NTNU
Department of Mathematical Sciences

Projection methods for Hamiltonian systems

Sindre Eskeland

February 3, 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. It was written during the winter 2015-2016.

Acknowledgment

Thanks to Elena Celledoni for guiding me, and Lu Li for giving me the theoretical insight this text needed. And trygve

Summary and Conclusions

To be filled in!

Contents

Chapter 1

Introduction

The equation

$$\begin{aligned} \dot{u}(t) &= Au(t) + p \cdot f(t) = g(t) \\ u(0) &= u_0 \end{aligned} \tag{1.1}$$

often makes an appearance when solving partial differential equations with numerical methods. The author has earlier observed how the heat equation, discretized with finite difference methods to be on the form of equation (1.1) can be solved with the use of the Krylov projection method(KPM) [2], which uses Arnoldi's algorithm(Arnoldi) as orthogonalisation method. This note will continue on the same track, but with more focus on Hamiltonian discretizations of wave equations, random problems, and energy preservation. It will also feature a comparison between symplectic Lanzcos method(SLM) [6] and Arnoldi. SLM is a projection technique that only works on Hamiltonian matrices. Due to this, SLM (claims to) preserve energy better than Arnoldi. This note will also compare time consumption and global error for the different methods, but will not contain much theoretical derivation. For this I recommend reading [1], [2], [6], [5], and [3].

There will also be a comparison between some different integration methods, and how the projection methods change when the time domain is divided into smaller pieces and each piece is considered independently with initial conditions from previous time domain.

!!!!!!!!!!!!!!!Skrive litt mer her om hva som skal skje!!!!!!!!!!!!!!!!!!!!!!

Chapter 2

TTD

- Jeg synes at som minimum i oppgaven din burde du klare å avgjøre om SLM med restart er energibevarende eller ikke og om den gir lineæar vekst av globalfeil eller raskere vekst.
- Skrive bilde text, og fjerne labler fra bilder!
- Er det flere figurer jeg burde ha med?
- Rekkefølgen må fikses
- Fix notasjon
- Sett parantes rundt iterasjonsvariabler på z-er, og endre z-er til ϵ og δ?
- Legg til hjelpelinjer på de plittene som trenger det, og kanskje ikke logloge alle plottene som er det
- når jeg bruker expm så må jeg gjøre det både med og uten restart.
- Har en følelse av at jeg mangler en restart et sted(et bilde).
- Lag bilder som viser feilen og energien med forskjellige n. med lang og kort tid
- Skriv mer i introduksjonskap
- Skriv om hva "constant energy" og "varying energy" betyr
- Med stor nok n så kan du tvinge SLM til å convergere ved lang tid!

- det er vist forskjell mellom energibevarende og symplektisk!!!!!!!!!!!!!
- Det må overalt stå om ting er vektor eller skalar funksjon!
- Ta bort en del ligningsnummer, stjerne alle begin equation...!!!!!!!!!!!!!!!!
- Skriv hvilken ligning som blir brukt til å beregne energi i praksis!!!!!!!!!!
- Flytte diskretisering og slikt inn sammen med integrasjonemstemder og wave
- burde jeg ha residual energy for arnoldi også?
- dele opp resultater i konstant og varierende energi
- Presantere b!
- Flytte definisjonen av symplectisk of hamiltosnk

Chapter 3

Theory

!!!!!!!!!!!Ny tekst ehr!!!!!!!!!!!!!!

There will here be a short explanation of all solvers, constants, abbreviations and expressions used in this text. MATLAB notation is used where applicable.

3.1 Hamiltonian and Symplectic

!!!!!!!!!!!!!!!!!!!!!Dette er bare dårlig!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!Skriv hva hamiltonsk og symplectisk betyr!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!Citing!!!!!!!!!!!!!!!!!!!!!!

A matrix is said to be Hamiltonian if [**?**]

$$(JA)^\top = JA. \tag{3.1}$$

A matrix is symplectic if [**?**]

$$A^\top JA = J. \tag{3.2}$$

3.2 Zero initial condition

For both KPM and SLM it is important that the initial conditions are zero(due to the restart). Equation (1.1) can be transformed so that it has zero initial conditions in the following way:

Start by shifting the solution

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

then rewrite the original equation as

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

The equation above solves the shifted problem, solve the original problem 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. The letter b will be used as a collective term to describe both Au_0, or any constant vector occurring on the right hand side of equation (1.1).

3.3 Energy

It is well known that the energy of a system on the form

$$\begin{aligned} \dot{u}(t) &= Au(t) \\ u(0) &= u_0 \end{aligned} \tag{3.3}$$

can be expressed as [**?**]

$$\mathscr{H}_1(u) = \frac{1}{2} u^\top(t) JAu(t) \tag{3.4}$$

If the transformation in section 3.2 is used, the energy is be written as

$$\mathscr{H}_2(u) = \frac{1}{2} \hat{u}^\top(t) JA\hat{u} + \hat{u}^\top(t) JAu_0. \tag{3.5}$$

For most theoretical derivation $\mathscr{H}_2$ will be used, as the shifted problem is the one actually solved. But in the figures $\mathscr{H}_1$ is used since the unshifted problem is regarded.

Trapezoidal rule [7]	$U_{i+1} = U_i + h_t g\left(\frac{1}{2}(t_i + t_{i+1}), \frac{1}{2}(U_i + U_{i+1})\right)$ $U_{i+1} = (I - \frac{Ah_t}{2})^{-1}\left(U_i + \frac{h_t}{2}\left(AU_i + (F_{i+1} + F_i)\right)\right)$
Forward Euler [8]	$U_{i+1} = U_i + h_t g(t_i, U_i)$ $U_{i+1} = U_i + h_t\left(AU_i + F_i\right)$
Midpoint rule [9]	$U_{i+1} = U_i + h_t g\left(t_{i+\frac{1}{2}}, \frac{1}{2}(U_i + U_{i+1})\right)$ $U_{i+\frac{1}{2}} = U_i + \frac{h_t}{2}(AU_i + F_{\frac{1}{2}})$ $U_{i+1} = (I - \frac{Ah_t}{2})^{-1}(U_{i+\frac{1}{2}} + \frac{h_t}{2}F_{i+\frac{1}{2}})$

Table 3.1: Methods for integrating in time. Note that since the midpoint rule uses the midpoint $F_{i+\frac{1}{2}}$, we need to save twice as many points for midpoint rule as for the other methods.

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

Table 3.2: Methods for exact integration in time. Since they are very computationally demanding they will only be used on small projected matrices. They also need F to be a constant vector, so these methods only work with constant energy.

3.4 Integration methods

The number of steps in time is denoted by k, making the step size in time, $h_t = 1/k$.

The integration 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 3.1.

Trapezoidal rule and midpoint rule is the same method if F is constant, so they are both energy conserving in this case. When F is not constant only midpoint rule is symplectic energy preserving [**?**]. Though both midpoint rule and trapezoidal rule converges quaderatically.

Forward Euler has no energy preserving properties, and is used to show the difference between a naive integration method, and an energy preserving. This method has a linear convergence rate.

In addition to the iteration schemes in table 3.1 some exact solvers are used, they are presented in table 3.2. !!!!!!!!!!!!!!!!!!!!!!!!tabl 3.2 har noen feil og mangler gange ting !!!!!!!!!!!!!!!!!!!!!!!!

3.4.1 Energy conservation for trapezoidal rule

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

$$\begin{aligned} \dot{u}(t) &= Au + b \\ u(0) &= 0. \end{aligned} \tag{3.6}$$

Note that b can be any constant vector, not just Au_0. The energy of this function has already been presented in equation (3.5). The main ingredients in this proof is 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

$$\nabla \mathscr{H}_1(u) = J(Au + b). \tag{3.7}$$

The trapezoidal rule found in table 3.1, used on equation (3.6) gives

$$\frac{U_{j+1} - U_j}{h_t} = A\frac{U_{j+1} + U_j}{2} + b. \tag{3.8}$$

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

$$\nabla \mathscr{H}_1(\frac{U_{j+1} + U_j}{2}) = JA\frac{U_{j+1} + U_j}{2} + Jb \tag{3.9}$$

Since

$$\frac{U_{j+1} - U_j}{h_t} = J^{-1}\nabla \mathscr{H}_1(\frac{U_{j+1} + U_j}{2}) \tag{3.10}$$

and that

$$\nabla \mathscr{H}_1(\frac{U_{j+1} + U_j}{2})^\top J^{-1}\nabla \mathscr{H}_1(\frac{U_{j+1} + U_j}{2}) = 0 \tag{3.11}$$

we have

$$\nabla \mathscr{H}_1(\frac{U_{j+1} + U_j}{2})^\top \frac{U_{j+1} - U_j}{h_t} = 0 \tag{3.12}$$

Substituting for $\nabla \mathscr{H}_1(\frac{U_{j+1}+U_j}{2})$, and solving the parenthesis gives

$$\mathscr{H}_1(U_{j+1}) - \mathscr{H}_1(U_j) = 0 \tag{3.13}$$

So trapezoidal rule conserves the energy. Since the initial conditions are satisfied, the energy will have the correct value at all times.

3.5 Dividing the time domain

!!!!!!!!!Forklar hvorfor dette er noe vil vil gjøre(jeg forstår det ikke nå)!!!!!!!!!

Since SLM diverges when restart is enabled, and it integrates over long time

SLM has a tendency to diverge when the time domain is big, and restart is enabled. A fix to this might be to divide the time domain in smaller pieces, and solve each piece individually.

Let T_s denote simulated time, let K be the number of pieces T_s is divide into, and let k be the number of pieces each K is divided into. The procedure is to solve each of the K intervals as separate problems, with the initial conditions updated. This is explained in a more precise manner in algorithm 1.

Algorithm 1 !!!!!!!Spør elena om dette har et fint navn jeg kan bruke!!!!

Start with an initial value U_0, K and k.
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.

3.6 Abbriviations

!!!!!!!!!!!!!!!!!!!Denne seksjonen er ubrukelig!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!dette kan være en innledende tabell eller noe slik, men tingenen må fortsatt presenteres i teksten, den er også nokså mangelfll!!!!!!!!!!!!!!!!!!!!1

Table 3.3 contains an explanation of the expressions you will see on figures later.

r_n	Number of restarts performed by Arnoldi or symplectic Lanczos method.
T_c	Computation time used to solve a problem
er_1	Difference between analytical solution, and orthogonalised solution.
en_1	Difference in energy between analytical solution, and orthogonalised solution.
er_2	Difference between orthogonalised solution, and the non-orthogonalised solution.
en_2	Difference in energy between orthogonalised solution, and the non-orthogonalised solution.
m	Number of point in each spacial direction
n	Size of orthogonal space, also called restart variable
k	Number of points in time
T_s	Simulated time
`restart`	A boolean value. If `restart` == 1, Arnoldi or symplectic Lanczos method will restart.
ϵ	If `restart` is true, restarting will commence until the change in the solution is less than ϵ

Table 3.3: Explanation of abbreviations.

3.7 Solution methods

!!!!!!!!!!!!!!!!!!!!Forklar iterasjonsvariablen i!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

There are two orthogonalisation methods that will be discussed in this text. Their names are symplectic Lanczos method, and Arnoldi's algorithm. This section will only contain some of the key points that makes these algorithms work.

!!!!!!!!!!Skriv hva projeksjonemetoder er!!!!!!!!!!!!!!

Projection methods are ways to make an approximated solution 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, the drawback is that the solutions are approximated and that the orthogonal system needs to be found, and this can be time consuming. The approximated solution can be improved by restarting, this means using the projection method again, and solve an equation for the difference between the projected solution, and the true solution repeatedly.

!!!!!!!!!!!!!!!!! forkalr restart bedre?!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Assume that the equations are on the form

$$\begin{aligned} \dot{u}(t) &= Au(t) + v \cdot f(t) \\ u(0) &= 0 \end{aligned} \tag{3.14}$$

when these methods are used, note that the initial values are zero. It is also important that A is a Hamiltonian matrix when using SLM.

Note that the relation between $\hat{m}$, $\tilde{n}$, $\tilde{m}$ used in the algorithms is given by $\hat{m} = 2\tilde{m} = 2(m-2)^2$ and $n = 2\tilde{n}$. Don't worry to much about these details, it is just a way to simplify the expressions.

3.7.1 Arnoldi's Algorithm and the Krylov projection method

This section is loosely based around the derivation of the method done in [**?**].

The Krylov subspace is the space $W_n(A, v) = \{v, Av, \cdots, A^{n-1}v\} = \{v_1, v_2, \cdots, v_n\}$, where $n \leq \hat{m}$. The vectors v_i together with $h_{i,j} = v_i^\top A v_j$, are found by using 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 [**?**]

$$\begin{aligned} AV_n &= V_n H_n + h_{n+1,n} v_{n+1} e_n^\top \\ V_n^\top A V_n &= H_n \\ V_n^\top V_n &= I_n. \end{aligned} \tag{3.15}$$

Here, e_n is the nth canonical vector in $\mathbb{R}^n$. n is the number of iterations Arnoldi ran, also called the restart variable.

By using the transformation $u(t) \approx V_n z_n(t)$, equation (3.14) can, with the help of equation (3.15) be written as

Algorithm 2 Arnoldi's algorithm[10]

Start with $A \in \mathbb{R}^{\hat{m}\times\hat{m}}$, $v \in \mathbb{R}^{\hat{m}}$, $n \in \mathbb{N}$ and $\iota \in \mathbb{R}$.
$v_1 = v/\|v\|_2$
for $j = 1, 2, \cdots, n$ **do**
  Compute $h_{i,j} = v_i^\top A v_j, v_i$ for $i = 1, 2, \cdots, j$
  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}, $h_{n+1,n}$.

$$\begin{aligned} \dot{z}_n(t) &= H_n z_n(t) + \|v\|_2 e_1 f(t) \\ z_n(0) &= 0. \end{aligned} \tag{3.16}$$

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

$$u_n(t) = V_n z_n(t), \tag{3.17}$$

where u_n is the (approximated) solution found with n as a restart variable by this method.

The residual of the method can be written as

$$r_n(t) = v f(t) - \dot{u}_n(t) + A u_n(t) \tag{3.18}$$

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

$$r_n(t) = v f(t) - V_n \dot{z}_n(t) + A V_n z_n(t) \tag{3.19}$$

By equation (3.15) it can be simplified to

$$r_n(t) = h_{n+1,n} e_n^\top z(t) v_{n+1} \tag{3.20}$$

Since $h_{n+1,n}$ is zero for some $n \leq \hat{m}$, the procedure will converge toward the correct solution $u(t)$.

So larger n gives a better approximation of the solution, but larger n also gives higher computational complexity. The drawback is also that there is no way of knowing in advance how well the approximation will be, but 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. If the approximation is not sufficient the solution must be recalculated with larger n, unless you perform a restart.

!!ARG, iterasjonsvariablene eer drepen!!!!!!!!!!!!!!!

A restart considers the difference $\epsilon_n^{(i)}(t) = u(t) - u_n^{(i)}$, as in equation (3.21). The iteration variable i is present since it is possible to restart several times, $u_n^{(i)}$ is the solution obtained after i restarts with n as a restart variable.

$$\dot{\epsilon}_n^{(i)}(t) = A\epsilon_n^{(i)}(t) - r_n^{(i)} \tag{3.21}$$

where

$$r_n^{(i)} = h_{n+1,n}^{(i-1)} v_{n+1}^{(i-1)} e_n^\top \epsilon_n^{(i-1)}(t) \tag{3.22}$$

This es exactly the same as in equation (3.20), except that $z(t)$ is replaced with $\epsilon(t)$, and the counter i is present.

Equation (3.21) can be simplified by using equation (3.15), and writing $\epsilon_n^{(i)}(t) = V_n \delta_n^{(i)}(t)$, to obtain equation (3.23).

$$\dot{\delta}_n^{(i)}(t) = H_n^{(i)} \delta_n^{(i)}(t) + e_1 h_{n+1,n}^{(i-1)} e_n^\top \delta_n^{(i-1)}(t), \quad i \geq 1 \tag{3.23}$$

The solution is found by $u_n^{(i)}(t) = \sum_{j=0}^{i} S_n^{(j)} \delta_n^{(i)}(t)$, where $\delta_n^{(0)}(t) = z_n(t)$ and found by equation (3.26).

Repeatedly solving this equation 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 the last.

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

!!!!!!!!!!!!!!!!!!!!!!!!!!!!Medf true solution så mener vi bare bruke trap!!!!!!!!!!!!!!!!!

3.7.2 Symplectic Lanczos method

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!Kommentere at metodenen er kjempe like, bare ortalgo er forskjellig!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!Mangler flere tellevariabler(både her og andre steder)!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!mangler e_1 noen steder!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!Mangler også tekst en del steder!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!SKriv forenklingen som LULI kom med!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!Arg, jeg skal sikkert gange med J overalt!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!Sjekk dimensjoner på alt jeg skriver om her!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!Teksten her må ses i sammenhenge med teksten i Arnoldi!!!!!!!!!!!!!!!!!
SLM and KPM are very similar, which is easy to see when comparing equation (3.15) and (3.24). The main difference is that orthonormality in Arnoldi is replaced by symplecticity in SLM. This makes the derivation quite similar.

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

$$\begin{aligned} AS_n &= S_n H_n + \zeta_{n+1} v_{n+1} e_{\hat{m}}^\top \\ J_n^{-1} S_n^\top J_{\hat{m}} A S_n &= H_n \\ S_n^\top J_{\hat{m}} S_n &= J_n \end{aligned} \tag{3.24}$$

!!Endre navnene på variablene!!!!!!!!!!!!!

Here S_n is an $\hat{m} \times n$ matrix, H_n is an $n \times n$ matrix, where $\frac{n}{2}$ is the number of iterations the algorithm performed, this is because the algorithm makes two vectors per iterations, v_j and w_j.

The process of making the vectors and matrix is a little more involved than for Arnoldi's al-

Algorithm 3 Symplectic Lanczos method [6], with reortogonalization from [5].

Start with a Hamiltonian matrix $A \in \mathbb{R}^{2\tilde{m}\times 2\tilde{m}}$, $\tilde{v}_1 \in \mathbb{R}^{2\tilde{m}}$, $\tilde{n} \in \mathbb{N}$
$v_0 = 0 \in \mathbb{R}^{2\tilde{m}}$
$\zeta_1 = \|\tilde{v}_1\|_2$
$v_1 = \frac{1}{\zeta_1}\tilde{v}_1$
for $j = 1, 2, \cdots, \tilde{n}$ **do**
  $v = Av_j$
  $\delta_j = v_j^\top v$
  $\tilde{w} = v - \delta_j v_j$
  $\kappa_j = v_j^\top J_{\tilde{m}} v$
  $w_j = \frac{1}{\kappa_j}\tilde{w}_j$
  $w = Aw^j$
  $\tilde{S}_{j-1} = [v_1, v_2, \cdots, v_{j-1}, w_1, w_2, \cdots, w_{j-1}]$
  $w_j = w_j + \tilde{S}_{j-1} J_{j-1} \tilde{S}_{j-1}^\top J_{\tilde{m}} w_j$
  $\beta = -w_j^\top J_{\tilde{m}} w$
  $\tilde{v}_{j+1} = w - \zeta_j v_{j-1} - \beta_j v_j + \delta_j v_j$
  $\zeta_{j+1} = \|\tilde{v}_{j+1}\|_2$
  $v_{j+1} = \frac{1}{\zeta_{j+1}}\tilde{v}_{j+1}$
  $\tilde{S}_j = [v_1, v_2, \cdots, v_j, w_1, w_2, \cdots, w_j]$
  $v_{j+1} = v_{j+1} + \tilde{S}_j J_j \tilde{S}_j^\top J_{\tilde{m}} v_{j+1}$
end for
$S_n = [v_1, v_2, \cdots, v_{\tilde{n}}, w_1, w_2, \cdots, w_{\tilde{n}}]$
$$H_n = \begin{bmatrix} \mathrm{diag}\left([\delta_j]_{j=1}^n\right) & \mathrm{tridiag}\left([\zeta_j]_{j=2}^n, [\beta_j]_{j=1}^n, [\zeta_j]_{j=2}^n\right) \\ \mathrm{diag}\left([\kappa_j]_{j=1}^n\right) & \mathrm{diag}\left([-\delta_j]_{j=1}^n\right) \end{bmatrix}$$
Return H_n, S_n, v_{n+1}, ζ_{n+1}.

gorithm, so there will be no thorough explanation of how it works, except for in Algorithm 3.

Transform the problem in equation (3.14) with $u(t) \approx S_n z_n(t)$, multiply with $J_n^{-1} S_n^\top J_{\hat{m}}$, and simplify with equation (3.24) to obtain

$$\dot{z} = H_n z_n(t) + J_n^{-1} S_n^\top J_{\hat{m}} v f(t) \tag{3.25}$$

This can be further simplified when writing $v = S_n e_1 \| v \|_2$, and using equation (3.24), to obtain

$$\dot{z} = H_n z_n(t) + \| v \|_2 e_1 f(t). \tag{3.26}$$

Equation (3.26) and (3.16) are identical, except the orthogonalization method used.

The residual of the method can be written as

$$r_n(t) = v f(t) - \dot{u}_n(t) A u_n(t) \tag{3.27}$$

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

$$r_n(t) = v f(t) - S_n \dot{z}_n(t) + A S_n z_n(t) \tag{3.28}$$

By equation (3.24) it becomes

$$r_n(t) = \zeta_{n+1} v_{n+1} e_{\hat{m}}^\top z(t) \tag{3.29}$$

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

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 $\epsilon_n = u(t) - S_n z_n(t)$. The result is given in equation (3.33).

$$\dot{\epsilon}_n^{(i)}(t) = A \epsilon_n^{(i)}(t) + r_n^{(i)}(t) \tag{3.30}$$

where

$$r_n^{(i)}(t) = \zeta_{n+1}^{(i-1)} v_{n+1}^{(i-1)} e_{\hat{m}}^\top \epsilon_n^{(i-1)}(t) \tag{3.31}$$

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

$$\dot{\delta}_n^{(i)} = H_n^{(i)} \delta_n^{(i)} + J_n^{-1} {S_n^{(i)}}^\top J_{\hat{m}} \zeta_{n+1}^{(i-1)} v_{n+1}^{(i-1)} e_n^\top \delta_n^{(i-1)} \tag{3.32}$$

This can be further simplified to

$$\dot{\delta}_n^{(i)} = H_n^{(i)} \delta_n^{(i)} + e_1 \zeta_{n+1}^{(i-1)} e_n^\top \delta_n^{(i-1)}, \quad i \geq 1 \tag{3.33}$$

The solution is found by $u_n^{(i)}(t) = \sum_{j=0}^{i} S_n^{(j)} \delta_n^{(i)}(t)$, where $\delta_n^{(0)}(t) = z_n(t)$ and found by equation (3.26).

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

Proof that SLM without restart is energy preserving

!!!!!!!!!!!!!!!Skriv litt nøyere her!!!!!!!!!!!!!!

If equation (3.26) is solved by an energy preserving method, eg. trapezoidal rule, the energy will be preserved. The energy of this equation is

$$\mathscr{H}(z_n) = \frac{1}{2} z_n(t)^\top J_n H_n z_n(t) + z_n(t)^t op J_n e_1 \|b\|_2 \tag{3.34}$$

While the energy of the original problem is

$$\mathscr{H}(u_n) = \frac{1}{2} u_n(t)^\top J A u_n(t) + u_n(t)^\top J b \tag{3.35}$$

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

$$\mathscr{H}(u_n) = \frac{1}{2} z_n(t)^\top S_n^\top J A S_n z_n(t) + z_n(t)^t op S_n^\top J b \tag{3.36}$$

use that $b = S_n e_1 \|b\|_2$, and simplify with equation (3.24).

$$\mathscr{H}(u_n) = \frac{1}{2} z_n(t)^\top J_n H_n z_n(t) + z_n(t)^t op J_n e_1 \|b\|_2 \tag{3.37}$$

And we see that

$$\mathscr{H}(z_n) - \mathscr{H}(u_n) = 0 \tag{3.38}$$

So the transformation does not change the energy, thus SLM is energy preserving.

Residual energy

Equation (3.39) and (3.40) are solemnly used to describe the residual energy of the methods. $\mathscr{H}_3$ describes the residual energy of the projected method, in u space, that is, after transforming back from z space, $\mathscr{H}_4$ is the residual energy in Z space. Since the transformation is symplectic it should not change the energy, $\mathscr{H}_3$ and $\mathscr{H}_4$ should therefore be identical. They are presented in the equation below.

$$\mathscr{H}_3(t) = \frac{1}{2}\epsilon^{(1)^\top}(t) J_m A \epsilon^{(1)} + \epsilon^{(1)^\top} J_m h^{(1)}_{n+1,n} v^{(1)}_{n+1} e_n^\top z(t) \tag{3.39}$$

$$\mathscr{H}(t) = \frac{1}{2}\frac{1}{2}\delta^{(1)^\top}(t) J_n H^{(2)}_n \delta^{(1)} + \delta^{(1)^\top} S^{(2)^\top}_n J_m h^{(2)}_{n+1,n} v^{(2)}_{n+1} e_n^\top z(t) \tag{3.40}$$

3.7.3 A comment on the orhogonalisation methods

The derivation of the methods above shows the vast similarities between SLM and KPM. The only practical difference between the two methods are the orthogonalisation methods. This makes implementation of the methods easier and more robust.

3.7.4 Direct method

It is important to have some method to compare with. This will be done by solving the problem without using any of the projection methods. It is easy to show that when solving a problem with any of the projection methods presented here you are trying to approximate the solution obtained without using a projection method. This is also the best any projection method can do. Thus direct method, or `DM` as it will be called in this text, is the natural comparator for these projection methods.

The proof of this will be shown here with trapezoidal rule and KPM, but the proof are very similar for SLM. Assume that the initial condition is zero.

$$\dot{u}(t) = Au(t) + F(t) \tag{3.41}$$

$$(I - \frac{Ah_t}{2})U_{i+1} = \Big(U_i + \frac{h_t}{2}\big(AU_i + F_{i+1} + F_i\big) \tag{3.42}$$

Apply the transformation $u = V_n z(t)$

$$(V_n - \frac{AV_n h_t}{2})z_{i+1} = \Big(V_n z_i + \frac{h_t}{2}\big(AV_n z_i + (F_{i+1} + F_i)\big) \tag{3.43}$$

Multiplying with $(V_n - \frac{AV_n h_t}{2})^{-1}$ gives

$$z_{i+1} = (I - \frac{Ah_t}{2})^{-1} V_n^\top \Big(V_n z_i + \frac{h_t}{2}\big(AV_n z_i + (F_{i+1} + F_i)\big) \tag{3.44}$$

Finally simplifying gives

$$z_{i+1} = (V_n - h_t \frac{V_n H_n + r_n e_n^\top}{2})^{-1}\Big(V_n z_i + \frac{h_t}{2}\big((V_n H_n + r_n e_n^\top)z_i + (F_{i+1} + F_i)\big) \tag{3.45}$$

If we now let n grow, r_n will become zero and the methods are the same. but if r_n is not zero, this will be an approximation of the untransformed problem. Note that r_n is unknown, and that the iteration scheme for the method is

$$z_{i+1} = (V_n - h_t \frac{V_n H_n + r_n e_n^\top}{2})^{-1}\Big(V_n z_i + \frac{h_t}{2}\big((V_n H_n + r_n e_n^\top)z_i + (F_{i+1} + F_i)\big) \tag{3.46}$$

3.7.5 Energy preservation of the methods

Suppose that the (shifted) problem

$$\dot{u}(t) = Au(t) + Au_0 u(0) = 0 \tag{3.47}$$

with a Hamiltonian matrix A can be solved with an energy preserving method, with energy calculated by equation (**??**). Examples of such energy preserving methods will in this text be trapezoidal rule, and midpoint rule. If the problem is approximated with $u = S_n z_n$, with S_n as defined in algorithm 3, the problem can be rewritten as

$$\dot{z}_n(t) = H_n z_n(t) + e_1 A u_0 u(0) = 0 \tag{3.48}$$

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!Her skal tinge bevises!!!!!!!!!!!!!!!!!!!!!!!!

Since H_n, given by Algorithm 3, is Hamiltonain, this will also be energy preserving if an energy preserving method is used.

The matrix S_n is symplectic, and so the transformation $u = S_n z_n$ is symplectic, meaning the transformation does not alter the energy.

That the energy is correct can be ensured by initial conditions since the methods are energy preserving. This means that the initial energy, given by the initial conditions will be the energy of the method at all times.

In theory this means that any Hamiltonian problem can be solved with SLM, without loosing any energy preserving properties as long as restart is not used.

SLM with restart does not have this property. The restart is influenced by a time dependent error function, and is therefore not Hamiltonian.

Note that this does not hold for KPM, since it does not give a Hamiltonian matrix H_n, or a symplectic transformation V_n.

!!!!!!!!!!!!!!!!!Om ikke determinanten til H_n er 1!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!Skriv at om du starter på nytt mange nok ganger så vil feilen bli så liten at metoden kanskje vil bli wymplektisk igjen.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

3.7.6 Linearity of the methods

In this section, assume that the initial condition of all differential equations is zero, or shifted so it becomes zero.

The projection methods require a vector that can be used to generate the orthogonal space. In the general (initial condition shifted) problem,

$$\dot{u}(t) = Au(t) + v \tag{3.49}$$

v is used to create the orthogonal space. But what if instead of just v, there where some time dependance, e.g. $v + F(t)$, or (more illustrative)

$$\dot{u}(t) = Au(t) + v + F(t) \tag{3.50}$$

In this case the projection method needs to be used two times, one time to solve $\dot{u}(t) = Au + v$ and another to solve $\dot{u}(t) = Au + F(t)$. The solutions 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 $\tilde{v}$ so that $\tilde{v}\tilde{F}(t) = v + F(t)$.

An even bigger problem arises when a differential equation has a source term that is not separable in time and space. An equation that is separable in time and space can be written as $p(t,x,y) = g(x,y) \cdot F(t)$, if it is not separable, it cannot be written in this way. As an example, consider the wave equation

$$\frac{\partial^2 u(t,x,y)}{\partial t^2} = \frac{\partial^2 u(t,x,y)}{\partial x^2} \frac{\partial^2 u(t,x,y)}{\partial y^2} + F(t,x,y) \quad \text{where } F(t,x,y) \neq g(x,y)F(t) \tag{3.51}$$

This equation cannot be discretized with a single vector v that makes the orthogonal space. If this was to be discretized as has been done above it would look like this:

$$\dot{u}(t) = Au(t) + \sum_{i=1}^{\hat{m}} e_i F_i(t) \tag{3.52}$$

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

Table 3.4: Computational cost of some mathematical operations.
!!!!!!!!!!!!!!!!!!Husk å cite alt!!!!!!!!!!!!!!
Sjekk om disse operasjonene stemmer!!!!!!!!!!!!!!

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 and y, gives a different time dependent function, and all of these functions need to be used to obtain the correct solution.
The reason the canonical vectors e_i are used is coincidental, actually set of orthonormal vectors can be used, but it is very simplifying to use the canonical vectors.
This means that equation (3.52) needs to be solved $\hat{m}$ times to obtain the solution. This gives the projection methods a terrible run time, and I advise you not to use any projection method if this is the case.

3.7.7 Number of operations

Since computation times might be uncertain due to different additional loads (web surfing, writing and so on) there will here be a short comparison between the number of operations for each method. This will also give a basis for what to expect from the different methods.

A table of computational cost for different operations is given in table **??**. Sections **??** to 3.7.7 will contain a motivation for the results found in table **??**. It is assumed that trapezoidal method is used.
!!!!!!!!!!!!!!!!!!!!!!!husk at det er forskjell på $\hat{m}$ og m!!
!!!!!!!!!!!!!!!!!!!Skriv mye her!!!!!!!!!!!!!!!!!!!!!

Method	Number of operation
KPM	$\mathcal{O}((n^2m + kn^3 + mnk)\gamma)$
SLM	$\mathcal{O}((nm^2 + kn^3 + mnk)\gamma)$
DM	$\mathcal{O}(km^3)$

Table 3.5: Number of operations needed for the different methods. γ is the number of restarts needed for the method to converge.
!!!!!!!!!!!!!!!!!Sjekk disse resultatene!!!!!!!!!!!!!!
!!!!!!!!!!!!!Husk å cite alt!!!!!!!!!!!!!!!

KPM

KPM uses Arnoldi's algorithm, an integration method, and requires an additional mnk operations when transforming $V_n z_n(t)$ to $u(t)$.

SLPM

SLPM uses SLM, an integration method, transformation, and requires an additional m^2 operations to calculate $J^{-1}S_nJ\zeta v$, which is a matrix vector product, and S_n is a dense matrix. !!!!!!!!!!!!!Forklar hvor de tallene kommer fra!!!!!!!!!!!!!!!!

DM

DM uses just an integration method, the downside is that $n = \hat{m}$.

3.8 SLM and its eigenvalue solving properties

If you do a quick internet search of "symplectic Lanczos method" most articles you find are in relation to symplectic Lanczos method's ability to approximate eigenvalues of Hamiltonian matrices. This section I will try to explain what makes this algorithm so attractive for these types of problems.

The eigenvalue problem is found in many different areas, eg. control theory, model reduction, system analysis and many more [**?**] [**?**].

SLM is a relatively fast algorithm[**?**] for transforming big sparse Hamiltonian matrix to smaller sparse Hamiltonian matrix with similar properties [**? ? ?**]. Eigenvalues of Hamiltonian matrices comes in pares, $\{\pm\lambda\}$, or in quadruples, $\{\pm\lambda, \pm\bar{\lambda}\}$ [**?**]. Since the small 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 eigenvalues, are found first [**?**].

But the method is not without is flaws. Frequent breakdown due to ill conditioned matrices occurs[**?**]. But due to its large potential much work has been make to overcome the difficulties. The most promising improvements are different types of restarts. This way the numerical estimations can be improved without loosing the Hamiltonian properties, and without a to high cost. Unfortunately not all are equally efficient. And there are still much ongoing research on the topic [**? ? ?**].

An Other method based on SLM is the SR - algorithm which has similarities with the QR - algorithm. Where 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, [**?**] shows under which assumptions the method converges.

Interestingly many of these papers [**? ?**] compare SLM to Arnoldi's algorithm and KPM for finding eigenvalue problems.

Chapter 4

Practice

Till now the explanation as been mostly theoretical and has explained how it should work. In this section there will be more of programming choices, how pictures was created and implementation.

4.1 Pictures

There are in general two different kinds of pictures.
One shows how some interesting data (eg. energy or error) behaves during one run. This picture is used in section **??**.
The other kind of picture is where each run of the program gives one point on the figure. The in data to the program is changed. The reason to use this technique is to remove noise from the calculations, or make more involved plots, for example convergence plots. To simplify the creation of these figures a program was made, and the whole method was made automatic, which in turn made the pictures less prone to errors. All other pictures is made with this program.

4.2 Measure error and energy

All errors are measured with the same function which measures the maximum absolute difference at each time step between the approximated solution, and the correct solution, relative to the correct solution. In the case where there are no correct solution, as in `semirandom`, the error

is measured in relation to the solution without the projection method.

The energy, or the change in energy, which is what is interesting, is also always found with the function. The energy is calculated for each time step, and initial energy is subtracted. When comparing energies the energies are calculated independently, and then subtracted from each other, without any scaling. In the case where the energy is supposedly constant the energy is calculated, and then not compared to any thing. This was because the correct solution would sometimes be the problematic part, and had an energy much larger than the approximated solution.

The exceptions are $\mathscr{H}_3$ and $\mathscr{H}_4$ which has their energy calculated by their own function.

These energies and errors was used to plot pictures. In the case where the automatic picture making program was used the maximum deviation (fro all times) was saved and used.

Other interesting results are 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 was used without restart. Each 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 or zeros from the plot, which made some of the plots confusing.
Computation time was measured as the time for the actual computation of the problem to be completed. Initial computations, or plots was not counted, as these was common for both the projection methods, and the direct method. The important part of the computation time is the difference between the methods, and I want to show this.

4.3 Test problems

Let the matrix I_j be the identity matrix with dimension j, and let

$$J_j = \begin{bmatrix} 0 & I_j \\ -I_j & 0 \end{bmatrix}. \tag{4.1}$$

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 was implemented, with some test problems. All test problems satisfies the condition

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

The test problems are discretized with $y_i = ih_s$, $x_i = ih_s$ and $t_j = jh_t$ with $i = 1,2,\cdots,m-1$ and $j = 1,2,\cdots,k$. The time discretized solution of u will be called U.

4.3.1 The wave equation

!!!!!!!!!!!!!!!!!!!!!Skriv at den ene løsningen er en tidsderiverte av den andre, og hvorfor vi måler feil på denne måten!!!!!!!!!!

The first is the 2 dimensional wave equation,

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + p(x,y)f(t). \tag{4.2}$$

The discretization is done by dividing each spacial direction in m pieces, with step size $h_s = 1/m$. To obtain the matrix, approximate $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ with $\tilde{A}$, where $\tilde{A}$ is the five point stencil[4]. Then write it as a system of first order (in time) differential equations

$$\begin{aligned} \dot{q}(t) &= Iw(t) \\ \dot{w}(t) &= -\tilde{A}q(t) + \tilde{v}\tilde{F}(t) \end{aligned} \tag{4.3}$$

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

$$\begin{aligned} \tilde{A} &= \frac{2}{h_s^2}\,\text{gallery}('\text{poisson}', m-2) \\ A &= \begin{bmatrix} 0 & I_{\hat{m}} \\ -\tilde{A} & 0 \end{bmatrix} \end{aligned} \quad , \tag{4.4}$$

which is a Hamiltonian matrix, equation (4.2) can now be written as equation (1.1). This matrix will be referred to as `wave`, and is a second order approximation.

Two test problems will be used, one with constant energy, and one with varying energy.

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

$$\begin{aligned} u(t,x,y) &= \sin(\pi x)\sin(2\pi y)\cos(\sqrt{5}\pi t) \\ \dot{u}(t,x,y) &= \sin(\pi x)\sin(2\pi y)\sqrt{5}\pi\sin(\sqrt{5}\pi t) \\ u_0(x,y) &= \sin(\pi x)\sin(2\pi y) \\ \dot{u}_0(x,y) &= 0 f(t,x,y) \qquad\qquad\qquad\qquad = 0, \end{aligned} \tag{4.5}$$

and for varying energy

$$\begin{aligned} u(t,x,y) &= \sin(\pi x)y(y-1)(t^2+1) \\ \dot{u}_0 &= \sin(\pi x)y(y-1)(2t) \\ u_0(x,y) &= \sin(pix)y(y-1) \\ \dot{u}_0 &= 0 \\ f(t,x,y) &= 2\sin(\pi x)y(y-1) - (t^2+1)\sin(\pi x)(2 - pi^2 y(y-1)). \end{aligned} \tag{4.6}$$

The test problem are not symmetric, and has variety of exponential and polynomial functions, which makes it more general.

When measuring error it will compare the approximated error for both q and w with the correct solution, not just q. The reason for this is that there might be some relevance in solving the problem for u, and not just q.

4.3.2 A random test problem

The second implemented Hamiltonian matrix is random, and given by

$$\begin{aligned} \hat{A} &= \text{rand}(\hat{m}) \\ A &= \frac{1}{2} J_{\hat{m}} \cdot (\hat{A} + \hat{A}^\top + m^2 I_{\hat{m}}). \end{aligned} \tag{4.7}$$

Since we are interested in comparing the different projection methods to each other, the matrix will be saved and reused. This matrix will be referred to as `semirandom`. The part $m^2 I_{\hat{m}}$ 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.

The test problem when the energy is constant is

$$\begin{aligned} u(t,x,y) &= \text{unknown} \\ \dot{u}(t,x,y) &= \text{unknown} \\ u_0(x,y) &= \text{rand}(2(m-2)^2, 1) \\ f(t,x,y) &= 0 \end{aligned} \tag{4.8}$$

and

$$\begin{aligned} u(t,x,y) &= \text{unknown} \\ u_0(x,y) &= 0 \\ f(t,x,y) &= \text{rand(1,k)} \cdot \text{rand}(2(m-2)^2, 1), \end{aligned} \tag{4.9}$$

when the energy is varying.

Since the solution 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 this as a test problem. This case shows how the projection methods solves a more general Hamiltonian system, making it more difficult for the projection methods to find an approximated solution, while `wave` always has the same matrix, with different initial vectors. This case will also show more correctly how the restart can

be used to improve the solution.

The error will in this case be measured as the difference between the projection method used, and without the use of any projection method. This will make the error seam a lot smaller than it really is, but as shown in section **??**, it is correct to compare the solution methods 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.

4.4 Implementation

!!!!!!!!!!!!!!!!!!!!!!Skriv at alt er funksjoner, som er brukt om på nytt!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!SKRIV bedre her!!!!!!!!!!!!!!!!!!!!!!!!!

All algorithms and methods was made in matlab R2014b on an ubuntu 14.04 LTS computer with intel i7 4770 CPU and 16 GB of RAM. Matlab's backslash operator was used to solve the linear system in trapezoidal rule, and midpoint rule.

!!!!!!!!!!!Skrive hvordan bilder er laget!!!!!!!!!!!

!!!!!!!!!!!!!!Skrive hvordan funksjoner er implementert!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!Skrive hvordan ting er sammensatt!!!!!!!!!!!!!!!!!!

Chapter 5

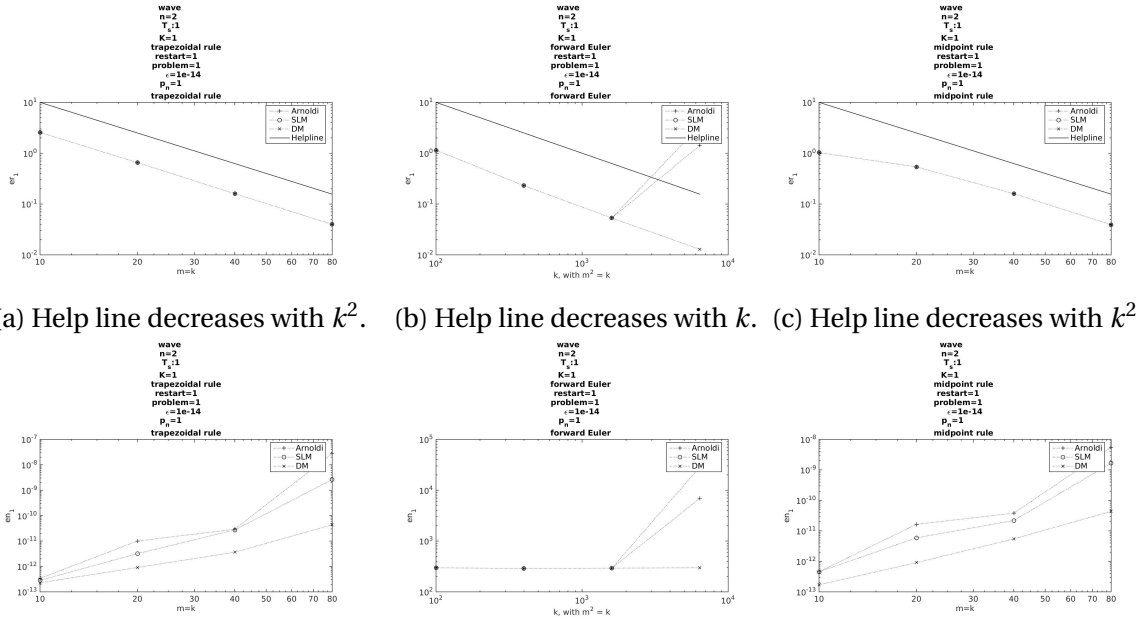
Results

This chapter will contain a comparison between the different projection methods, and also with DM. Energy preservation with SLM is one of the primary concerns, but computation time, and difference between integration methods will also be mentioned.

5.1 Convergence

This section will show convergence of all the methods, with both constant and varying energy. As a bonus, the energy will also be shown.

5.1.1 Constant Energy



(a) Help line decreases with k^2. (b) Help line decreases with k. (c) Help line decreases with k^2.

(d) Help line decreases with k^2. (e) Help line decreases with k. (f) Help line decreases with k^2.

Figure 5.1: Figure of the convergence for the different integration methods. Notice that forward Euler requires a much larger k to obtain the same precision as the other methods. All methods converge with the expected rate. $n = 4$ and $\epsilon = 1e-14$.

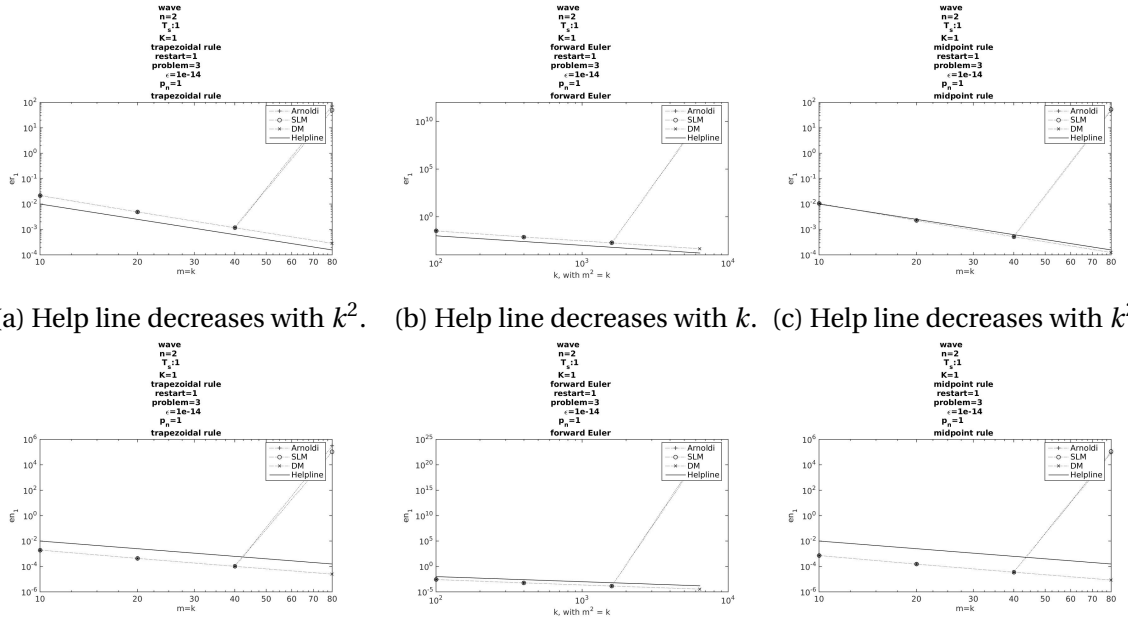
It is clear from figure 5.1 and 5.2 that all methods can be forced to have the same convergence if ϵ is chosen to be near machine accuracy.

Clearly forward Euler is not the iteration method you should use when you are concerned about the energy, but it does give the correct convergence rate, which is linear.

The other methods give a better approximation of the energy, and converges quadratically and near identically, as they should. The energy for all the methods is increasing, which is not desirable. This happens because more round off errors occur with larger m and k, and when all these small numbers are added together to estimate the energy the become significant.

The reason for the small difference between midpoint and trapezoidal rule is that when the restart is performed midpoint rule is better at approximating the resulting error equation (the equation has non constant energy) than trapezoidal rule is.

5.1.2 Varying Energy



(a) Help line decreases with k^2. (b) Help line decreases with k. (c) Help line decreases with k^2.

(d) Help line decreases with k^2. (e) Help line decreases with k. (f) Help line decreases with k^2.

Figure 5.2: Figure of the convergence for the different integration methods. Notice that forward Euler requires a much larger k to obtain the same precision as the other methods. All methods converge with the expected rate. $n = 4$ and $\epsilon = 1e-14$. All methods converge with the expected rate.

Forward Euler is running into some unexplainable problems with a sudden diverge at the last point. Though in this case it gives a nice convergence for both global energy and global error, except for the last point.

Trapezoidal rule and midpoint rule are again the better choices, with midpoint rule being slightly ahead. This is not a surprise since midpoint rule is a symplectic method while trapezoidal rule's good energy can be explained from is't quderatic convergence.

If you compare these figures to figure 5.1, you will see that the energy is much higher in this case, but that they are decreasing, and not increasing. The reason for this is that the energy in figure 5.2 is compared to the energy for the correct solution, while in figure 5.1 the energy is calculated, and not compared to anything. This gives a sort of bias and random answer, but is more useful than seeing how the energy changes for this case, or seeing how poorly the correct

solution preserves energy.

!!!!!!!!!!!!!!!!!!det må forklares et sted at trapezoidal ikke alltid er symplectisk men at midpouint alltid er det og at forward euler er ubrukelig (teoretisk altså)!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!Skriv om noe om energiens konvergens når bildene er ferdige!!!!!!!!!!!!!!!!!

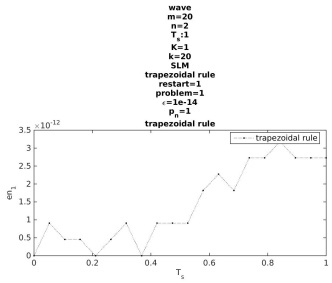
!!!!!!!!!!!!!!Kommentere litt at metodene fungerer i alle tilfeller!!!!!!!!!!!!!

5.2 Integration method

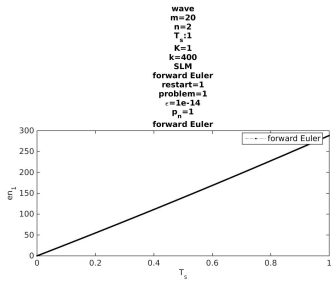
This section will be concerned with comparing some different methods for integration in time. How well they estimate the error and energy will be the primary concern.
!!!!!!!!!!!!med eller uten restart?!!!!!!!!!!!!!!!!!!!!

5.2.1 Constant energy with wave

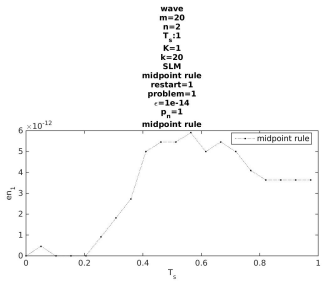
!!!!!!!!!!!!!!Skrive litt om hva denne konvergensen er!!!!!!!!!!!!!!!



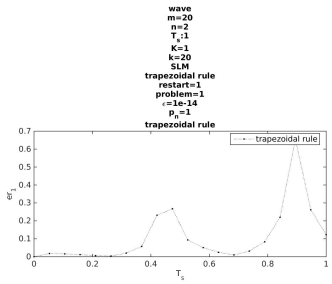
(a) Energy with trapezoidal rule.



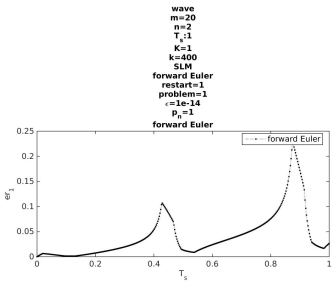
(b) Energy with forward Euler



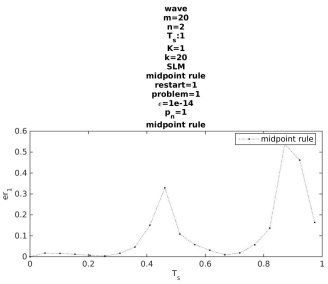
(c) Energy with midpoint rule



(d) Error with trapezoidal rule



(e) Error with forward Euler



(f) Error with midpoint rule

Figure 5.3: A snapshot of how energy and error changes over the simulated time with the different integration method without restart. Notice that forward Euler requires a much larger k to obtain the same precision as the other methods. $m = 20$, $n = 4$ $k = 20$, restart is enabled with $\epsilon = 1e-14$.

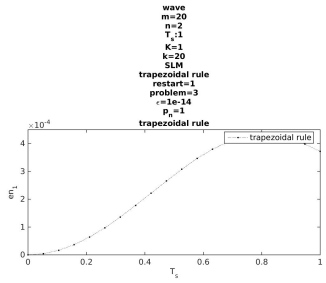
Both trapezoidal rule and midpoint rule to gives a very precise estimation of energy and error. Forward Euler on the other hand gives a linearly increasing energy, and with that a larger error, making it useless as an integration method in this case. The reason for the wired shapes in figure 5.3d and 5.3f is the periodicity of the test problem.

Forward Euler gives a good approximation of the error, but since the energy is linearly increasing it should not be used together with SLPM.

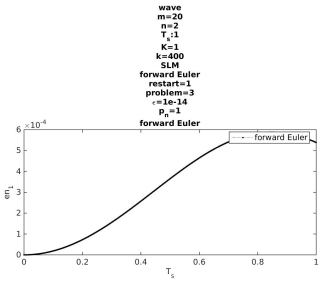
Trapezoidal rule and midpoint rule behaves very similar, with midpoint rule having a small advantage again because of the restart.

!!!!!!!!!!!!!!!Skrive at disse bildene er laget på en annen måte enn de andre!!!!!!!!!!!!

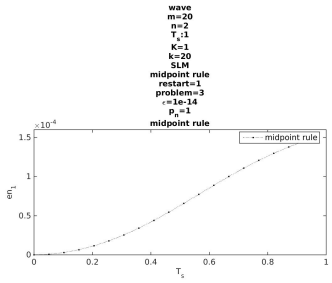
5.2.2 Varying energy with waves



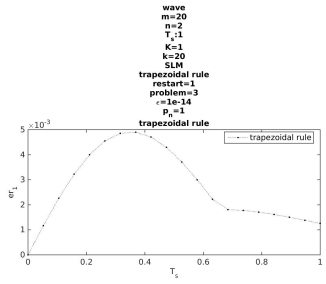
(a) Energy with trapezoidal rule.



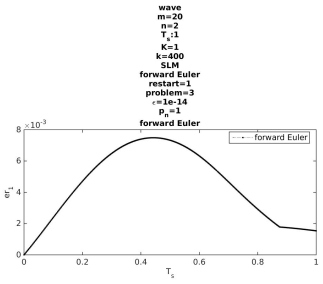
(b) Energy with forward Euler.



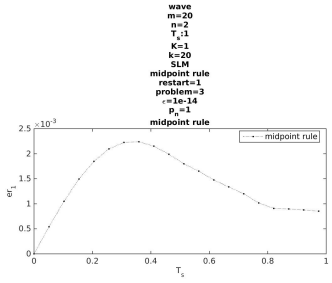
(c) Energy with midpoint rule.



(d) Error with trapezoidal rule.



(e) Error with forward Euler.



(f) Error with midpoint rule rule.

Figure 5.4: A snapshot of how energy and error changes over the simulated time with the different integration method without restart.

In this case all methods gives a suitable convergence for the error and the energy. There is no longer any problems with forward Euler's increasing energy. Again the shapes on the figures is a property of the test problems, and not a flaw in the method.

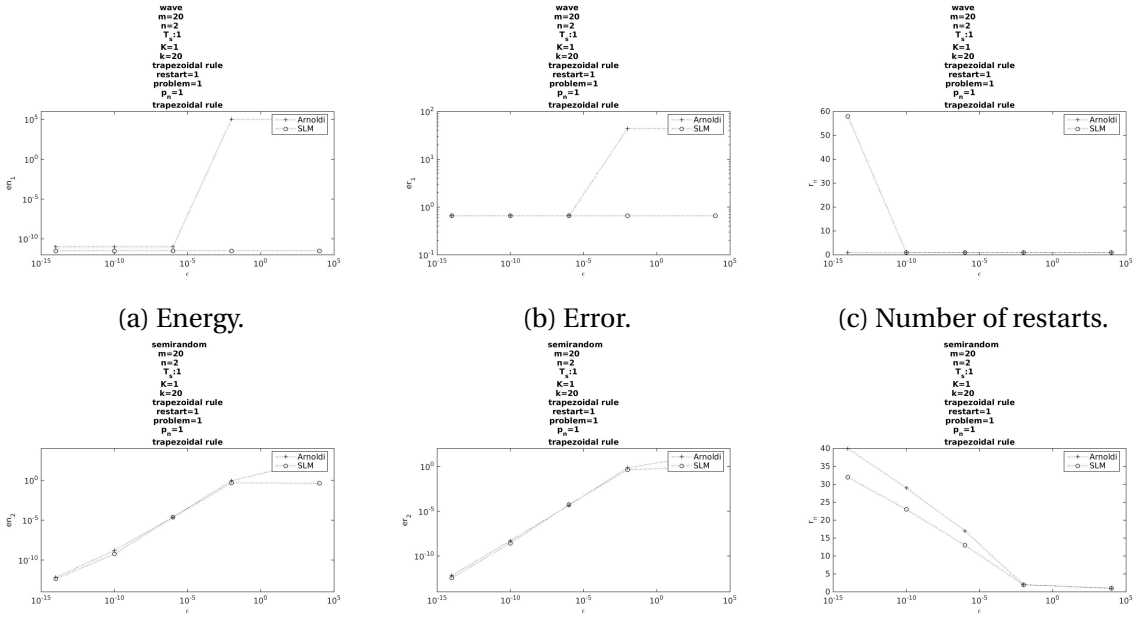
Again midpoint rule performs better. From here on, forward Euler will no longer be used due to its poor energy preserving properties. Trapezoidal rule will be used when the energy is constant, though midpoint gives a better approximation. The reason for this is that there is no huge difference between trapezoidal rule and midpoint rule when the energy is constant, trapezoidal rule is also a bit faster. When the energy is not constant midpoint rule will be used.

5.3 How to choose ϵ

This section will look at how the different projection methods compare to each other in relation to error, energy and time consumption. There will also be a comparison between restart variable and computation time.

5.3.1 Constant energy

!!!!!!!!!!!!!!!!!!!!Det må skrives en del mer i dette kapitelet!!!!!!!!!!!!!!!!!!!!!!!



(a) Energy. (b) Error. (c) Number of restarts.

(d) Energy. (e) Error. (f) Number of restarts.

Figure 5.5: The figure shows how the different methods change the energy and error with different number of restart for `semirandom`.

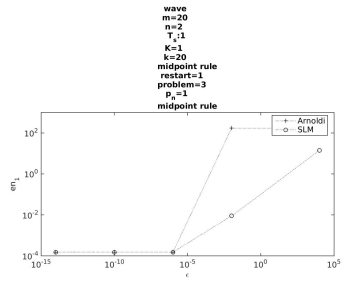
There is a clear difference between how well the energy is estimated for `wave` and `semirandom`. For `semirandom` there is not much difference between SLM and Arnoldi, although SLM seams to give a slight better approximation than Arnoldi. With `semirandom` at $\epsilon = 10^{-10}$ Arnoldi preforms one more iteration than SLM, which makes it give a better approximation, aside from that SLM consistently preform better than Arnoldi.

When comparing figure 5.5 and figure **??** it is important to remember that the way the er_1 and er_2 is found is very differently, and that might explain the why the cases differs so much. But the

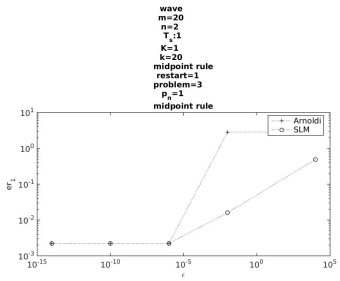
difference might also be explained with `semiradom` being a much harder equation to solve.

!!!!!!!!!!!!!Skriv litt mer om hvorfor bildene for semirandom og wave er så forksjelkige!!!!!!!!!!!!!!!!!!!!!

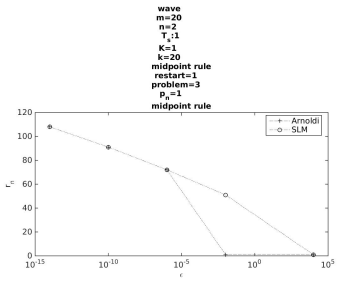
5.3.2 Varying energy



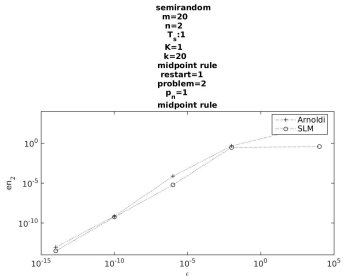
(a) The difference in energy with and without restart.



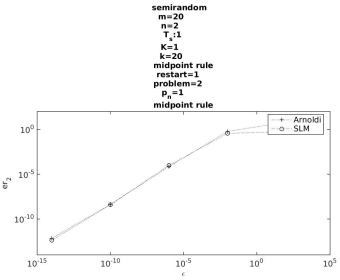
(b) The difference in energy with and without restart.



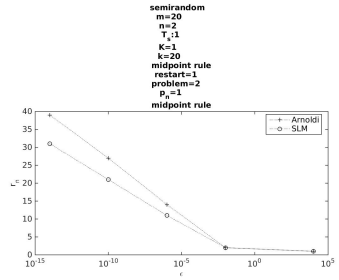
(c) The number of iterations performed with and without restarting.



(d) The difference in energy with and without restart.



(e) The difference in energy with and without restart.

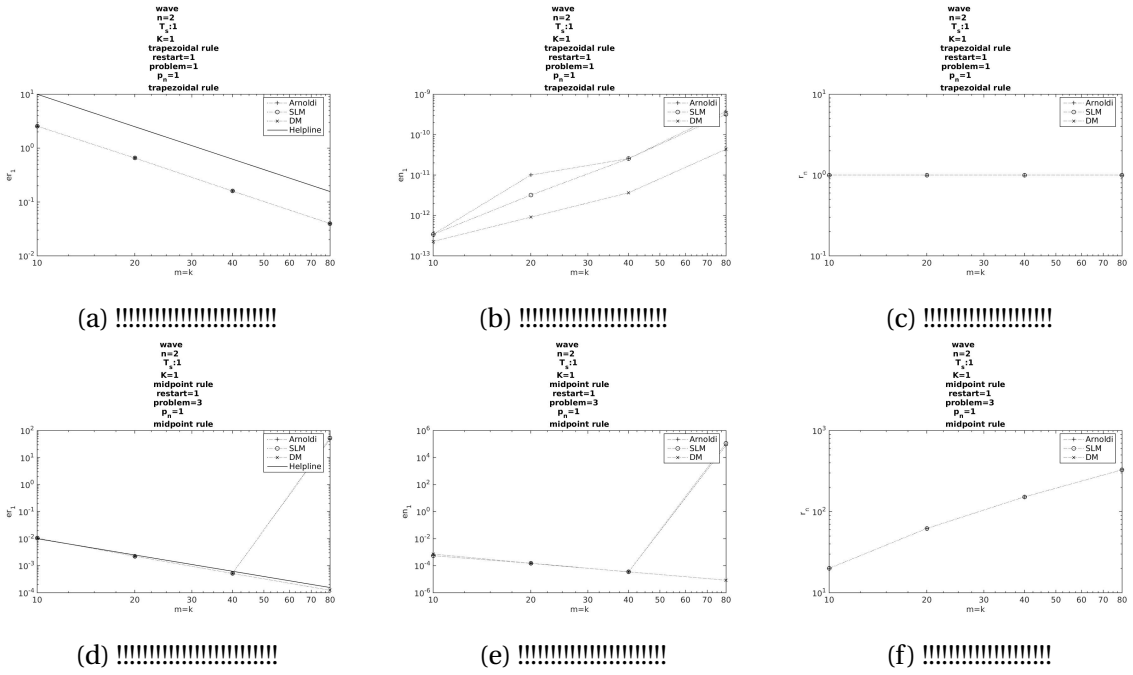


(f) The number of iterations performed with and without restarting.

Figure 5.6: The figure shows how the different methods change the energy and error with different number of restart for `wave`.

The results here are very similar to the results found in section 5.3.1, except that the energy and error changes for the first points for both SLM and Arnoldi. SLM still gives the better approximation.

5.3.3 A rule based for ϵ



(a) !!!!!!!!!!!!!!!!!!!!!!!!!!!!

(b) !!!!!!!!!!!!!!!!!!!!!!!!!!!!

(c) !!!!!!!!!!!!!!!!!!!!!!!!!!!!

(d) !!!!!!!!!!!!!!!!!!!!!!!!!!!!

(e) !!!!!!!!!!!!!!!!!!!!!!!!!!!!

(f) !!!!!!!!!!!!!!!!!!!!!!!!!!!!

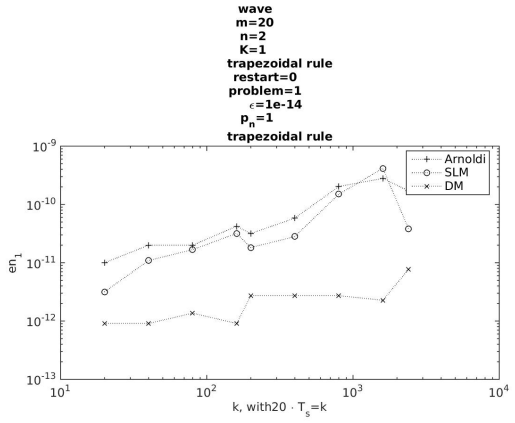
Figure 5.7: The figure shows how the different methods change the energy and error with different number of restart for `wave`.

5.4 Energy and error

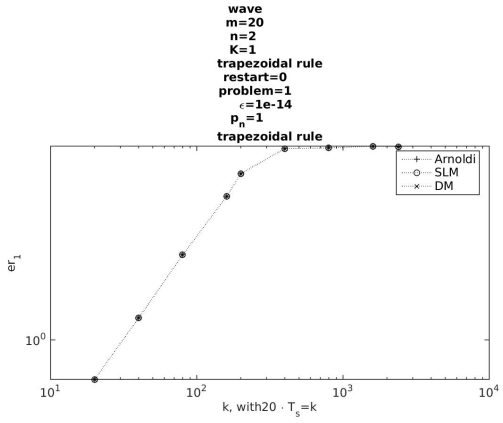
Since SLM gives a symplectic matrix, the energy should not change with time, and the error should increase linearly with time. This section will be devoted to seeing how well this works in practice. There will also be a comparison about how the restart changes the energy preservation. In addition we will see if there is much difference if the energy is constant or varying.

5.4.1 Constant energy

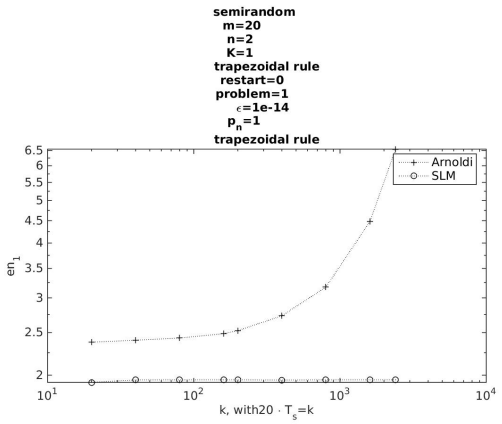
Energy and error without restart



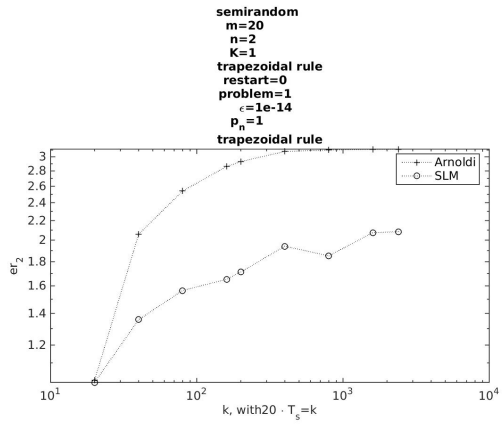
(a) Global error for `wave`.



(b) Global energy for `wave`.



(c) Global error for `semirandom`.

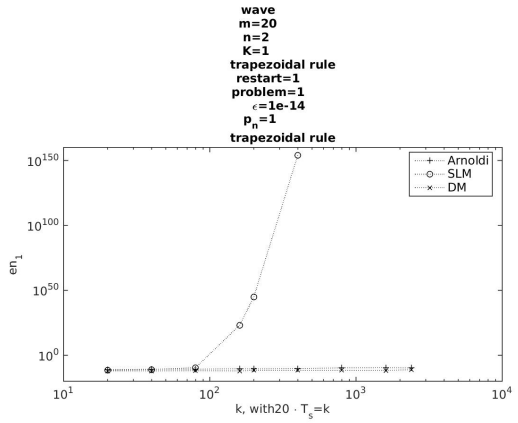


(d) Global energy for `semirandom`.

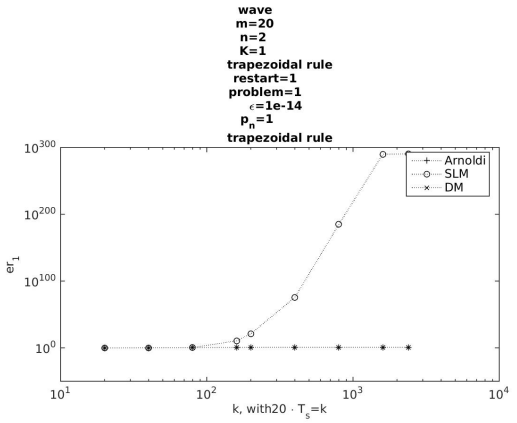
Figure 5.8: The figure shows how the global energy and global error changes as a function of simulated time, without restart. $m = 20$, $n = 8$ and trapezoidal rule is used to integrate.

The figures shows no or small differences between the global error for KPM and SLPM. It is clear that DM gives a better approximation than the other two methods, but not by far, and this is without restart. Regarding energy I think figure 5.8c shows it best. KPM is clearly not the best orthogonalisation method when it comes to estimating energy.

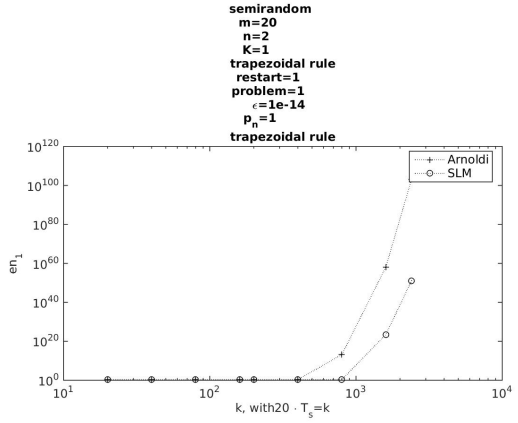
Energy and error with restart



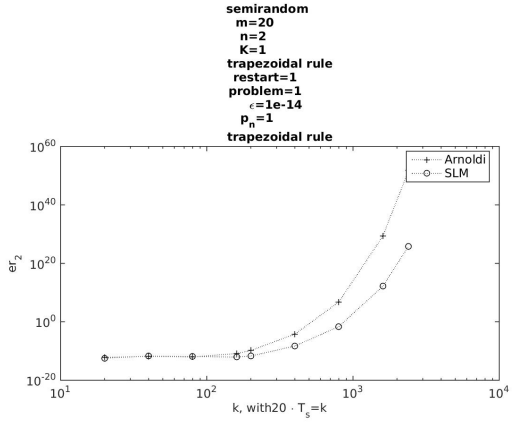
(a) Global error for `wave`.



(b) Global energy for `wave`.



(c) Global error for `semirandom`.



(d) Global energy for `semirandom`.

Figure 5.9: The figure shows how the global energy and global error changes as a function of simulated time with restart. $m = 20$, $n = 8$ and trapezoidal rule is used to integrate.

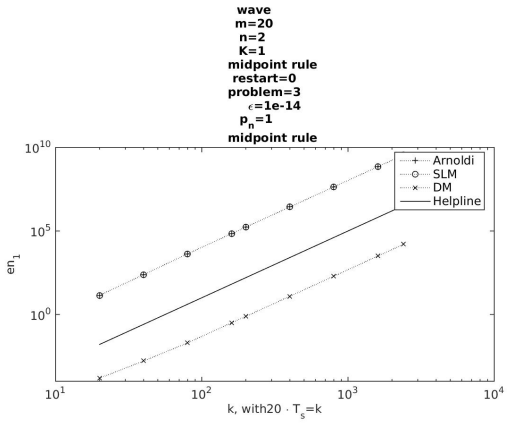
This picture shows some of the biggest weakness of the orthogonalisation methods I have uncovered. The restart! The reason for the incredible increase on the energy is due to the fact that when restart is enabled the system is no longer symplctic, and thus the energy is no longer constant, and every restart changes the energy creating a feedback loop.

The good news on the other hand is that if ϵ is kept small enough the error will not diverge. SLPM also seams to be better at maintaining a small error, probably because of a smaller increase in energy.

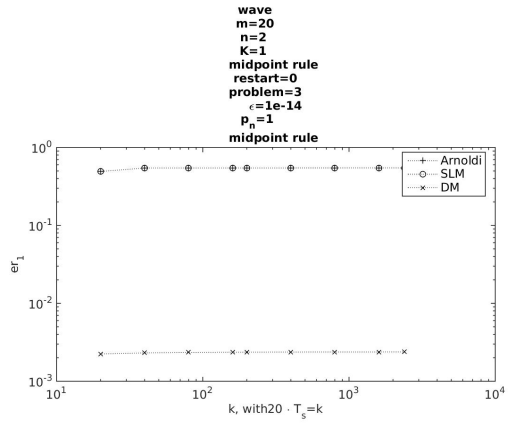
5.4.2 Varying energy

The focus has till now been to look at how the methods work when the energy is constant. This section will show how the methods work when the energy is varying.

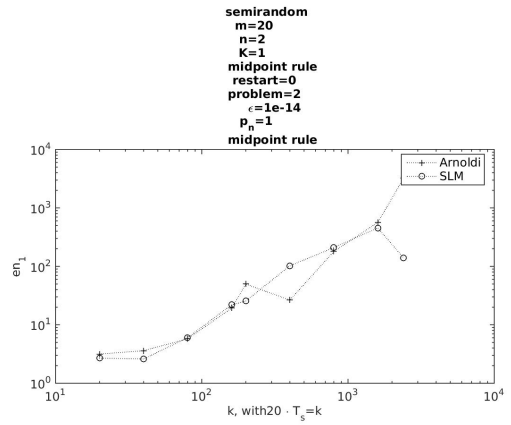
Energy and error without restart



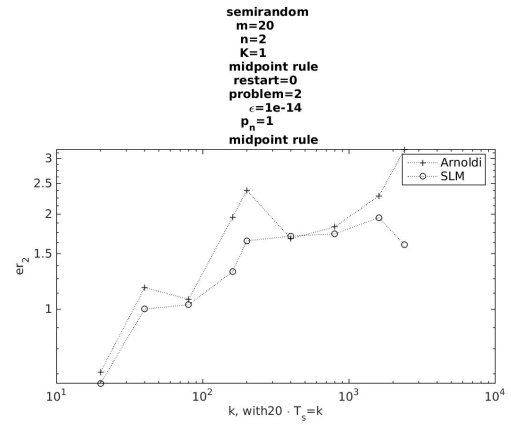
(a) Global error for `wave` without restart.



(b) Global energy for `wave` without restart.



(c) Global error for `wave` with restart.

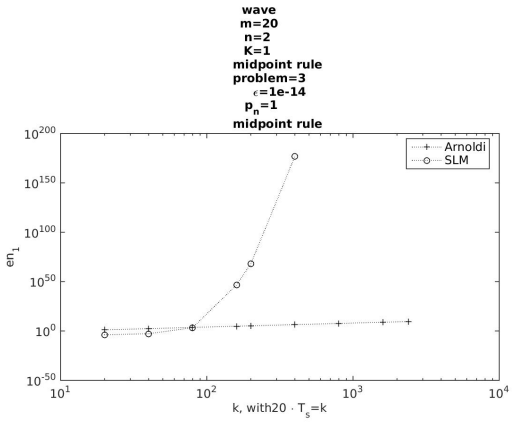


(d) Global energy for `wave` with restart.

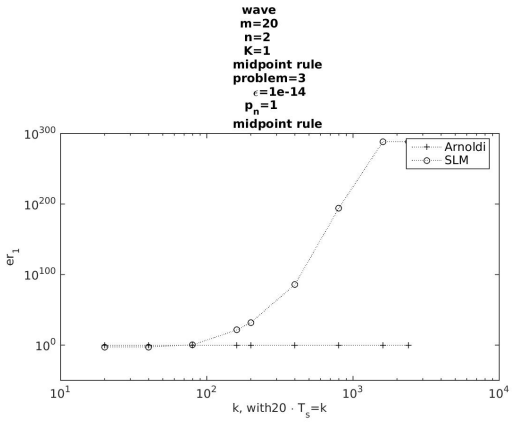
Figure 5.10: !!!!!!!!!!!!!!!Skriv noe her!!!!!!!!!!!!!!! og på alle subcaptioner!!!!!!!!

In this case there is no difference between the projection methods. DM performs marginally better than the other methods, but not asymptotically better, there seams to be a constant difference between the two methods, probably depending on n, where a larger n would give a smaller difference. !!!!!!!!!!!!!!!!!!!Skriv noe!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

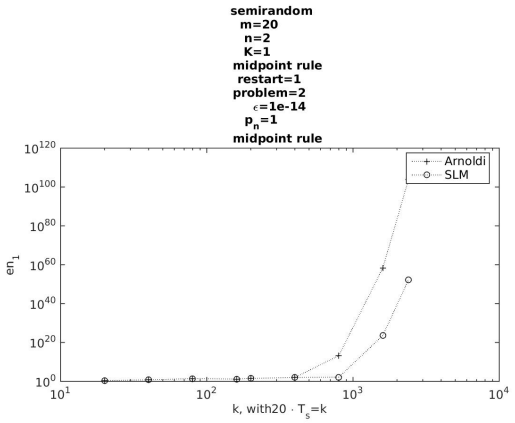
Energy and error with restart



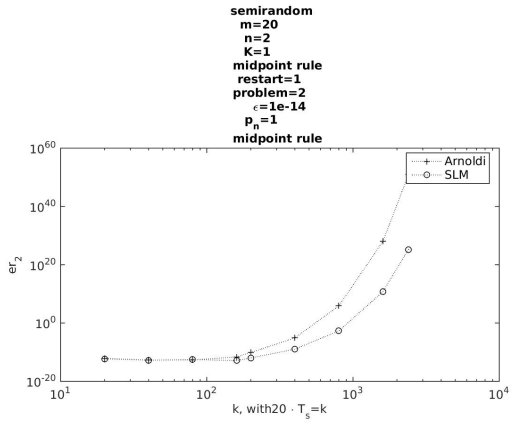
(a) Global error for `wave` without restart.



(b) Global energy for `wave` without restart.



(c) Global error for `wave` with restart.



(d) Global energy for `wave` with restart.

Figure 5.11: !!!!!!!!!!!!!!!!Skriv noe her!!!!!!!!!!!!!!!! og på alle subcaptioner!!!!!!!!

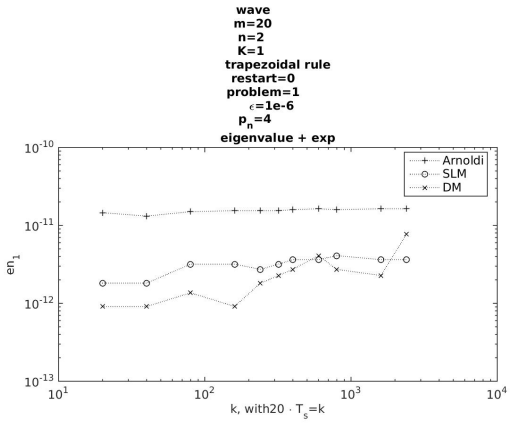
Both SLPM and KPM does an equally poor job at approximating the solution when restart is enabled. An interesting event is the step happening in figure 5.11b, this is the point where the energy starts to diverge. Another interesting thing about that is that KPM never seams to increase with the same rate as SLPM. So for longer times it seams that SLPM does not always have an advantage over KPM. But if you look closely you will see that the energy for SLPM is initially much smaller for KPM.

5.4.3 Constant energies with SLM

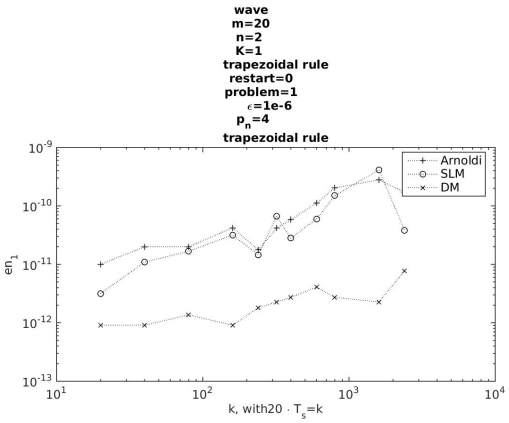
Some theoretical results was presented for me by Lu Li. The theory can be found in [**?**]. In short she predicted that the global energy of SLPM should be constant when restart is not enabled,

in addition to energy $\mathscr{H}_3$ and $\mathscr{H}_4$ being constant. An other important result would be to see if the global energy is constant when restart is enabled. These question will be answered in this section.

Energy of SLM without restart



(a) Global energy for `wave` without restart.

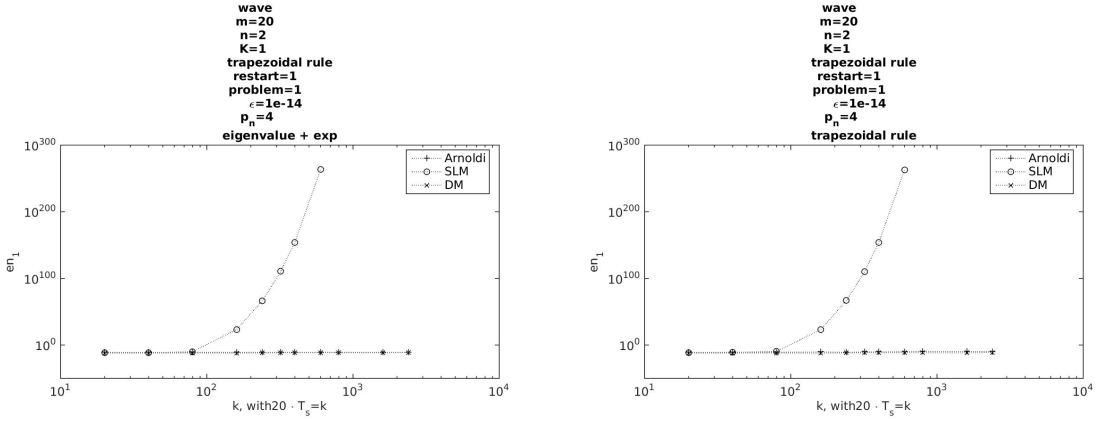


(b) Global energy for `semirandom` without restart.

Figure 5.12: The change in energy for functions that should have constant energy. $m = 20$, $n = 4$, trapezoidal rule

When I look at the figure I would say that it is clear that the energy is not constant for either SLPM or DM. But DM is a symplectic method! It uses a Hamiltonian matrix, and a symplectic integrator, So why does it increase? My guess would be small round off errors that when summed up enough becomes considerable large. So this explains why DM has a small increase. But what about SLPM, my guess is that the increase in energy not is actually that much bigger, it changes more, both up and down, but if you look at the first and last point it is actually about the same increase as DM has. The increase in energy is in any case much smaller than the linear increase a non-symplecitc method would have. It seams close enough for me, and I conclude that SLPM is energy preserving, with a symplectic integrator.

Energy og SLM with restart



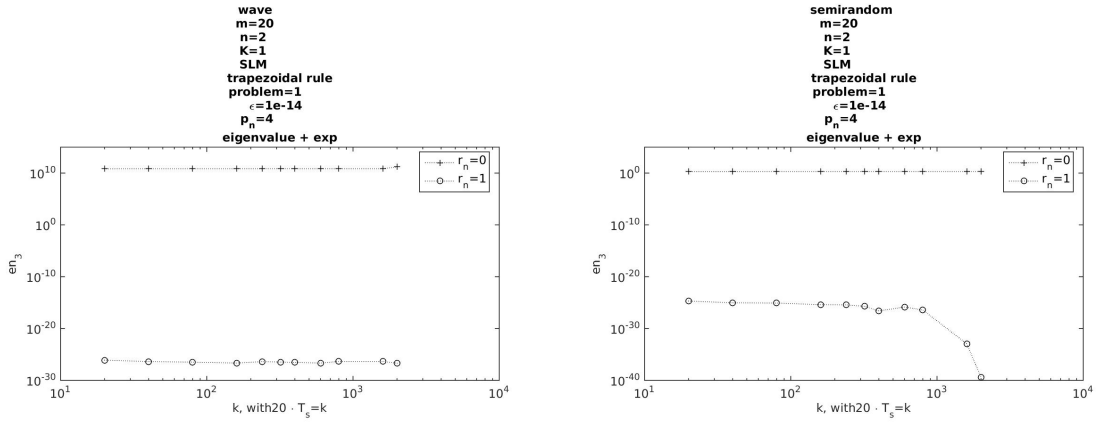
(a) Global error for `wave` without restart.

(b) Global energy for `wave` without restart.

Figure 5.13: !!!!!!!!!!!!!!!!Skriv noe her!!!!!!!!!!!!!!!! og på alle subcaptioner!!!!!!!!

I have pointed it out before, but the restart is not energy preserving. The picture shows an incredible increase in energy. Once again it is apparent that you should not let the time intervals become to big when restart is enabled.

Noe med energy3



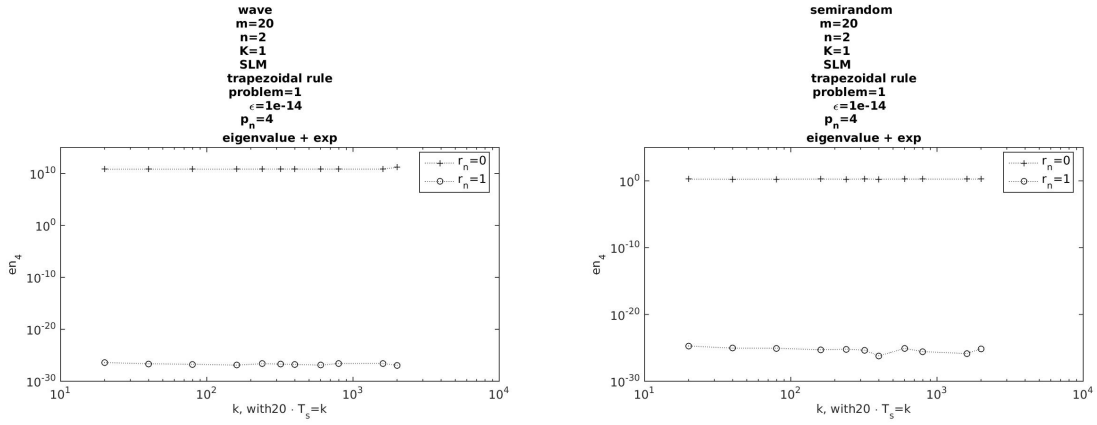
(a) Energy $\mathscr{H}_3$ for `wave`.

(b) Energy $\mathscr{H}_3$ for `semirandom`.

Figure 5.14: A plot of $\mathscr{H}_3$, discussed in section **??**. $m = 20$, $n = 4$, trapezoidal rule, $\epsilon = 1e-14$, problem 1.

This seams to be quite straight forward, the energy is constant. It is much more constant than anything discussed so far, restarting does, for once, not ruin this property.

Noe med energy4

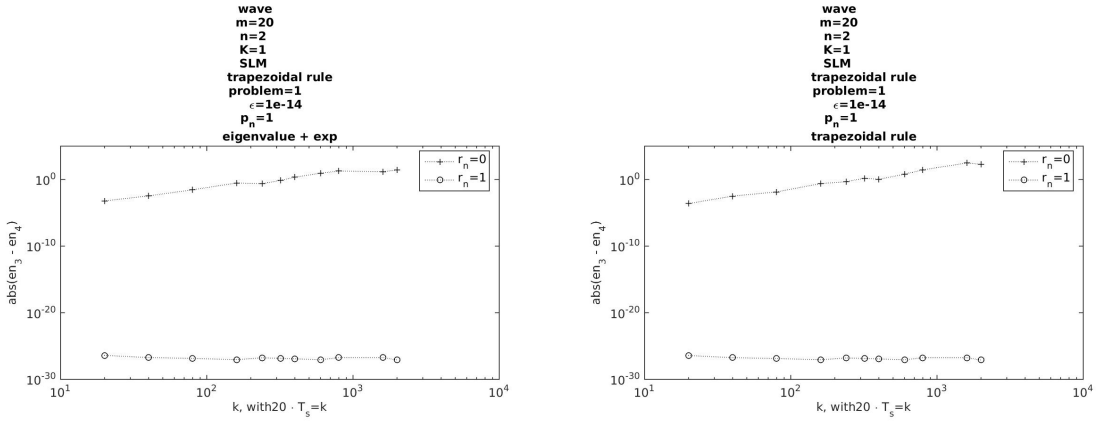


(a) Global energy for `wave` without restart. (b) Global energy for `semirandom` with restart.

Figure 5.15: !!!!!!!!!!!!!!!!Skriv noe her!!!!!!!!!!!!!!!! og på alle subcaptioner!!!!!!!!

This also seams to be quite forward. On the plots there seams to miss a lot of points. The missing points are in fact values calculated to be true zeros. This is clearly either zero or constant. ! !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!Skal det jeg beviser være null eller constant? SJEKK!!!!!!!!!!!!!!!!

Difference between energy3 and energy4



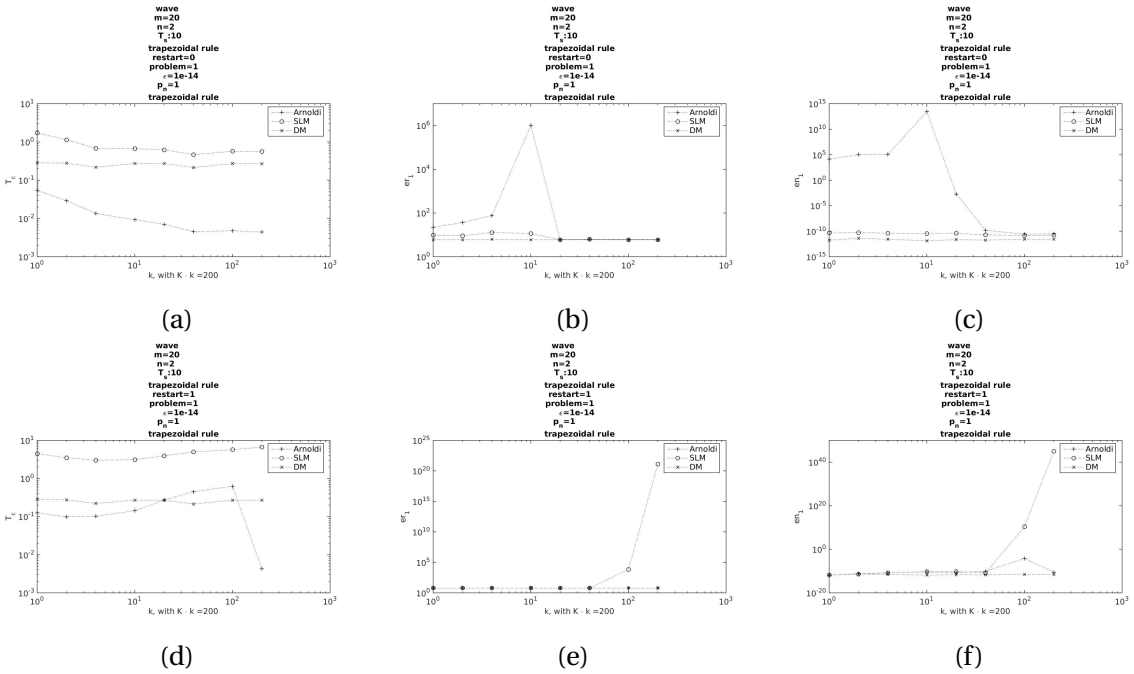
(a) Global energy for `wave` without restart. (b) Global energy for `semirandom` with restart.

Figure 5.16: !!!!!!!!!!!!!!!!Skriv noe her!!!!!!!!!!!!!!!! og på alle subcaptioner!!!!!!!!

5.5 Divide the time domain

One of the major problems with the orthogonalisation methods is that when dealing with large time intervals it might not converge, and the restarts does not help at all. So lets try something different. In this section the time domain is divided in smaller pieces so that the restart can be used without the energy and error diverging, as described in section **??**. Since using the orthogonalisation algorithms might be time consuming that will also be something to look at.

5.5.1 Constant energy



(a) (b) (c)

(d) (e) (f)

Figure 5.17: Figure showing how the different methods perform with different distribution of K and k. Notice that the product Kk is constant.

DM is not affected much by this change, with energy, error and time comsuption nearly constant. But for SLPM and KPM

5.5.2 Varying energy

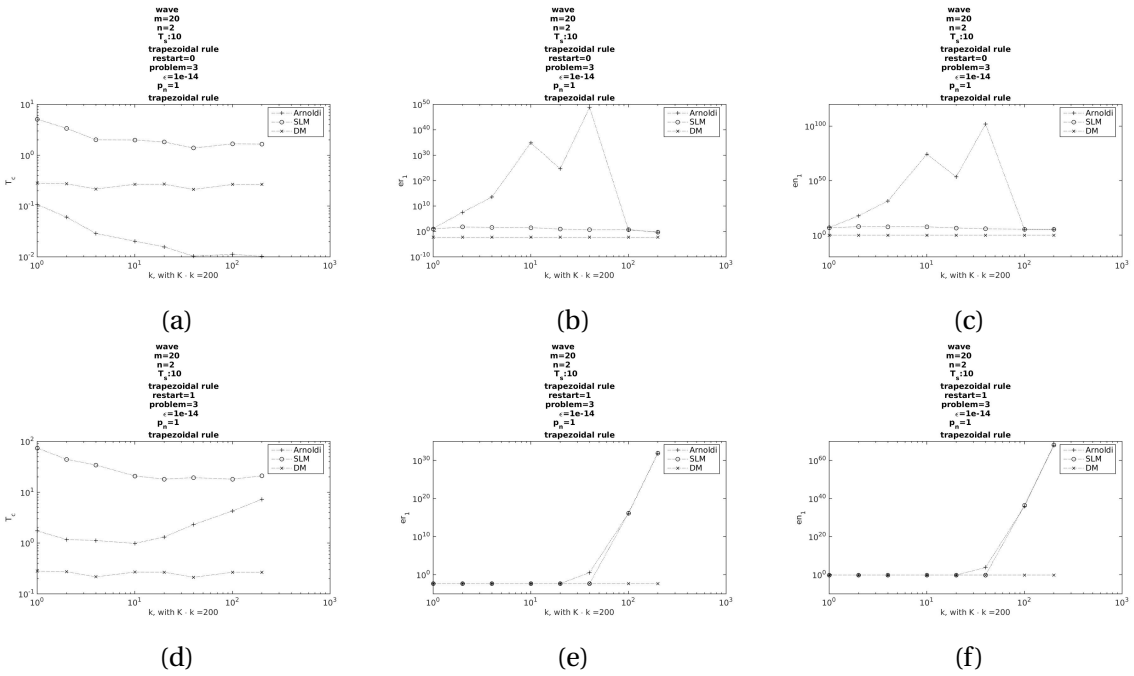


Figure 5.18: Figure showing how the different methods perform with different distribution of K and k. Notice that the product Kk is constant.

Figure 5.18 is very similar to figure 1. But there is here a definite decrease in computation time for SLM. Notice also that for larger K the energy and error diverges for both SLM and Arnoldi.

5.6 How to choose n

The restart variable, denoted by n, is the size of the orthogonal system used with one of the projection method. Performing n iterations of Arnoldi's algorithm or $\frac{n}{2}$ iterations of SLM gives an orthogonal space of size n.

Constant energy

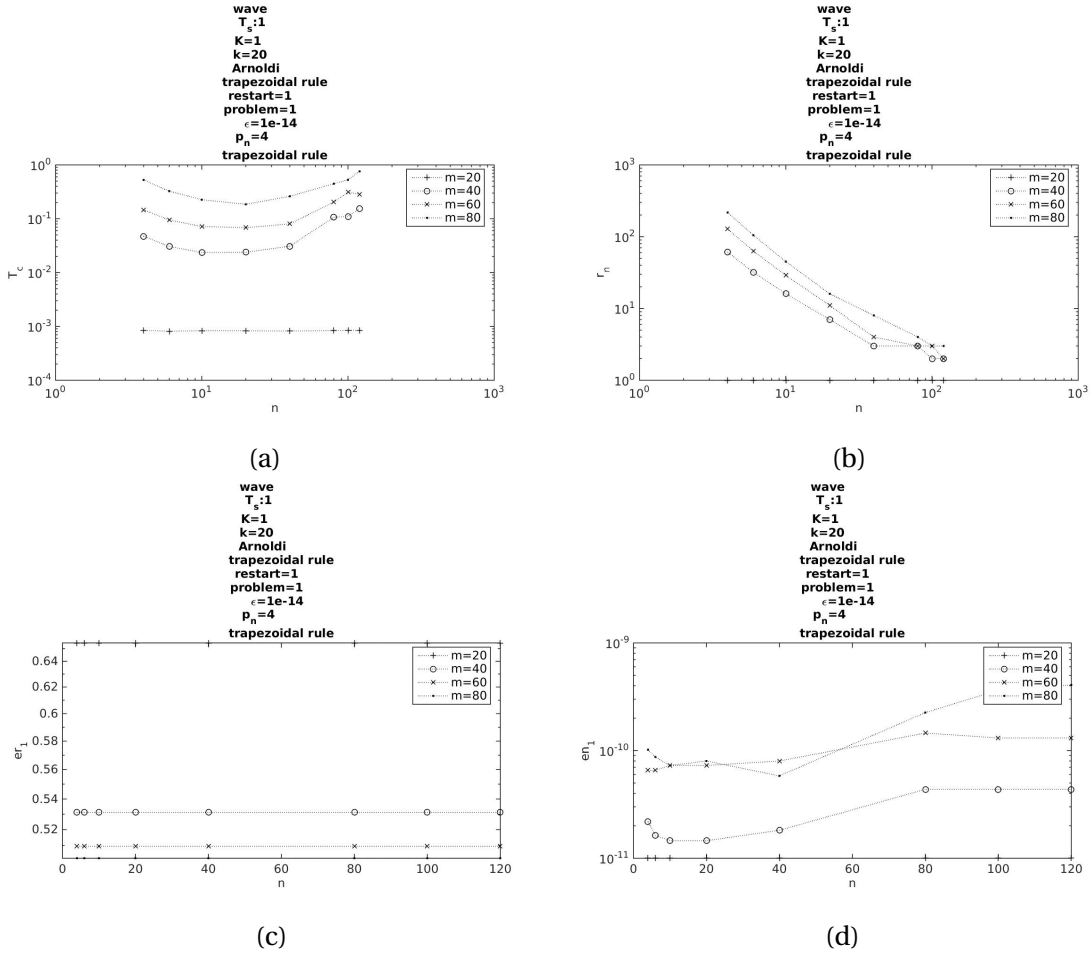


Figure 5.19: computation time, number of restarts, error and energy are all plotted here with different m and n.

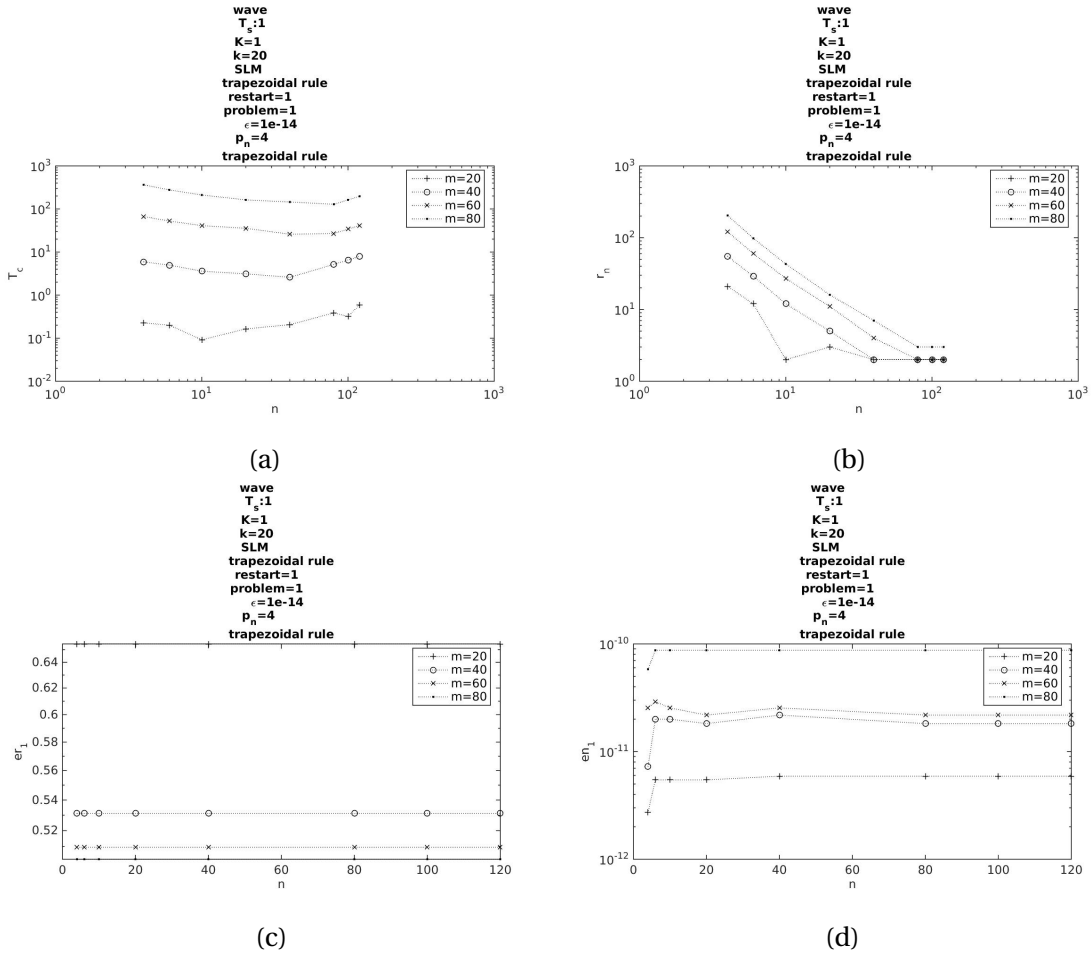


Figure 5.20: computation time, number of restarts, error and energy are all plotted here with different m and n.

!!!!!!!!!!!!!!!Skriv noen commentarer her!!!!!!!!!!!!!!!!!!!!!!

Varying energy

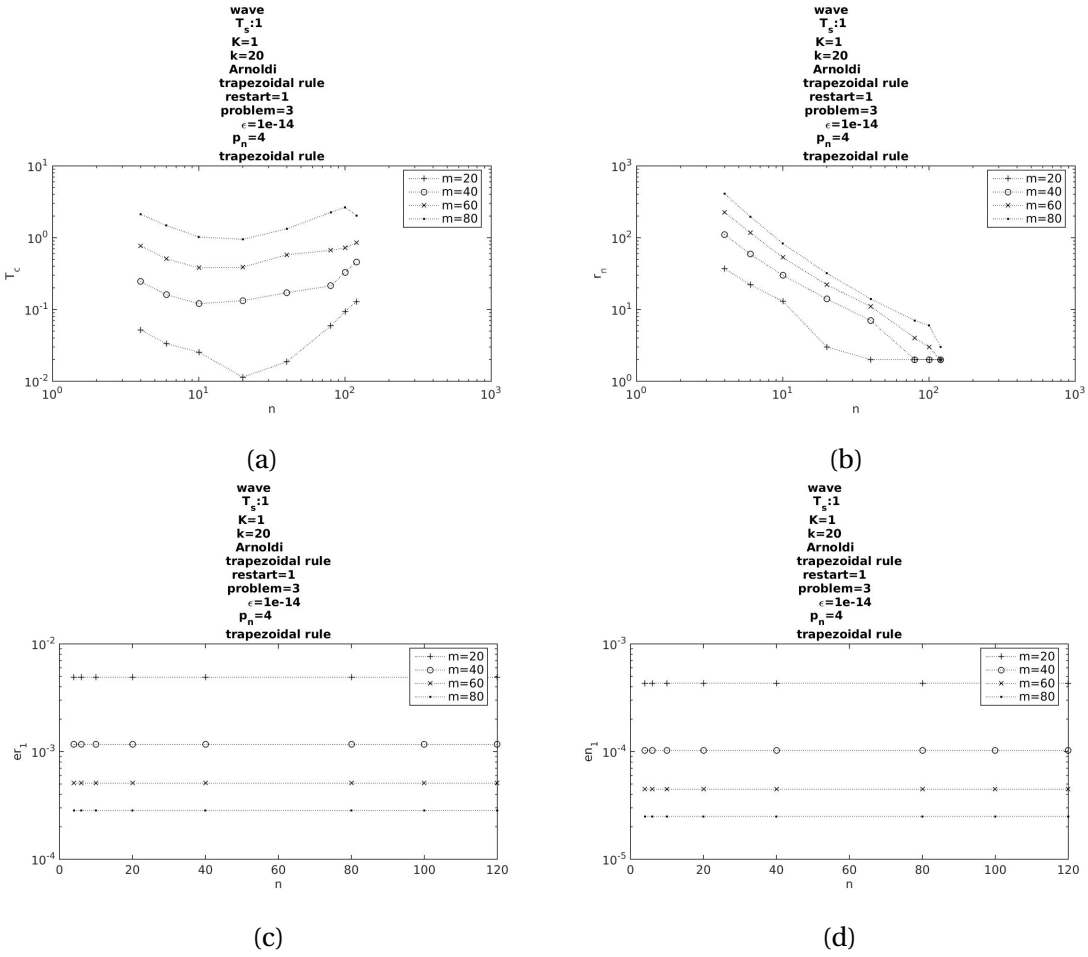


Figure 5.21: !!!!!!!!!!!!!!!!!SKRIV NOE HER!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!Kommentar her!!!!!!!!!!!!!

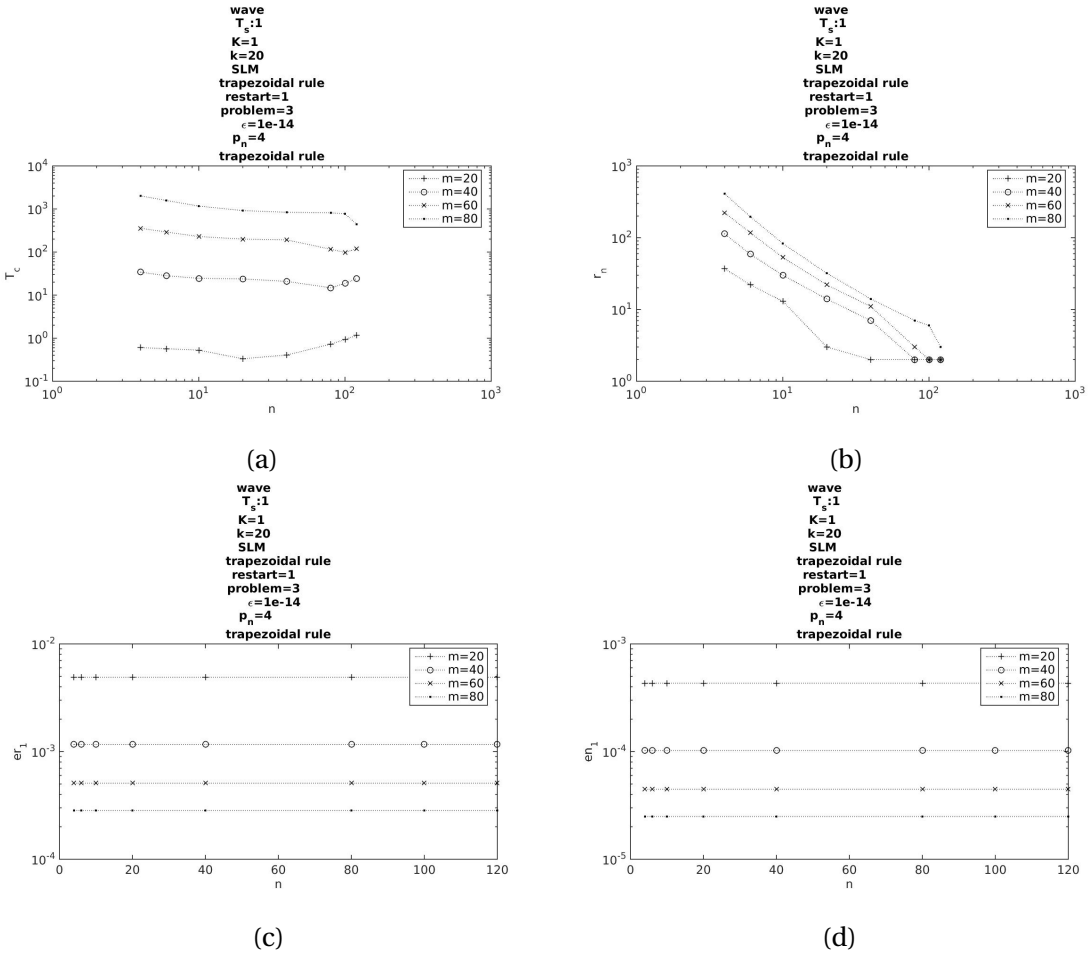


Figure 5.22: !!!!!!!!!!!!!!!!SKRIV NOE HER!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!KOmmentar her!!!!!!!!!!!!!!!

figure 5.21 shows a few different important things. First figure 5.21c shows that the error and energy does not change (much) with n, only with m. The second thing is that the number of iterations needed to converge decreases when n becomes larger. The finale thing is that there seams to be a restart variable smaller than the dimension of A, and larger than one that is optimal. !!!!!!!!!!!KOmmentar om forskjellene på Arnoldi og SLM!!!!!!!!!!!!!!!

5.7 Computation time

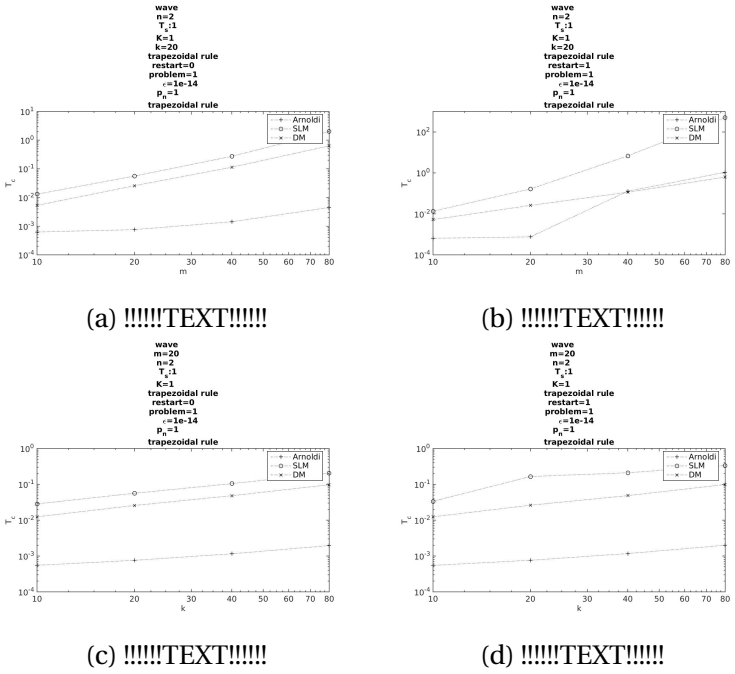
!!!!!!!!!!!!!!!!!!!!!!!!!!!Burde jeg ha med restart eller ikke på disse bildene?!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!Begge deler sikkert!!!!!!!!!!!!!!!!!

!!!!!!TEXT!!!!!!

5.7.1 Constant energy

!!!!!!TEXT!!!!!!



(a) !!!!!!TEXT!!!!!! (b) !!!!!!TEXT!!!!!!

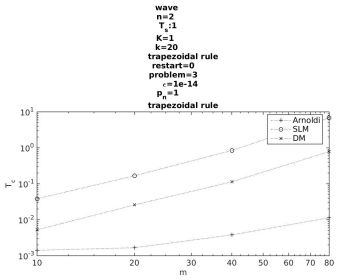
(c) !!!!!!TEXT!!!!!! (d) !!!!!!TEXT!!!!!!

Figure 5.23: !!!!!!TEXT!!!!!!

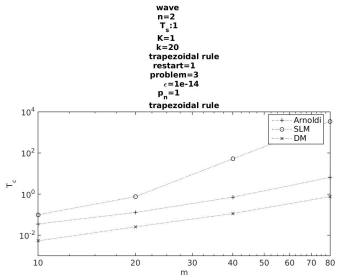
!!!!!!TEXT!!!!!!

5.7.2 Varying energy

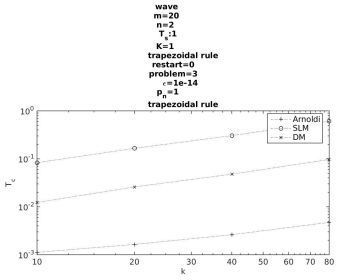
!!!!!!TEXT!!!!!!



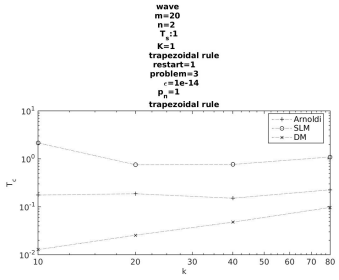
(a) !!!!!!TEXT!!!!!!



(b) !!!!!!TEXT!!!!!!



(c) !!!!!!TEXT!!!!!!

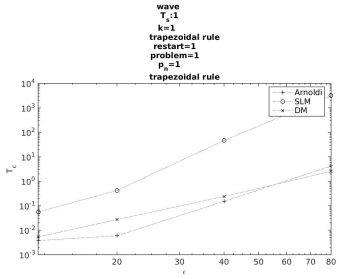


(d) !!!!!!TEXT!!!!!!

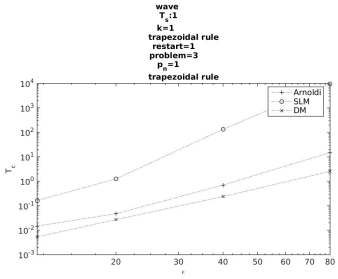
Figure 5.24: !!!!!!TEXT!!!!!!

!!!!!!TEXT!!!!!!

5.7.3 Best case for SLM and KPM



(a) !!!!!!TEXT!!!!!!



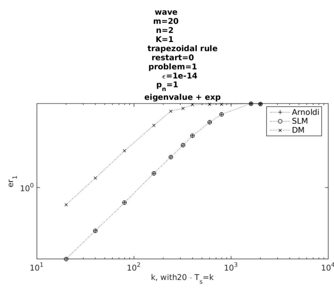
(b) !!!!!!TEXT!!!!!!

Figure 5.25: !!!!!!TEXT!!!!!!

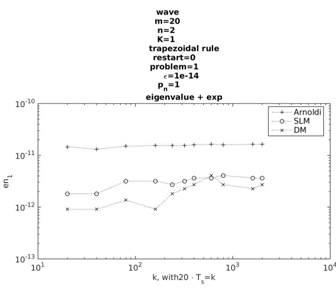
5.8 A different idea

!!!!!!!!!!!!Skrive om å bruke arnoldi og SLM med expm og veldig få restarter!!!!!!!!!!

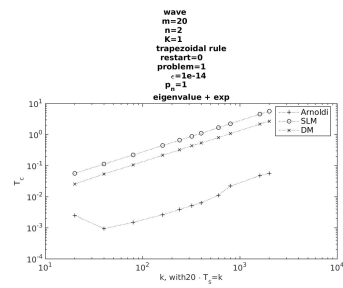
5.8.1 Without restart



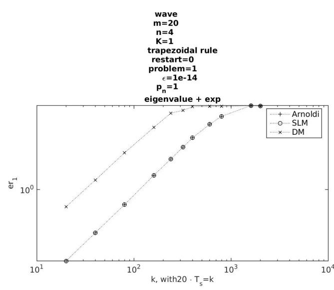
(a) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!



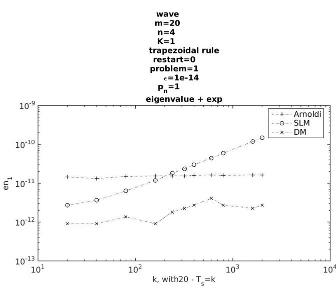
(b) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!



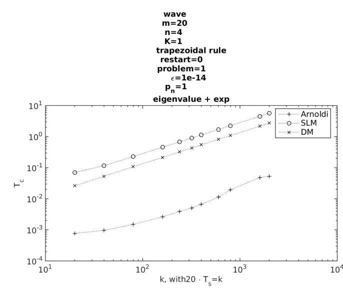
(c) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!



(d) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!



(e) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!

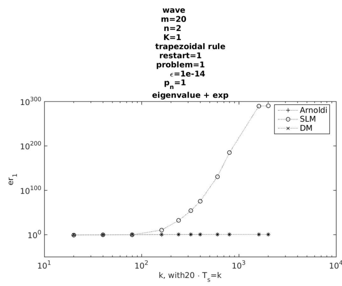


(f) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!

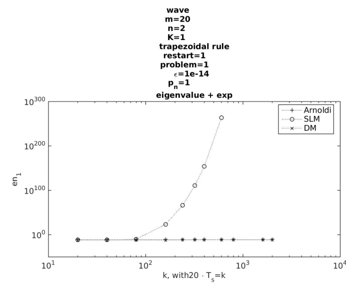
Figure 5.26: !!

5.8.2 With restart

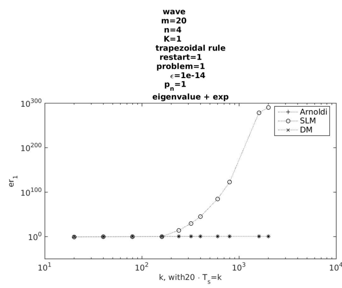
5.8.3 Without restart



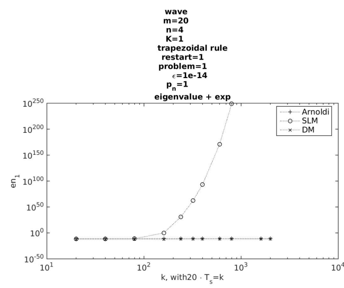
(a) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!



(b) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!



(c) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!

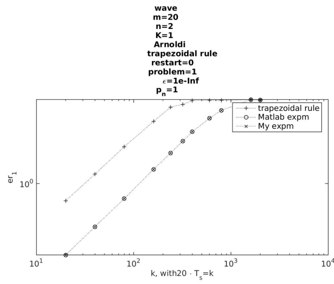


(d) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!

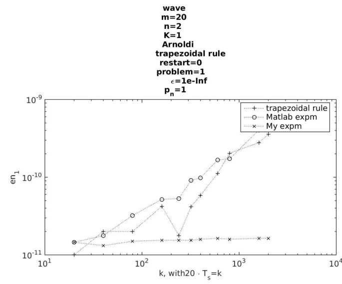
Figure 5.27: !!!

5.9 Matlabs `expm` function

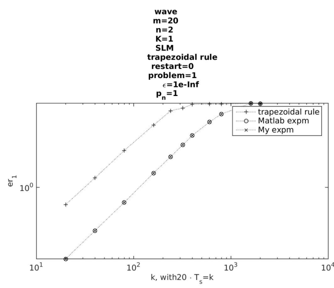
!!!!!!!!Vis at matlabs exponen er dårligere enn trapezoidal, og at det ikke er lurt å bruke!!!!!!!!!!!!



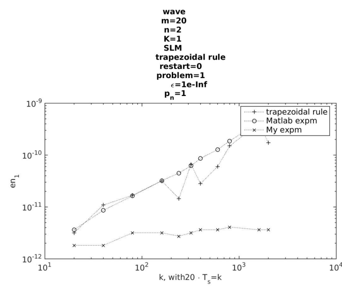
(a) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!



(b) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!



(c) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!



(d) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Figure 5.28: !!!

Bibliography

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

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

[3] Lu Li. SYMPLECTIC LANCOZS METHOD FOR SOLVING HAMILTONIAN SYSTEMS.

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

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

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

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

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

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

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