linearly.

4.6.1 Matlabs `expm` function

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



(a) Error. (b) Energy.

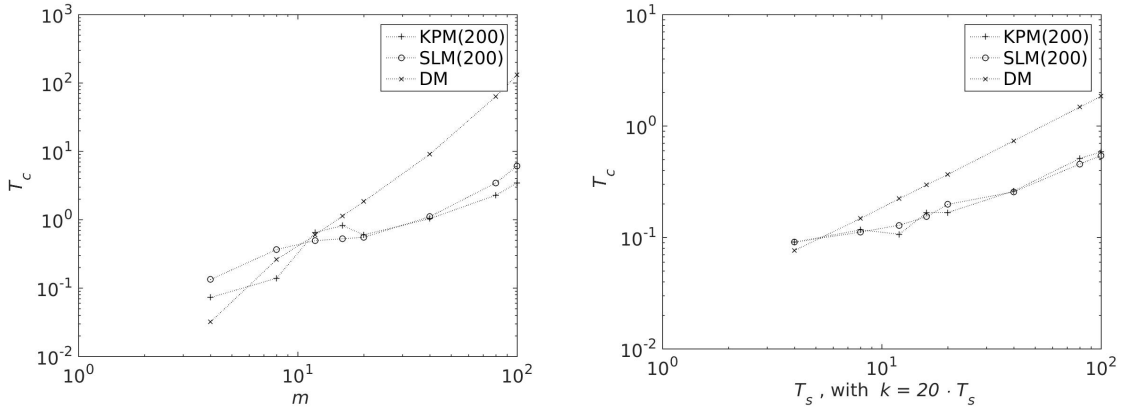
Figure 4.16: A figure showing the difference in error and energy for the different exact integration methods. Restart is not enabled, $m = 20$.

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

4.7 Computation time

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

4.7.1 Without restart



(a) Computation time as a function of m.

(b) Computation time as a function of k.

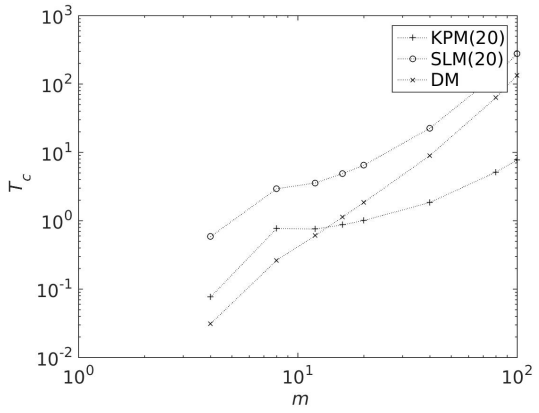
Figure 4.17: A figure of the computation times without restart. $n = 200$, $T_s = 100$, $k = 2000$, $m = 20$ unless stated.

All computation times scales linearly with k, but `SLM` and `KPM` require much less computation time than `DM`.

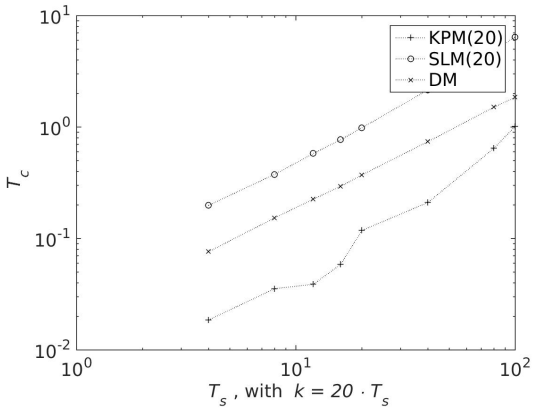
The scaling with m is a little more difficult, but `DM` seams to be increasing quadratically, while `SLM` and `KPM` increases closer to linear. Though the last points of the projection methods are suggesting a faster increase. `SLM` and `KPM` has very similar computation times.

4.7.2 With restart

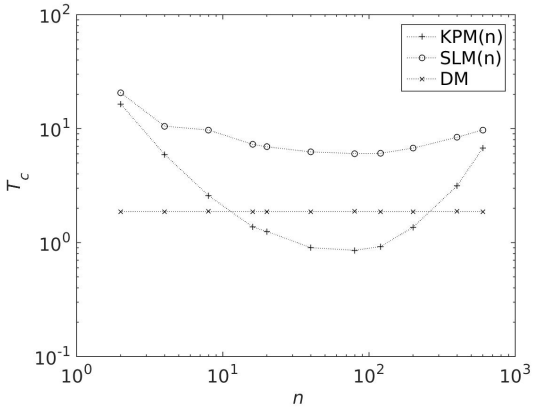
Since n can be chosen freely above a certain threshold, but smaller n might give higher i_r, it is interesting to see how computation time changes as a function of n. When restart is not enabled it is quite obvious that smaller n gives smaller computation times.



(a) Computation time as a function of m.



(b) Computation time as a function of k.



(c) Computation time as a function of n.

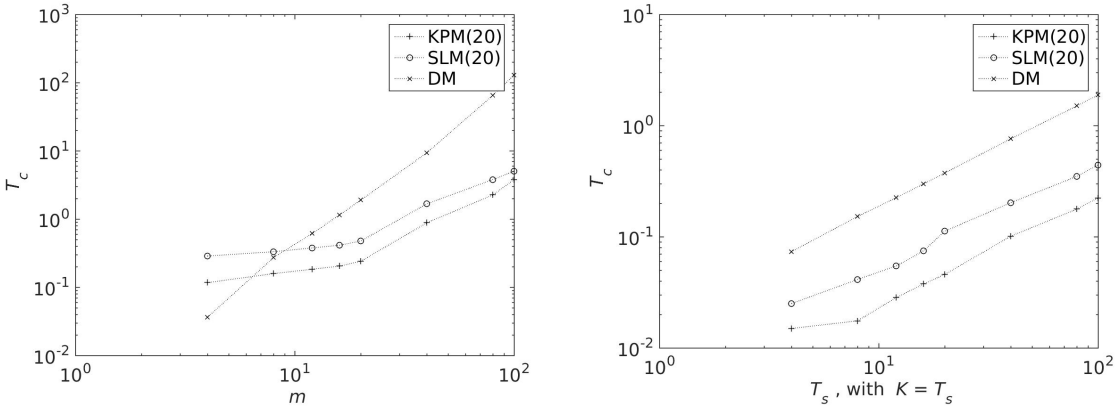
Figure 4.18: The top pictures show the computation times with restart. The bottom picture shows how computation time changes as a function of n. $n = 20$, $T_s = 100$, $k = 2000$, and $m = 20$ unless stated.

SLM and DM is now increasing equally fast with m, while KPM increases about the same as without restart. In this case DM would be a better choice than SLM, while KPM at least gives a smaller computation time.

Computation time as a function of k increases faster for KPM than for the other methods. This is because KPM needs more restarts on larger time domains, and makes KPM useful when m is large and k is small. For SLM this effect is not visible.

Figure 4.18c shows that computation times are depending on n. For KPM this effect is very important.

4.7.3 Windowing

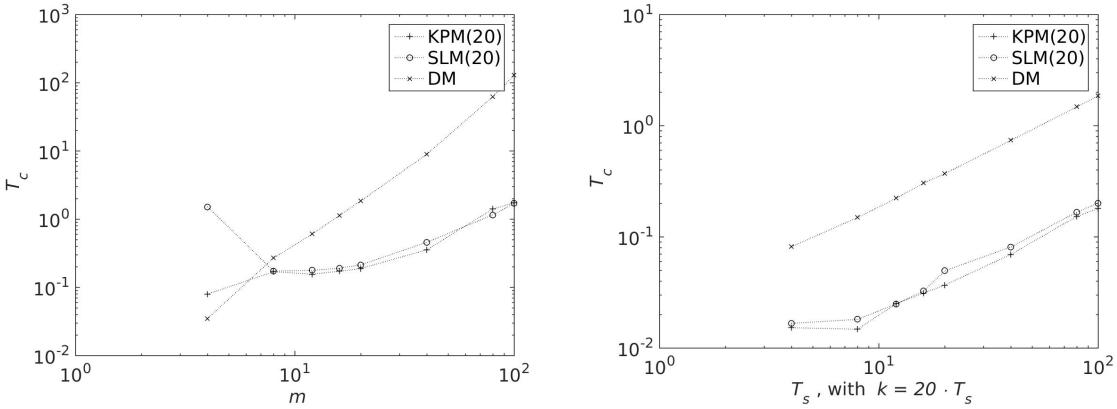


(a) Computation time as a function of m. (b) Computation time as a function of k.

Figure 4.19: A figure of the computation times with restart. $n = 20$, $T_s = 100$, $k = 20$ per second, and $m = 20$ unless stated.

Windowing gives the best performance for `SLM` and `KPM` till now. It is interesting that this method gives the best estimates for error and error, and the fastest computation time. The reason for the faster computation time is probably the small n without the need to restart.

4.7.4 With `diag`



(a) Computation time as a function of m. (b) Computation time as a function of k.

Figure 4.20: A figure of the computation times with `diag`. $n = 20$, $T_s = 100$, $k = 2000$, $m = 20$ unless stated.

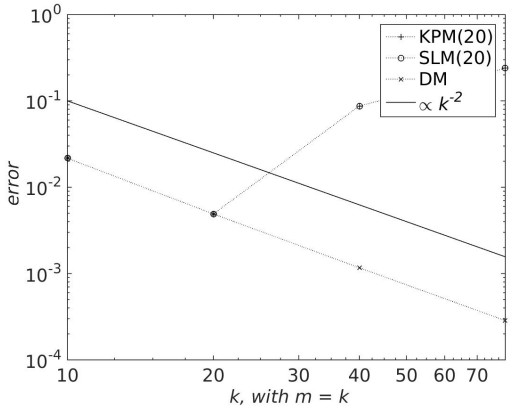
Figure 4.20 shows that the exact solver without restart is even faster than windowing. This might be due to a artificially small n, due to the unconditional convergence with `wave`.

Chapter 5

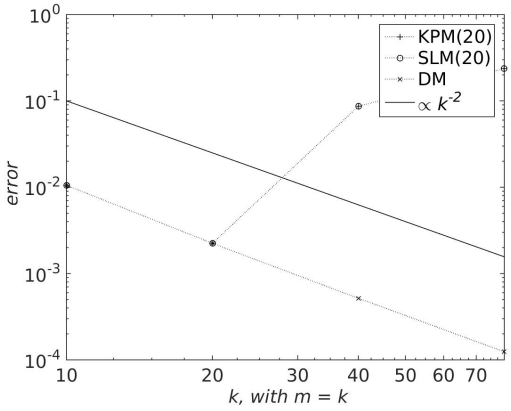
Results for test problems with varying energy

This chapter will look at many of the same elements that was discussed in section 4. Some results are similar, in this case results may not be shown here. 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 when the energy is varying, thus there will be no discussion about that in this chapter.

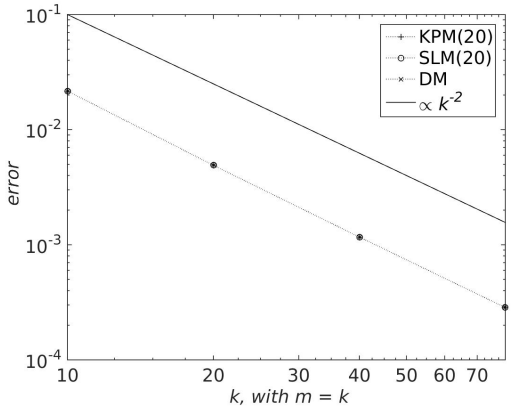
5.1 Convergence



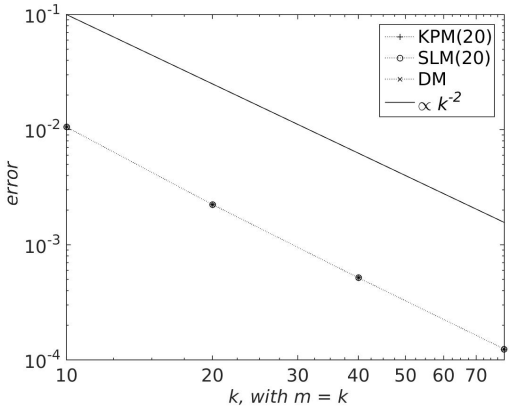
(a) Convergence for `wave` with trapezoidal rule without restart.



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



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



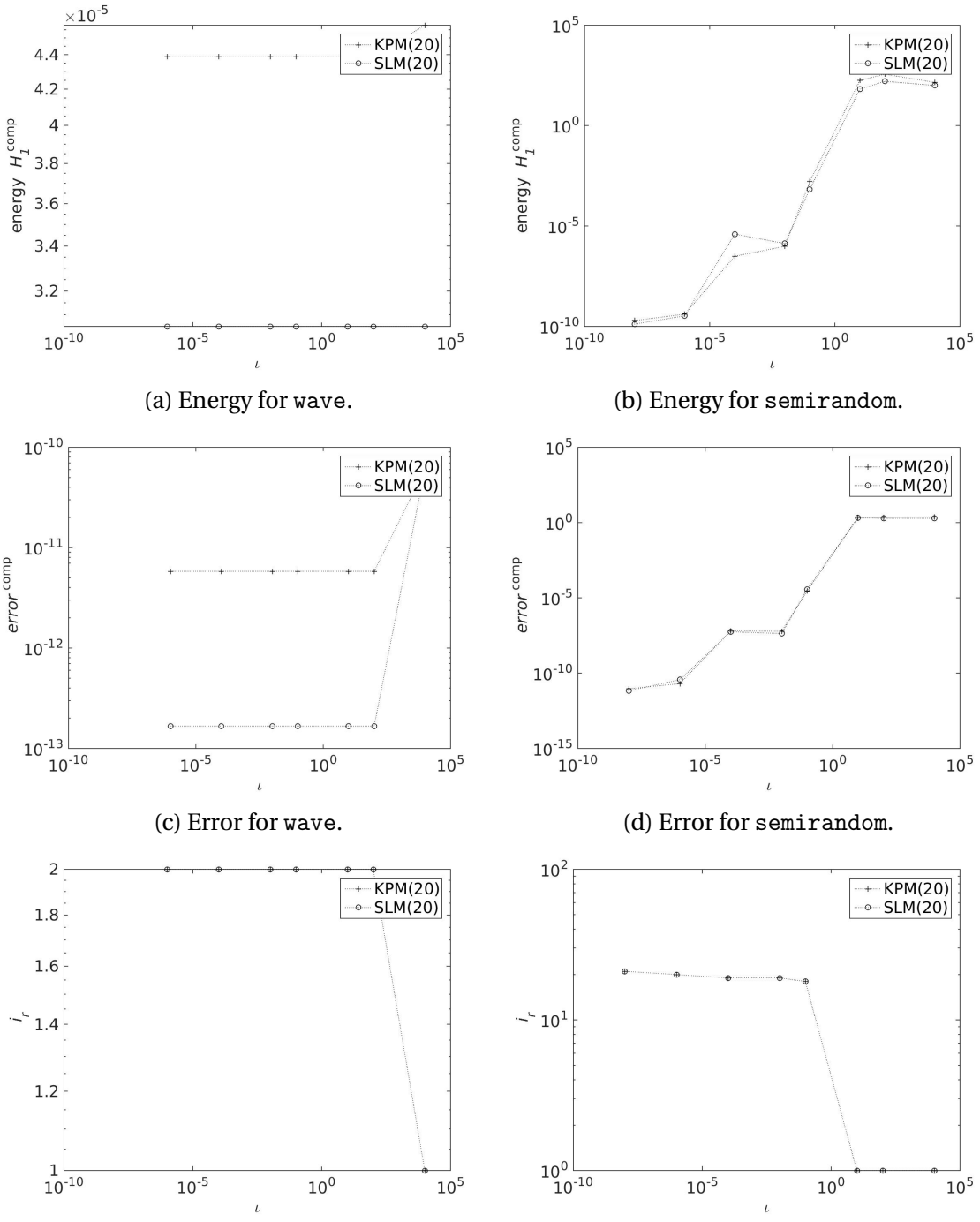
(d) Convergence for `wave` with midpoint rule rule with restart.

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) Energy for `semirandom`.

(c) Error for `wave`.

(d) Error for `semirandom`.

(e) Number of restarts for `wave`

(f) Number of restarts for `semirandom`.

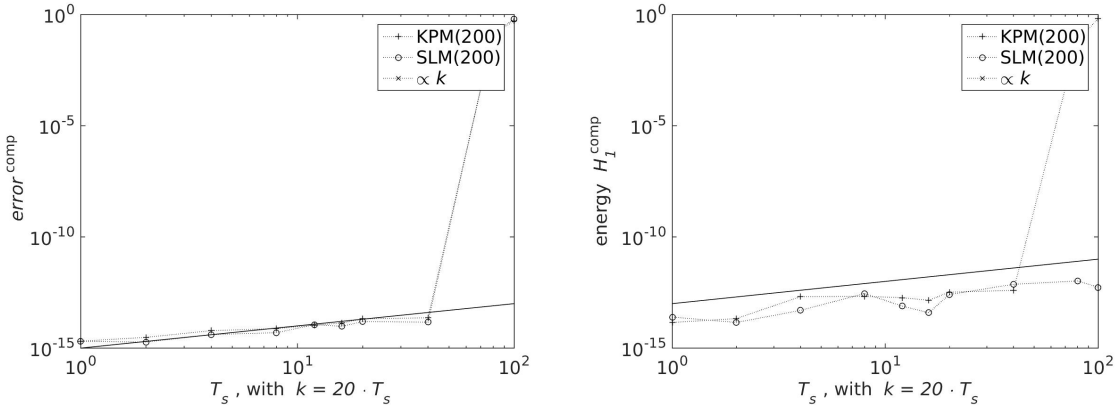
Figure 5.2: These figures show how restarting can improve the solution. The pictures on the left is for `wave` and one the right is for `semirandom`. This plot considers 100 seconds, with $n = m = 20$, $k = 2000$, $m = 20$, and midpoint rule.

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 $1e-15$. The difference between SLM and KPM is definitely smaller in this case.

5.3 Energy and error

The dependance between n is the same as in section 4.4, thus n will be kept at the same value it was. The results in this section is very similar to the results in section 4.5. Results not shown are to similar to show.

5.3.1 Without restart



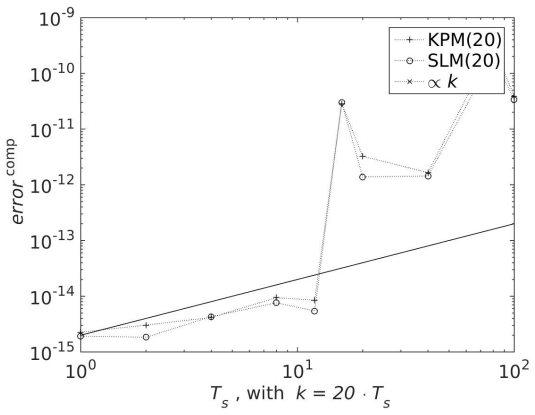
(a) Error.

(b) Energy.

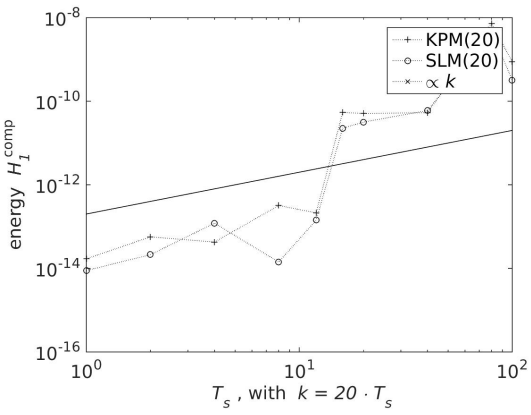
Figure 5.3: The change in error and energy as a function of time. $m = 20$, restart is not enabled.

Figure 5.3 is very similar to Figure 4.9. The only difference is the more powerful divergence occurring at the last points. SLM manages to maintain a good approximation of the energy throughout the entire time domain. This shows that SLM may still be useful in this setting.

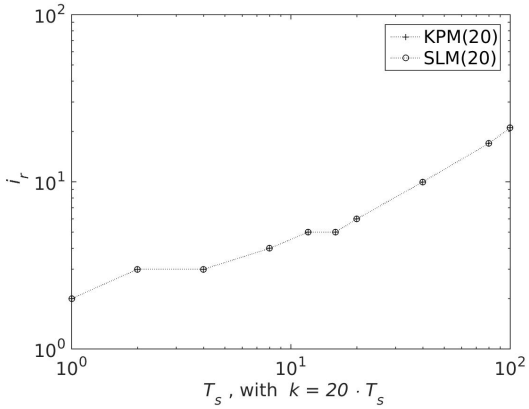
5.3.2 With restart



(a) Error.



(b) Energy.

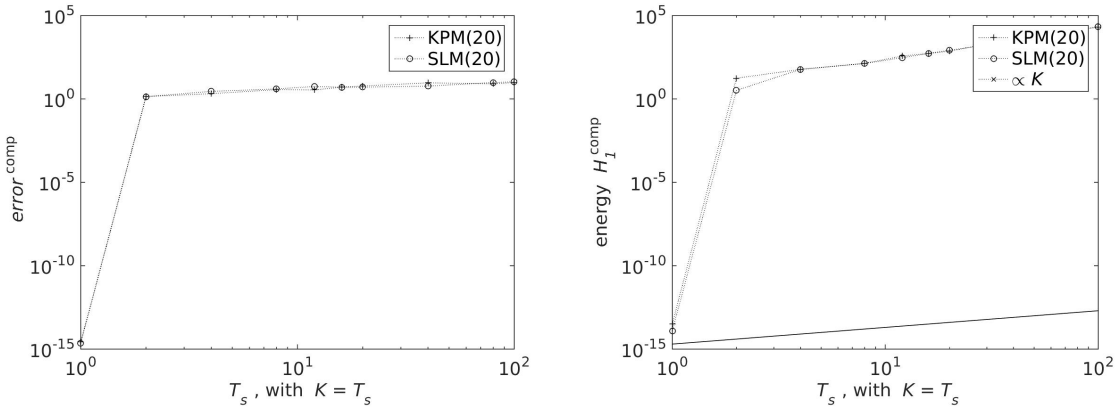


(c) Number of restarts.

Figure 5.4: The top pictures shows error and energy, while the bottom picture shows the number of iterations. $m = 20$, restart is enabled with $\iota = 1e-6$.

The restart makes `SLM` and `KPM` behave very similar. No divergence occurs on this small time interval. Restarting is a better idea when the energy is varying than when the energy is constant. These figures are very comparable to Figure 4.10. Even thoug it is not shown here the methods diverges on time domains

5.3.3 Windowing



(a) Error with restart.

(b) Energy with restart.

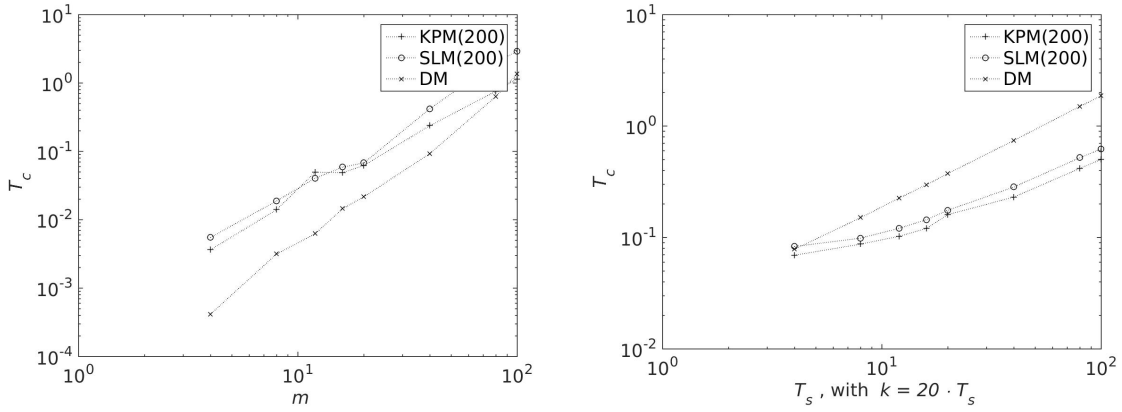
Figure 5.5: Windowing with $m = 20$, $k = 20$. The restart does not change the result.

Figure 5.5 shows that windowing does not work with varying energy, thus a very promising solution strategy from the previous chapter has a limiting factor.

5.4 Computation time

Computation times are fairly different due to extra difficulties with convergence.

5.4.1 Without restart



(a) Computation time as a function of m.

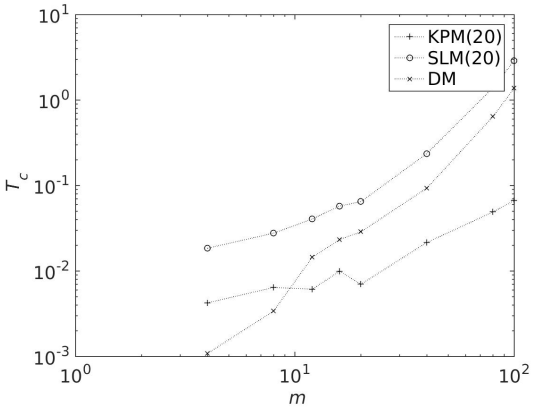
(b) Computation time as a function of k.

Figure 5.6: A figure of the computation times without restart. $n = 200$, $T_s = 100$, $k = 2000$, and $m = 20$ unless stated.

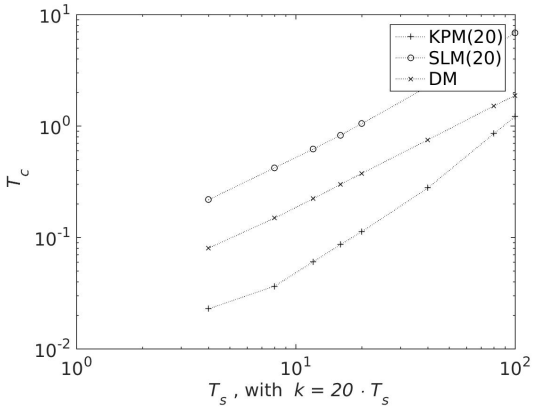
The difference between DM and the projection methods are a lot smaller in this case. At the last point on figure 5.6, DM and KPM are intersecting. KPM becomes faster than DM after this, but SLM does not overtake DM.

These results are a little surprising since the numbers are the same as in section 4.7, and there is no restarting.

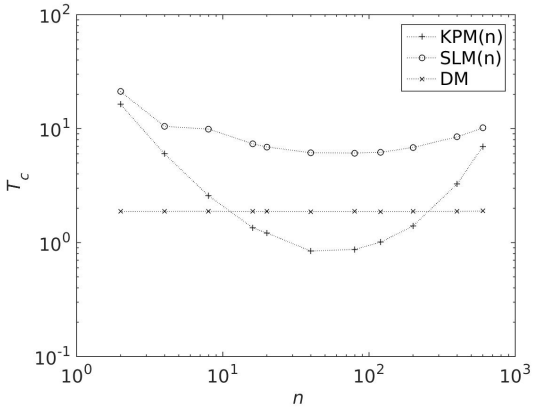
5.4.2 With restart



(a) Computation time as a function of m.



(b) Computation time as a function of k.



(c) Computation time as a function of n.

Figure 5.7: A figure of the computation times with restart. $n = 20$, $T_s = 100$, $k = 2000$, and $m = 20$ unless stated.

Figure 5.7 shows that it can be a good idea to use KPM when m is large and k is small. The faster increase with k for KPM comes from extra number of restarts that KPM needs to ensure convergence. The relation between n and computation time shows that KPM has the possibility if being alot [12] faster than DM, SLM does not have this possibility. This makes SLM consistently solver than DM.

Chapter 6

6.1 Code

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

6.2 Further work

The implemented test problems has some limitations. It could be interesting to see how the error and energy behaved with a random Hamiltonian matrix, with a known analytical solution, specifically `expm` with restart, and the relation between n and m.

Graphics cards are designed to solve small matrices in parallel [20]. When the energy is constant windowing can be used to ensure convergence with this n. This could then be used to solve non linear Hamiltonian problems in parallel under optimized conditions.

6.3 Conclusion

Since the results are divided in two cases it feels natural to divide the conclusion in the same two case.

6.3.1 Constant energy

The error for both KPM and SLM is found to increase linearly within a reasonable time domain with suitable n. With large time domains or 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, 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 method. 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 with restart, or with a larger n. If restart is used SLM looses its energy preserving property, and behaves very similar 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 has only 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 can definitely be faster than this if n is chosen optimal.

If an exact solver is used it is possible to achieve better accuracy with smaller computation time with the projection methods than with trapezoidal rule. The energy for SLM increases linearly in this case, and is constant for KPM.
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 with

large time domains can be solved with windowing, which also makes computations faster. Using an exact solver can also increase the accuracy, without any downsides.

6.3.2 Varying energy

In this case SLM's energy preserving properties does not work, making the difference between SLM and KPM even smaller. Windowing is also not working, along with the exact solvers, thus several of the important reasons to use the projection methods are removed. Even with this, both SLM and KPM manages to get error and energy close to DM on small time domains. The restriction on small time domains might not be to big since the linearly increasing error of DM will make any approximation useless on time domains over a certain size.

Suitable n is found to be depending linearly with the length of the time domain.

Convergence is a lot harder for the projection methods. This means that either a larger n, or more restarts are necessary to achieve convergence. This results in longer computation times, making SLM consistently worse 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.

Bibliography

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

[2] Elena Celledoni. https://www.ntnu.no/ansatte/elena.celledoni, february 2016.

[3] 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*.

[4] Archbishop Richard Bancroft. *King James Bible*. King's Printer Robert Barker, 1604.

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

[7] 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.

[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 (2011) 578–600*, page 579.

[9] 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.

[10] 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.

[11] 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.

[12] Allie Brosh. The Alot is Better Than You at Everything. http://hyperboleandahalf.blogspot.no/2010/04/alot-is-better-than-you-at-everything.html, April 2010.

[13] 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.

[14] 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.

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

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

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

[18] HEIKE FASSBENDER. ERROR ANALYSIS OF THE SYMPLECTIC LANCZOS METHOD FOR THE SYMPLECTIC EIGENVALUE PROBLEM. *SIAM J. S CI. C OMPUT . Vol. 35, No. 3, pp. A1376–A1397.*

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

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

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

[22] 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.

[23] 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.

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

[25] 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 Institu te of Science, Bangalore, India-560012*, januar 2014.

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

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

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

[29] Endre Süli and David F. Mayers. *An Introduction to Numerical Analysis*, chapter 12.5 Runge–Kutta methods, page 328. cambridge university press, 2003.

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

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

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