



Department of Mathematical Sciences

Projection methods for Hamiltonian systems

Sindre Eskeland

January 27, 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.

Summary and Conclusions

To be filled in!

Contents

Preface	i
Acknowledgment	ii
Summary and Conclusions	iii
1 Introduction	2
2 TTD	3
3 Theory	5
3.1 Hamiltonian and Symplectic	5
3.2 Discretization	6
3.3 Zero initial condition	8
3.4 Dividing the time domain	9
3.5 Energy	9
3.6 Integration methods	9
3.7 Abbreviations	10
3.8 Implementation	10
3.9 Solution methods	11
3.9.1 Arnoldi's Algorithm and the Krylov projection method	12
3.9.2 Symplectic Lanczos method	14
3.9.3 Direct method	16
3.9.4 Number of operations	16
4 Results	19
4.1 Convergence	19

4.1.1 Constant Energy	20
4.1.2 Varying Energy	21
4.2 Integration method	21
4.2.1 Constant energy	22
4.2.2 Varying energy	23
4.3 Energy and error	23
4.3.1 Constant energy	24
4.3.2 Varying energy	25
4.3.3 Constant energies with SLM	27
4.4 K versus k	30
4.4.1 Constant energy	30
4.4.2 Varying energy	31
4.5 How to choose ϵ	32
4.5.1 Constant energy	32
4.5.2 Varying energy	33
4.5.3 A rule based for ϵ	34
4.6 How to choose n	34
4.7 Computation time	38
4.7.1 Constant energy	39
4.7.2 Varying energy	39
4.7.3 Best case for SLM and KPM	40
4.8 A different idea	41
4.8.1 Without restart	41
4.8.2 With restart	42
4.8.3 Without restart	42
4.9 Matlabs <code>expm</code> function	42

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 Lanczos 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.

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 lineær vekst av globalfeil eller raskere vekst.
- Skrive bilde text, og fjerne labler fra bilder!
- Bevise at det er lurt å sammenligne KPM/SLM med DM?.
- SKrive mer generelt
- Er det flere figurer jeg burde ha med?
- Rekkefølgen må fikses
- Fix notasjon
- Skriv om error
- Skriv litt om antall matematiske operasjoner hver metode trenger
- Sett parantes rundt iterasjonsvariabler på z-er, og endre z-er til ϵ og δ ?
- Forandre slik at SLM og SLPN er noe greier!!!!!!!!!!!!
- Forklare hva Hamiltonsk betyr og noe slikt!
- Skrive om flere forskjellige energier

- Legg til hjelpe linjer på de plittene som trenger det, og kanskje ikke logloge alle plottene som er det
- Må ha en lignende ting for arnoldi som for SLM energi?
- Konstant energi en3 og en4 for SLM bilder?
- Må forklare det som LuLi ikke forstod (hvordan bølgeligningen er implementert, og diskretisert mer implementasjons vis)
- forklare at når semirandom brukes så måles feil og slikt på en annen måte!

Chapter 3

Theory

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 symplektisk betyr!!!!!!

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

A matrix is said to be Hamiltonian if

$$(JA)^t op = JA. \quad (3.1)$$

If a system is Hamiltonian it means that the energy in the system is preserved. The wave equation without a source term is an example of this. Some specific problems with this property will be presented later.

A matrix is symplectic if

$$A^t op JA = J. \quad (3.2)$$

!!!!!!!!!!!!!!Må skrive mer om hva dette etyr!!!!!!

3.2 Discretization

Divide each spacial direction in m pieces, with step size $h_s = 1/m$. The number of steps in time is denoted by k , making the step size in time, $h_t = 1/k$.

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

$$J_j = \begin{bmatrix} 0 & I_j \\ -I_j & 0 \end{bmatrix}. \quad (3.3)$$

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.

The first is the 2 dimensional wave equation,

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + f(t, x, y). \quad (3.4)$$

This can be discretized to be on the form of equation (1.1), with a Hamiltonian matrix

$$\begin{aligned} \tilde{A} &= \frac{2}{h_s^2} \text{gallery}'\text{poisson}', m-2) \\ A &= \begin{bmatrix} 0 & I_{\hat{m}} \\ -\tilde{A} & 0 \end{bmatrix} . \end{aligned} \quad (3.5)$$

The matrix \tilde{A} is also known as the five-point stencil[4]. This matrix will be referred to as wave. The second implemented Hamiltonian matrix is random, and given by

$$\begin{aligned} \hat{A} &= \text{rand}(\hat{m}) \\ A &= \frac{1}{2} J_{\hat{m}} \cdot (\hat{A} + \hat{A}^\top + m^2 I_{\hat{m}}). \end{aligned} \quad (3.6)$$

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.

These two matrices also has some test problems that satisfies the conditions

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

The test problems will be divided in two cases. One where the energy is constant, and one where the energy is varying.

In the case when the energy is constant and `wave` is used, the test problem is

$$\begin{aligned} u(t, x, y) &= \sin(\pi x) \sin(\pi y) \cos(\sqrt{2}\pi t) \\ u_0(x, y) &= \sin(\pi x) \sin(\pi y) \\ f(t, x, y) &= 0, \end{aligned} \tag{3.7}$$

and

$$\begin{aligned} u(t, x, y) &= \text{unknown} \\ u_0(x, y) &= \text{rand}(2(m-2)^2, 1) \\ f(t, x, y) &= 0 \end{aligned} \tag{3.8}$$

for `semirandom`.

In the case with varying energy and `wave` is used the test problem is !!!!!!Jeg bruker et annet testproblem nå!!!!!!!

$$\begin{aligned} u(t, x, y) &= (x-1)x(y-1)y(t^2 - t + 1) \\ u_0(x, y) &= (x-1)x(y-1)y \\ f(t, x, y) &= 2(y(y-1) + x(x-1)) \cdot (-(t^2 - t + 1)), \end{aligned} \tag{3.9}$$

and

$$\begin{aligned} u(t, x, y) &= \text{unknown} \\ u_0(x, y) &= 0 \\ f(t, x, y) &= \text{rand}(1, k) \cdot \text{rand}(2(m-2)^2, 1), \end{aligned} \tag{3.10}$$

for semirandom. When using one of the projection methods f needs to be separable in time and space. In the case where f is not separable it is not recommended to use this method, see [2] for more information.

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

3.3 Zero initial condition

For both KPM and SLM it is important that the initial conditions are zero. 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

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

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.

!!!!!!!!!!!!!!Skriv at det er vanskelig å sammenligne med semirandom fordi man ikke vet den ordentlige løsningen!!!!!!!!!!!!!!

3.4 Dividing the time domain

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

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 $j = 1, 2, \dots, K$ **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 .

3.5 Energy

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

The energy for a system on the form of equation 1.1 is [3]

$$\mathcal{H}(t) = \frac{1}{2} u(t)^\top J A u(t) \quad (3.11)$$

This energy is the

3.6 Integration methods

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 [7]	$U_{i+1} = U_i + h_t g\left(\frac{1}{2}(t_i + t_{i+1}), \frac{1}{2}(U_i + U_{i+1})\right)$
	$U_{i+1} = (I - \frac{Ah_t}{2})^{-1} \left(U_i + \frac{h_t}{2} (AU_i + (F_{i+1} + F_i)) \right)$
Forward Euler [8]	$U_{i+1} = U_i + h_t g(t_i, U_i)$
	$U_{i+1} = U_i + h_t (AU_i + F_i)$
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 $F_{i+\frac{1}{2}}$ we need to save twice as many points for midpoint rule, as for the other methods.

3.7 Abbreviations

Table 3.2 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
<code>restart</code>	A boolean value. If <code>restart == 1</code> , Arnoldi or symplectic Lanczos method will restart.
ϵ	If <code>restart</code> is true, restarting will commence until the change in the solution is less than ϵ

Table 3.2: Explanation of abbreviations.

3.8 Implementation

!!!!!!!!!!!!!!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!!!!!!

3.9 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. 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 equations on the form

$$\begin{aligned}\dot{u}(t) &= Au(t) + p \cdot \tilde{f}(t) \\ u(0) &= u_0\end{aligned}\tag{3.12}$$

is written as

$$\begin{aligned}\dot{u}(t) &= Au(t) + v \cdot f(t) \\ u(0) &= 0\end{aligned}\tag{3.13}$$

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 too much about these details, it is just a way to simplify the expressions.

3.9.1 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 $W_n(A, v) = \{v, Av, \dots, A^{n-1}v\} = \{v_1, v_2, \dots, v_n\}$, where $n \leq \hat{m}$. The vectors v_i together with $h_{i,j} = v_i^\top Av_j$, 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 $[v_1, v_2, \dots, v_n]$ and H_n be the $n \times n$ upper Hessenberg matrix containing all elements $(h_{i,j})_{i,j=1,\dots,n}$. Then the following holds [?]

$$\begin{aligned} AV_n &= V_n H_n + h_{n+1,n} v_{n+1} e_n^\top \\ V_n^\top A V_n &= H_n \\ V_n^\top V_n &= I_n. \end{aligned} \tag{3.14}$$

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 $\epsilon \in \mathbb{R}$.

$v_1 = v / \|v\|_2$

for $j = 1, 2, \dots, n$ **do**

 Compute $h_{i,j} = v_i^\top Av_j$, v_i for $i = 1, 2, \dots, j$

 Compute $w_j = Av_j - \sum_{i=1}^j h_{i,j} v_i$

$h_{j+1,j} = \|w_j\|_2$

if $h_{j+1,j} < \epsilon$ **then**

 STOP

end if

$v_{j+1} = w_j / h_{j+1,j}$

end for

Return H , V , v_{n+1} , $h_{n+1,n}$.

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

By using the transformation $u(t) \approx V_n z_n^i(t)$, equation (3.13) can, with the help of equation (3.14) be written as

$$\begin{aligned}\dot{z}_n^0(t) &= H_n z_n^0(t) = \|v\|_2 e_1 f(t) \\ z_n^0(0) &= 0.\end{aligned}\tag{3.15}$$

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

$$u(t) \approx V_n z_n^i(t).\tag{3.16}$$

The derivation of the method is shown in [?]. Here it is shown that larger n gives a better approximation of the solution, but larger n also gives higher computational complexity, see [?] for proof of convergence. 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.

The restart considers the difference between $V_n z_n^0(t)$ and $u(t)$, as in equation (3.17).

$$(V_{\hat{m}} z_{\hat{m}} - V_n^i z_n^i)'(t) = A(V_{\hat{m}} z_{\hat{m}} - V_n^i z_n^i)(t) - r_n^i\tag{3.17}$$

where

$$r_n^i(t) = v^{i-1} f(t) - V_n^{i-1} z_n^{i-1}(t) + A V_n^{i-1} z_n^{i-1}(t) = h_{n+1,n}^{i-1} e_n^\top z_n(t)^{i-1} v_{n+1}^{i-1}\tag{3.18}$$

This expression can be simplified by using equation (3.14) to look like equation (3.19).

$$\dot{z}_n^i(t) = H_n z_n^i(t) + h_{n+1,n}^{i-1} e_n^\top z_n^{i-1}(t), \quad i \geq 1\tag{3.19}$$

This equation can be repeatedly solved, and increase the accuracy of the approximated solution within an arbitrary constant of the true solution. This equation works for $i = 1, 2, \dots$, the 0th iteration is done by equation (3.15). It is no longer possible to use $h_{n+1,n}$ as a measure for the

error when the restart is used, so other ways of finding the error must be used.

If you want to derive this yourself, note that the z_n^1 in equation (3.19) is defined as the difference between $z_n^0(t)$ and $z_{\hat{m}}^0(t)$. The derivation of the method can be found in [?].

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

!!!!!!!!!!!!!!Flytt algoritmene!!!!!!!!!!!!!!

3.9.2 Symplectic Lanczos method

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

SLM and KPM are very similar, which is easy to see when comparing equation (3.14) and (3.20). 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

$$\begin{aligned} AS_n &= S_n H_n + h_{n+1,n} v_{n+1} e_{\hat{m}}^\top \\ J_{\hat{m}}^{-1} S_n^\top J_n A S_n &= H_n \\ S_n^\top J_n S_n &= J_{\hat{m}} \end{aligned} \tag{3.20}$$

!!!!!!!!!!!!!!Endre nevnene 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 and w .

The process of making the vectors and 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.

Algorithm 3 Symplectic Lanczos method [6], with reorthogonalization 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 $j = 1, 2, \dots, \tilde{n}$ **do**

$$v = Av_j$$

$$\delta_j = v_j^\top v$$

$$\tilde{w} = v - \delta_j v_j$$

$$\kappa_j = v_j^\top J_{\tilde{m}} v$$

$$w_j = \frac{1}{\kappa_j} \tilde{w}_j$$

$$w = Aw^j$$

$$\tilde{V}_{j-1} = [v_1, v_2, \dots, v_{j-1}, w_1, w_2, \dots, w_{j-1}]$$

$$w_j = w_j + \tilde{V}_{j-1} J_{j-1} \tilde{V}_{j-1}^\top J_{\tilde{m}} w_j$$

$$\beta = -w_j^\top J_{\tilde{m}} w$$

$$\tilde{v}_{j+1} = w - \zeta_j v_{j-1} - \beta_j v_j + \delta_j v_j$$

$$\zeta_{j+1} = \|\tilde{v}_{j+1}\|_2$$

$$v_{j+1} = \frac{1}{\zeta_{j+1}} \tilde{v}_{j+1}$$

$$\tilde{V}_j = [v_1, v_2, \dots, v_j, w_1, w_2, \dots, w_j]$$

$$v_{j+1} = v_{j+1} + \tilde{V}_j J_j \tilde{V}_j^\top J_{\tilde{m}} v_{j+1}$$

end for

$$V = [v_1, v_2, \dots, v_{\tilde{n}}, w_1, w_2, \dots, w_{\tilde{n}}]$$

$$H = \begin{bmatrix} \text{diag}([\delta_j]_{j=1}^n) & \text{tridiag}([\zeta_j]_{j=2}^n, [\beta_j]_{j=1}^n, [\zeta_j]_{j=2}^n) \\ \text{diag}([\kappa_j]_{j=1}^n) & \text{diag}([-{\delta_j}]_{j=1}^n) \end{bmatrix}$$

Return $H, V, v_{n+1}, \zeta_{n+1}$.

The reduced system, given in equation 3.21, is found by substituting $u(t) \approx S_n z(t)$, as in Arnoldi, and use equation (3.20).

$$\dot{z}^0 = H_n z_n^0 + (J_n)^{-1} S_n J_{\hat{m}} A v \quad (3.21)$$

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 between $V_n z_n(t)$ and $u(t)$, the result is given in equation (3.22).

$$\begin{aligned} \dot{z}_n^i &= H_n^i z_n^i + J_n^{-1} S_n^{i-1} \top J_{\hat{m}} \zeta_{n+1}^{i-1} v_{n+1}^{i-1} e_n^\top z_n^{i-1}, \quad i \geq 1 \\ z_n(0) &= 0 \end{aligned} \quad (3.22)$$

For the derivation of the method see [?]. As for Arnolds algorithm, $z_n^i, i \geq 1$ is renamed to be the difference between $V_n z_n(t)$ and $u(t)$.

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

3.9.3 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.

3.9.4 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.

Operation	Cost
Integration with forward Euler	$\mathcal{O}(kn^2)$
Integration with Trapezoidal or midpoint rule	$\mathcal{O}(kn^3)$
Arnoldi's algorithm	$\mathcal{O}(n^2 m)$
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 3.3: Computational cost of some mathematical operations. !!!!!!!Husk å cite alt!!!!!!

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

Table 3.4: 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!!!!!!

A table of computational cost for different operations is given in table ???. Sections ?? to 3.9.4 will contain a motivation for the results found in table ???. It is assumed that trapezoidal method is used.

!!!!!!Skriv mye her!!!!!!

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 $J^{-1} S_n J \zeta v$, 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 $n = \hat{m}$.

Chapter 4

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.

4.1 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.

4.1.1 Constant Energy

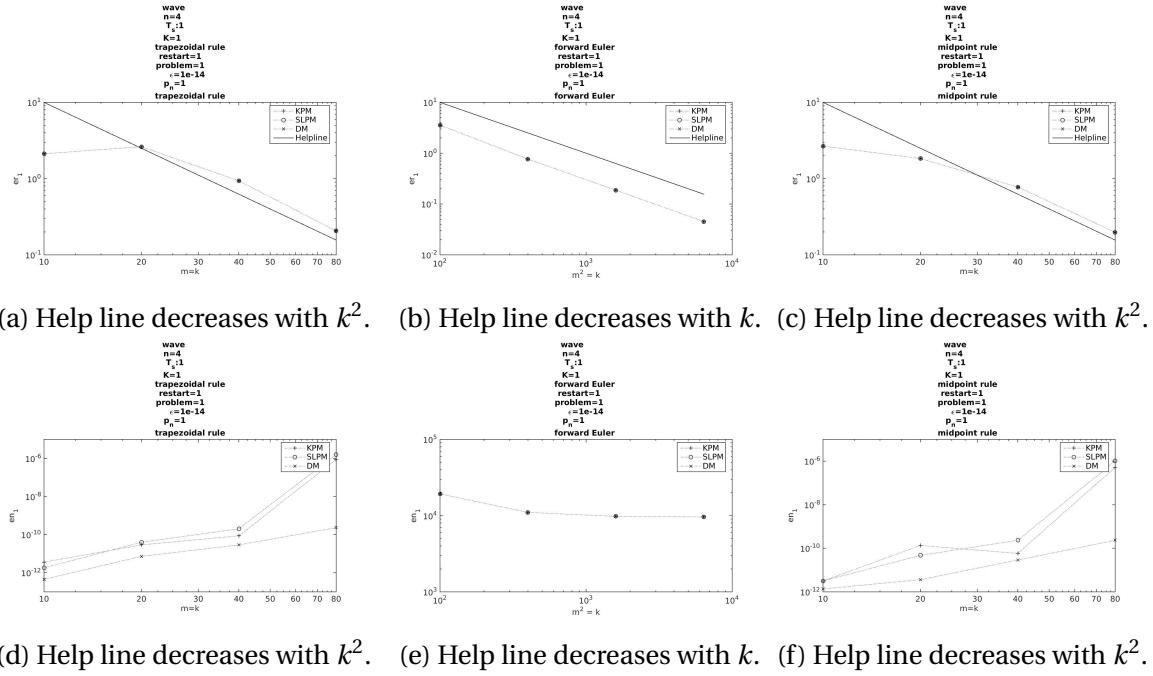


Figure 4.1: Figure of the convergence for the different integration methods. Notice that forward Euler requires a much larger k to obtain the same precision as the other methods. All methods converge with the expected rate. $n = 4$ and $\epsilon = 1e - 14$.

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.

4.1.2 Varying Energy

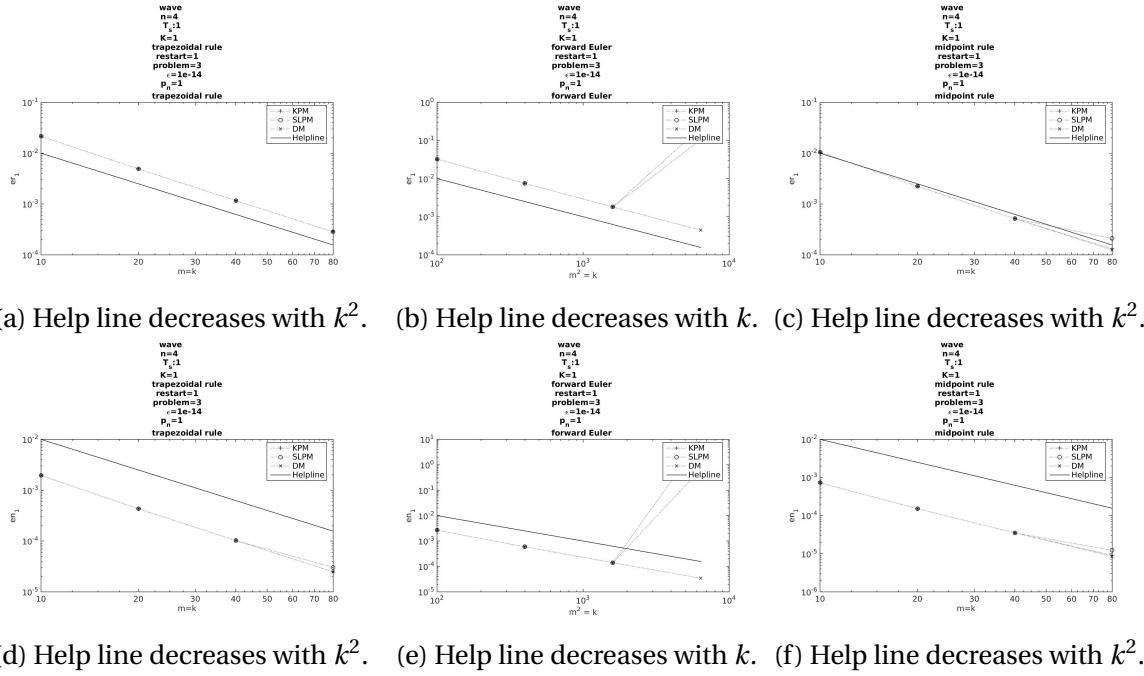


Figure 4.2: Figure of the convergence for the different integration methods. All methods converge with the expected rate.

It is clear from figure 4.1 and 4.2 that all methods can be forced to have the same convergence if ϵ is chosen to be near machine accuracy. Also note that it does not matter if the energy is constant or not, the method will converge either way. !!!!!!!Skriv om noe om energiens konvergens når bildene er ferdige!!!!!!

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

4.2 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?!!!!!!

4.2.1 Constant energy

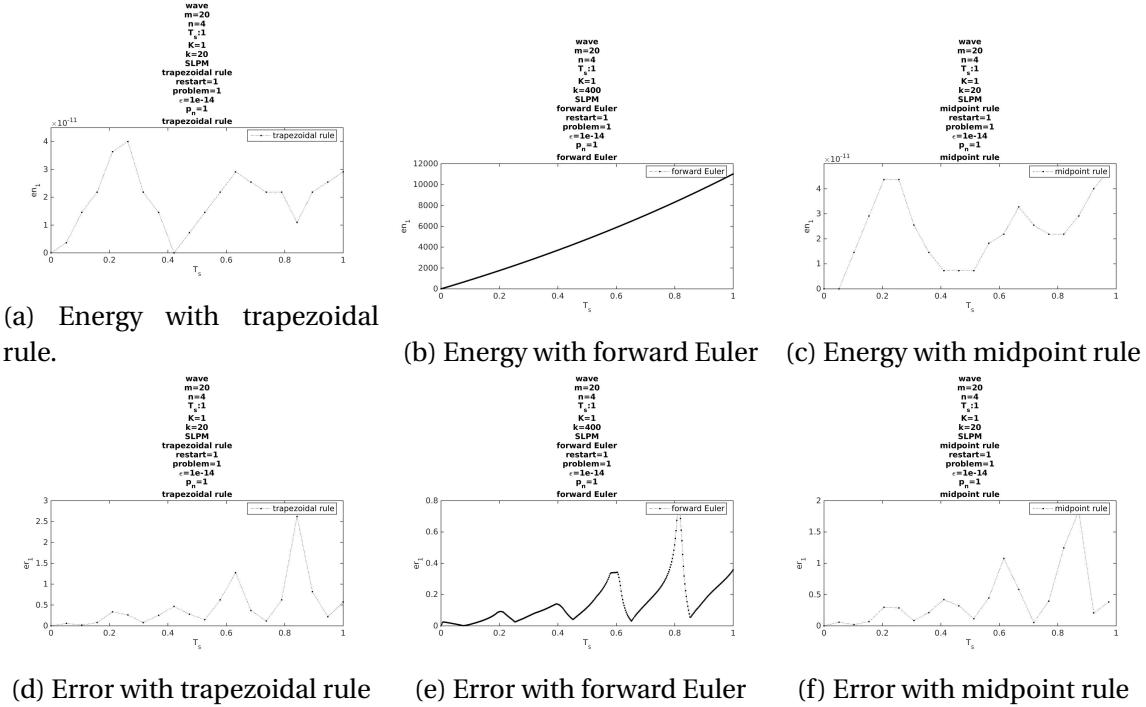


Figure 4.3: A snapshot of how energy and error changes over the simulated time with the different integration method without restart.

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 4.3d and 4.3f is the periodicity of the test problem.

4.2.2 Varying energy

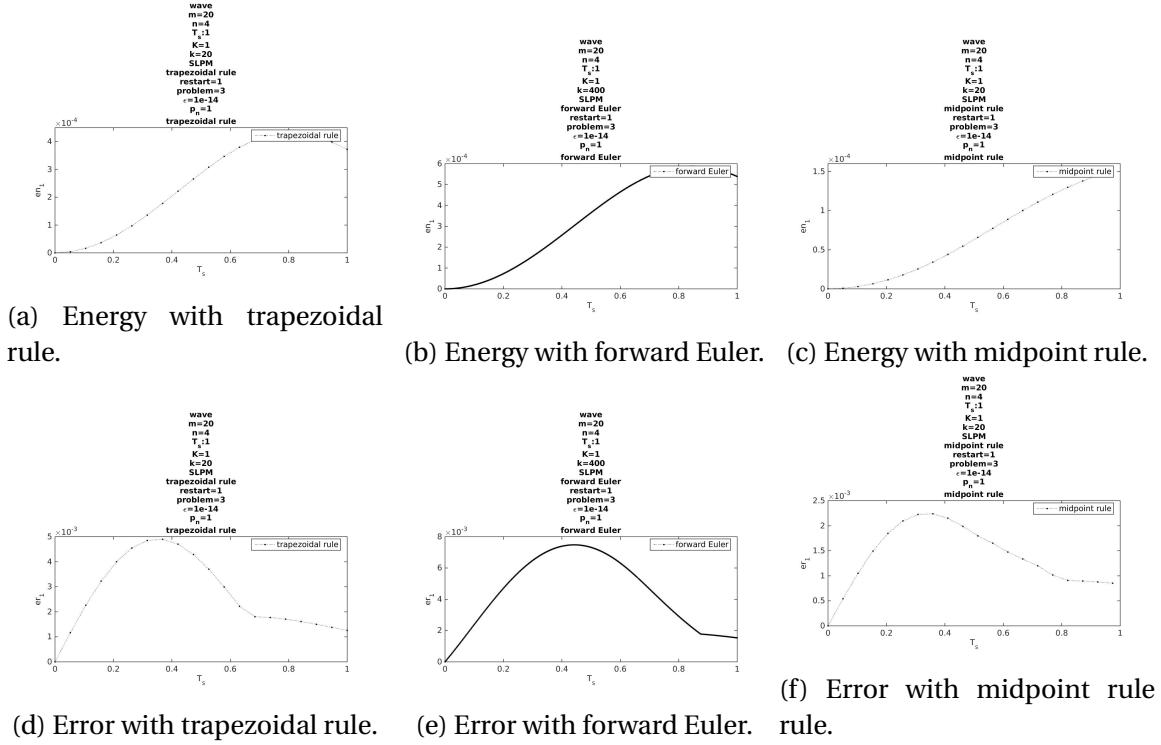


Figure 4.4: A snapshot of how energy and error changes over the simulated time with the different integration method without restart.

○

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.

4.3 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 system is non-Hamiltonian, compared to Hamiltonian systems.

4.3.1 Constant energy

Energy and error without restart

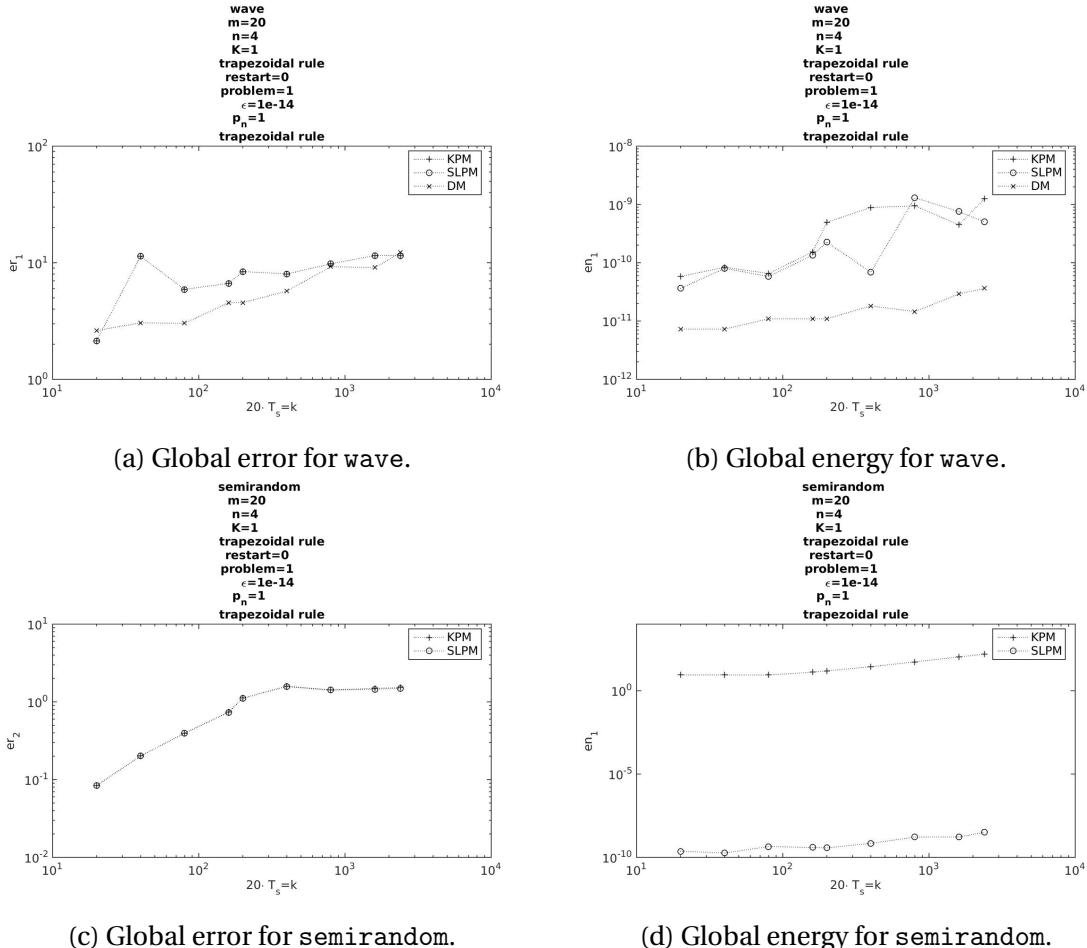


Figure 4.5: The figure shows how the global energy and global error changes as a function of simulated time, without restart. $m = 20$, $n = 8$ and trapezoidal rule is used to integrate.

!!!!!!!!!!!!Skriv om hvor utrolig kult dette bildet er!!!!!!!!!!

Energy and error with restart

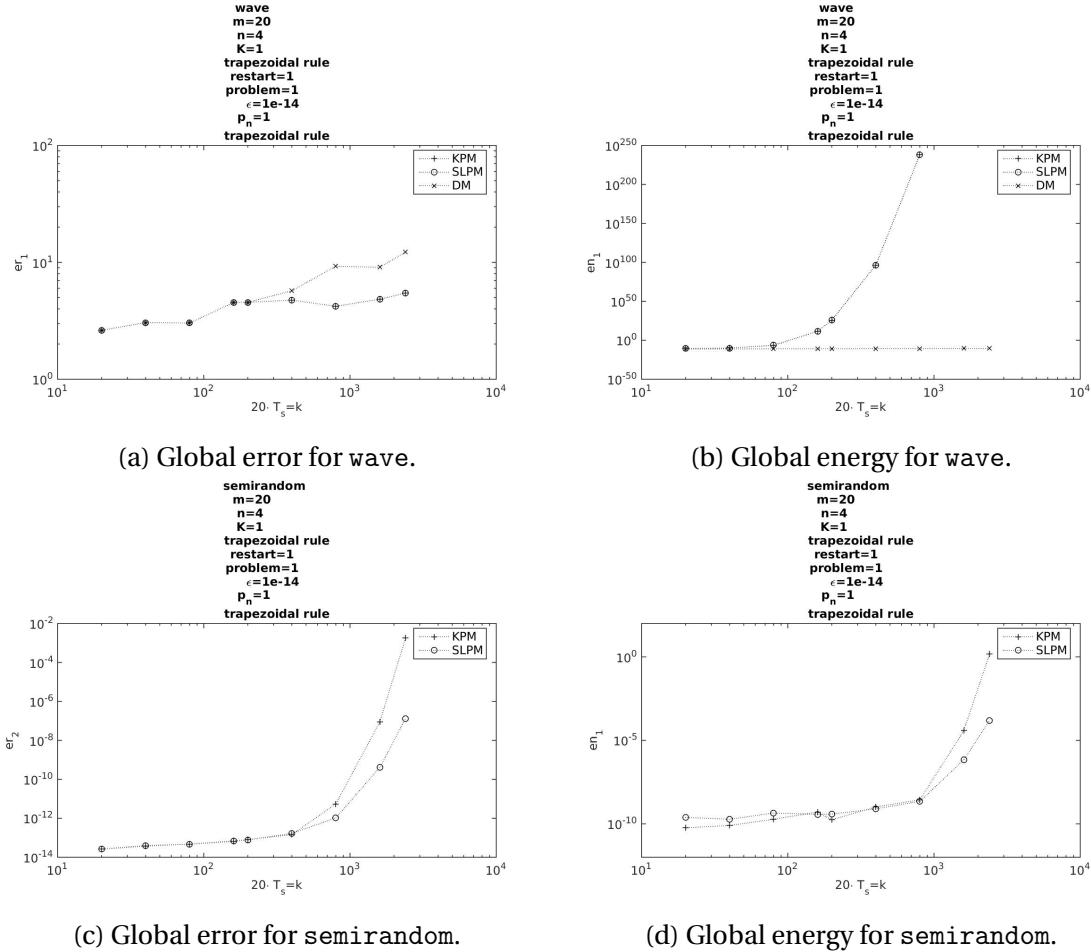
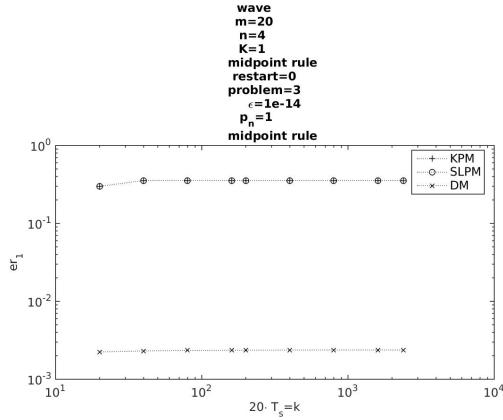


Figure 4.6: The figure shows how the global energy and global error changes as a function of simulated time with restart. $m = 20$, $n = 8$ and trapezoidal rule is used to integrate.

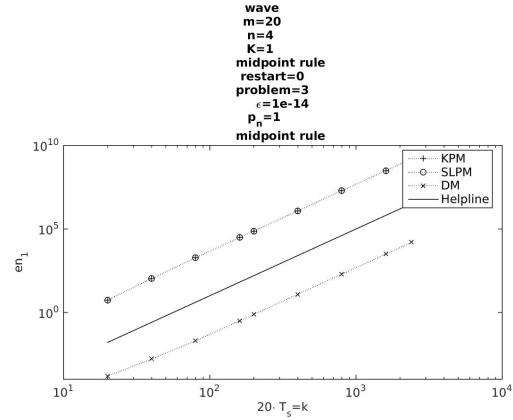
4.3.2 Varying energy

The focus has till now been Hamiltonian systems and energy preservation. This section will show how the methods work when the energy is no longer constant.

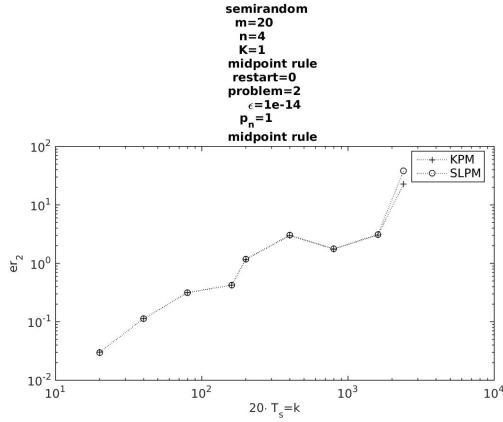
Energy and error without restart



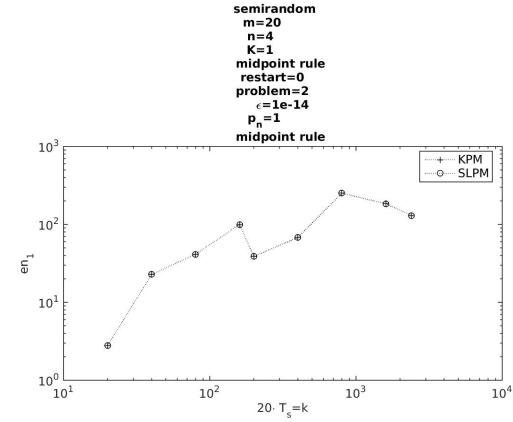
(a) Global error for wave without restart.



(b) Global energy for wave without restart.



(c) Global error for wave with restart.



(d) Global energy for wave with restart.

Figure 4.7: !!!!!!!Skriv noe her!!!!!! og på alle subcaptioner!!!!!!

Energy and error with restart

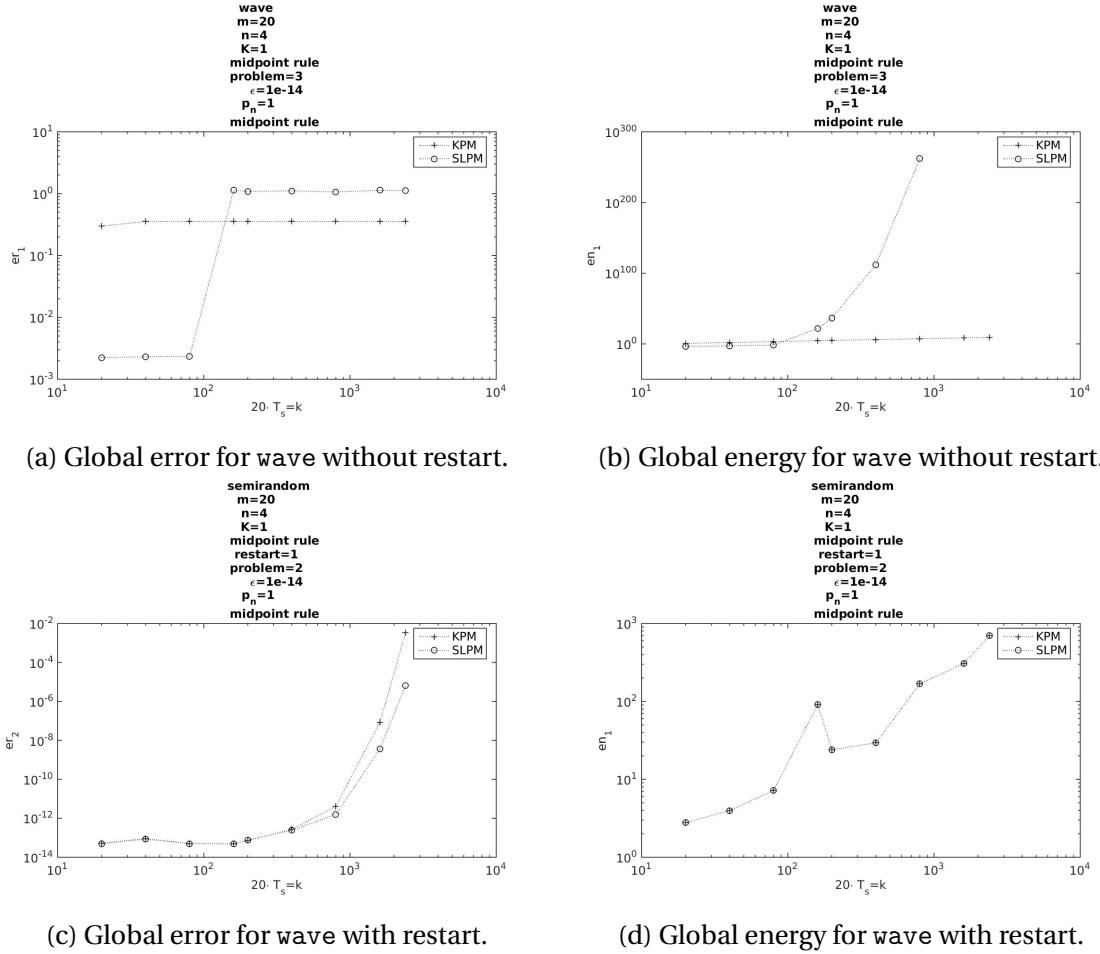


Figure 4.8: !!!!!!!Skriv noe her!!!!!! og på alle subcaptioner!!!!!!

!!!!!!Skriv noe her!!!!!!

4.3.3 Constant energies with SLM

!!!!!!Skriv mer her!!!!!!

!!!!!!Alle energier som LULi tror er konstante skal omtales her!!!!!!

Energy of SLM without restart

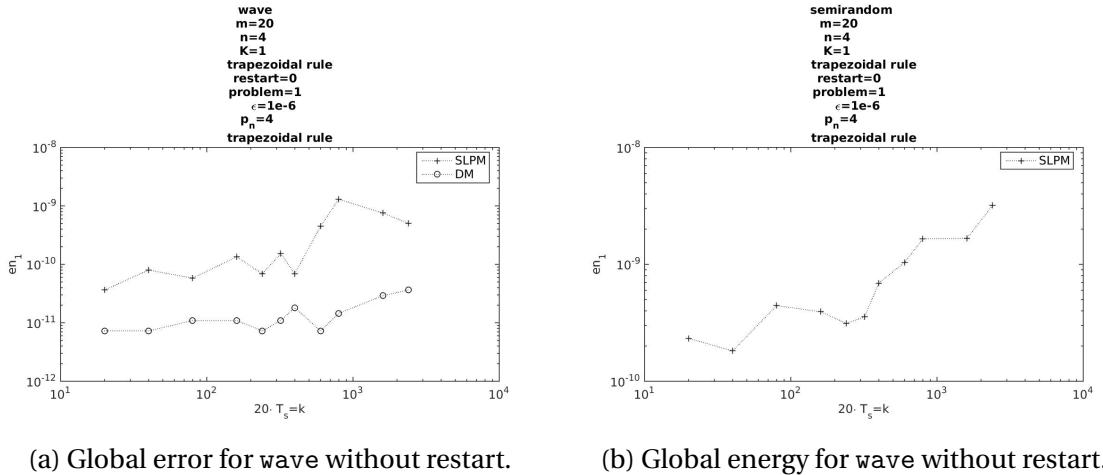


Figure 4.9: !!!!!!!Skriv noe her!!!!!! og på alle subcaptioner!!!!!!

Energy og SLM with restart

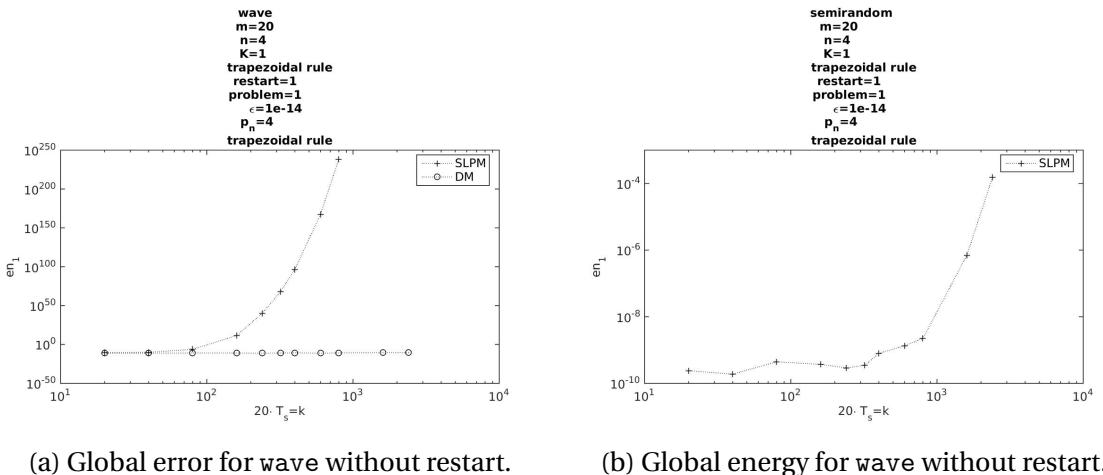


Figure 4.10: !!!!!!!Skriv noe her!!!!!! og på alle subcaptioner!!!!!!

Noe med energy3

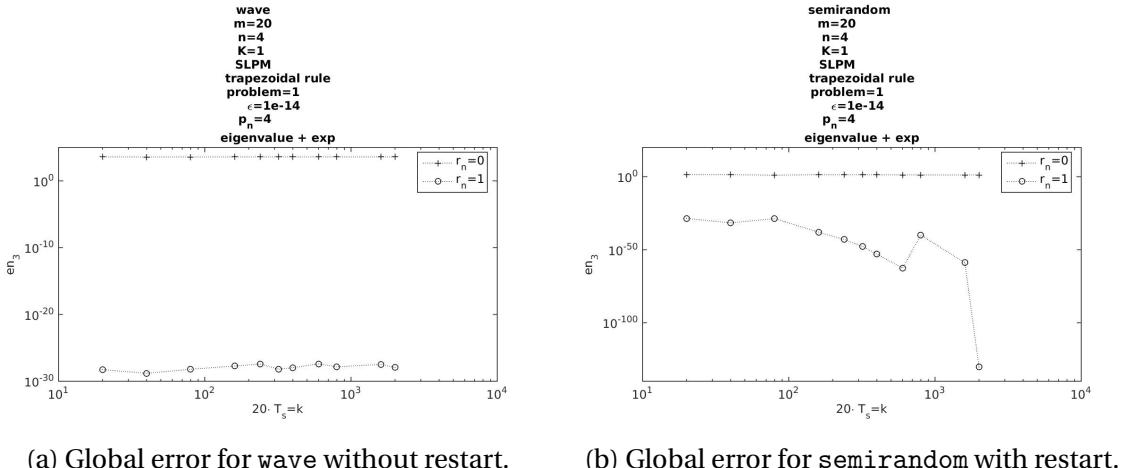


Figure 4.11: !!!!!!!Skriv noe her!!!!!! og på alle subcaptioner!!!!!!

Noe med energy4

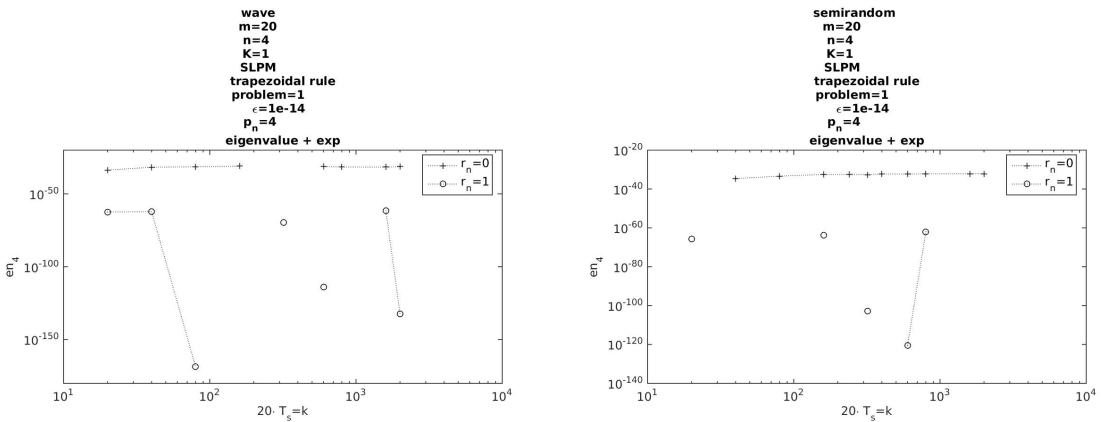


Figure 4.12: !!!!!!!Skriv noe her!!!!!! og på alle subcaptioner!!!!!!

4.4 K versus k

This method will now be tested, the goal is to see if the computation time can decrease without loosing precision in error or energy.

4.4.1 Constant energy

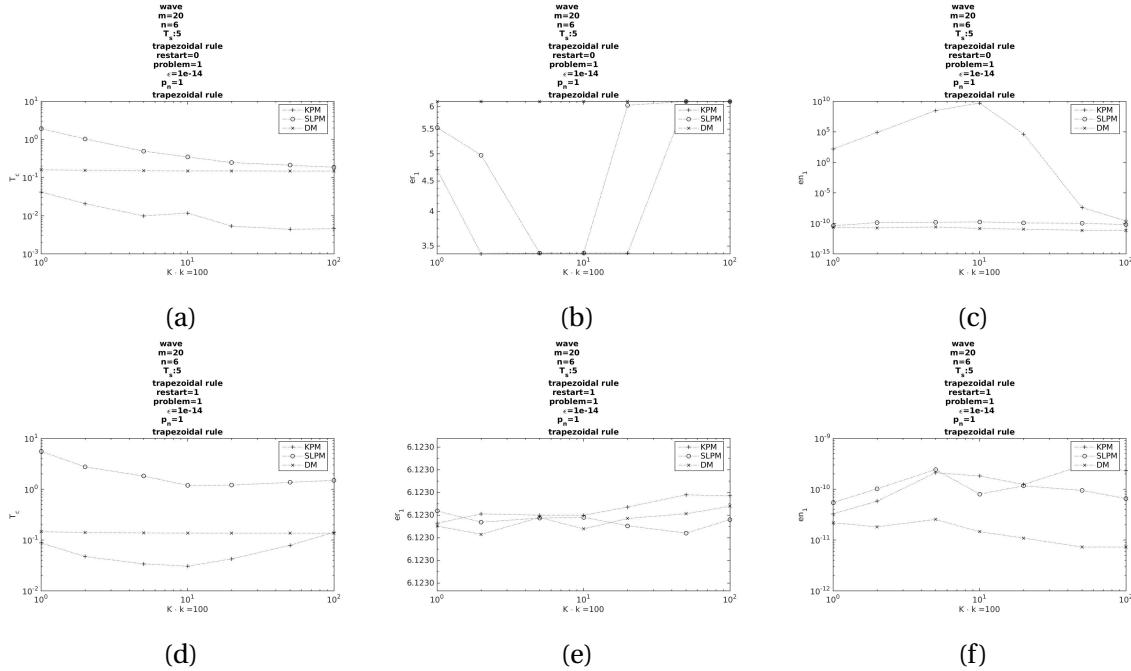


Figure 4.13: Figure showing how the different methods perform with different distribution of K and k . Notice that the product Kk is constant.

DM solving the equation with large K (and therefore smaller k) is faster, and gives no noticeable lack of precision in either energy and error. For Arnoldi and SLM there is not much to be gained computation time wise, notice also that when K is too large precision in energy and error is lost.

4.4.2 Varying energy

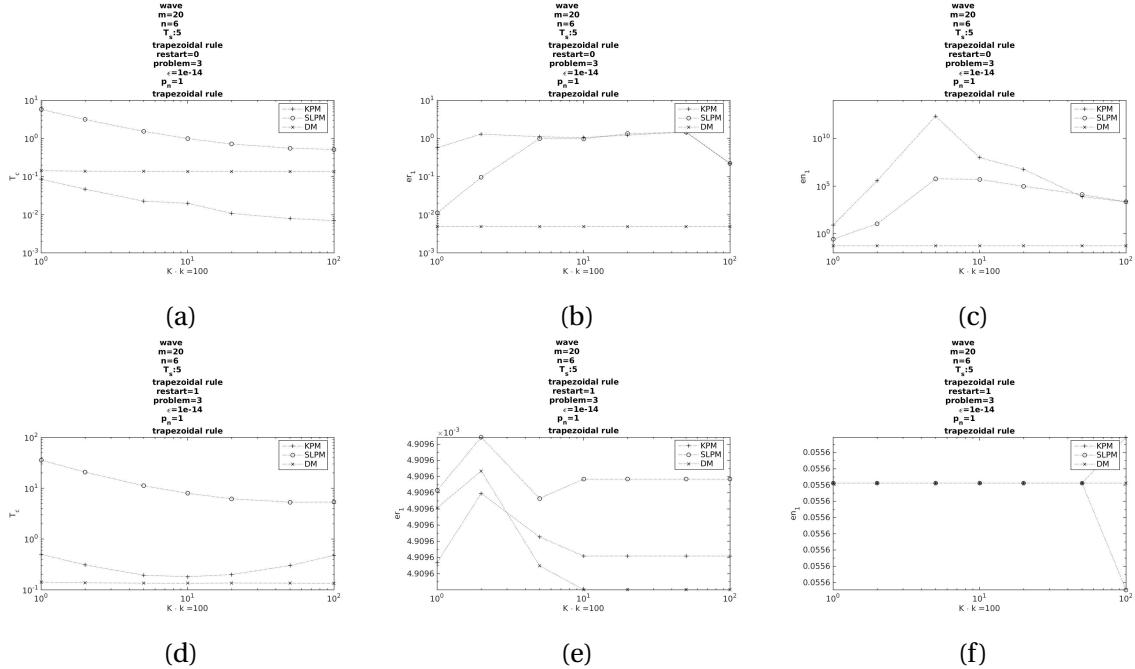


Figure 4.14: Figure showing how the different methods perform with different distribution of K and k . Notice that the product Kk is constant.

Figure 4.14 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.

4.5 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.

4.5.1 Constant energy

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

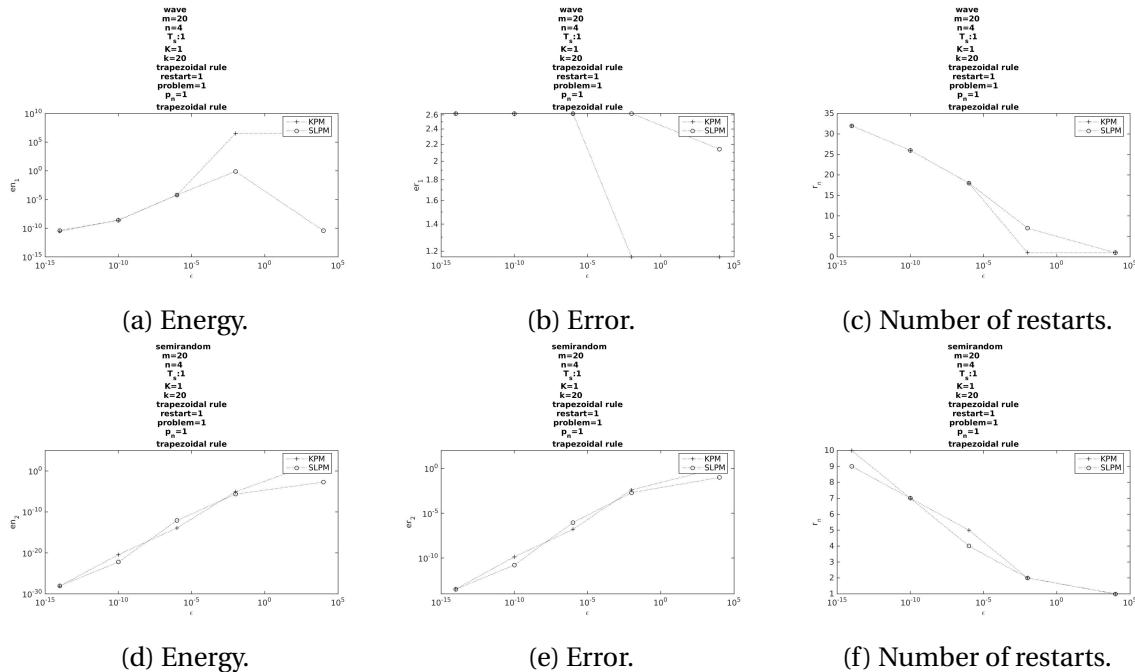


Figure 4.15: The figure shows how the different methods change the energy and error with different number of restart for `semirandom`.

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 seems 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 4.15 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 `semirandom` being a much harder equation to solve.

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

4.5.2 Varying energy

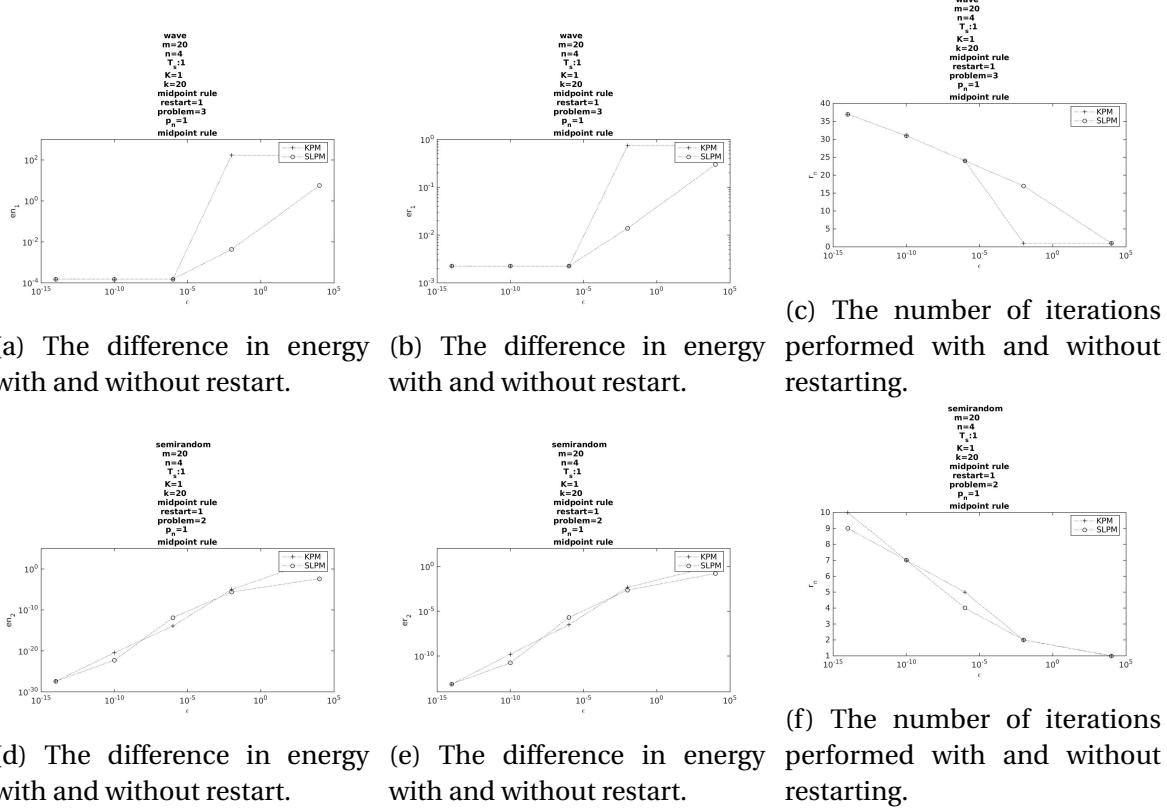


Figure 4.16: The figure shows how the different methods change the energy and error with different number of restart for wave.

The results here are very similar to the results found in section 4.5.1, except that the energy and error changes for the first points for both SLM and Arnoldi. SLM still gives the better approximation.

4.5.3 A rule based for ϵ

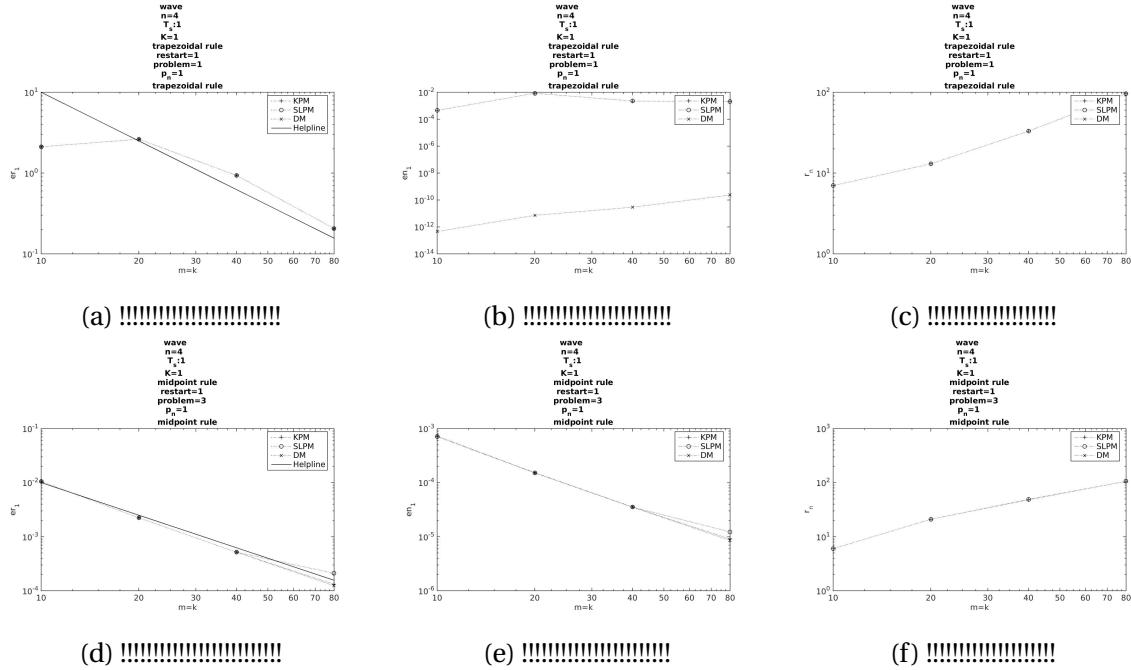


Figure 4.17: The figure shows how the different methods change the energy and error with different number of restart for wave.

4.6 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

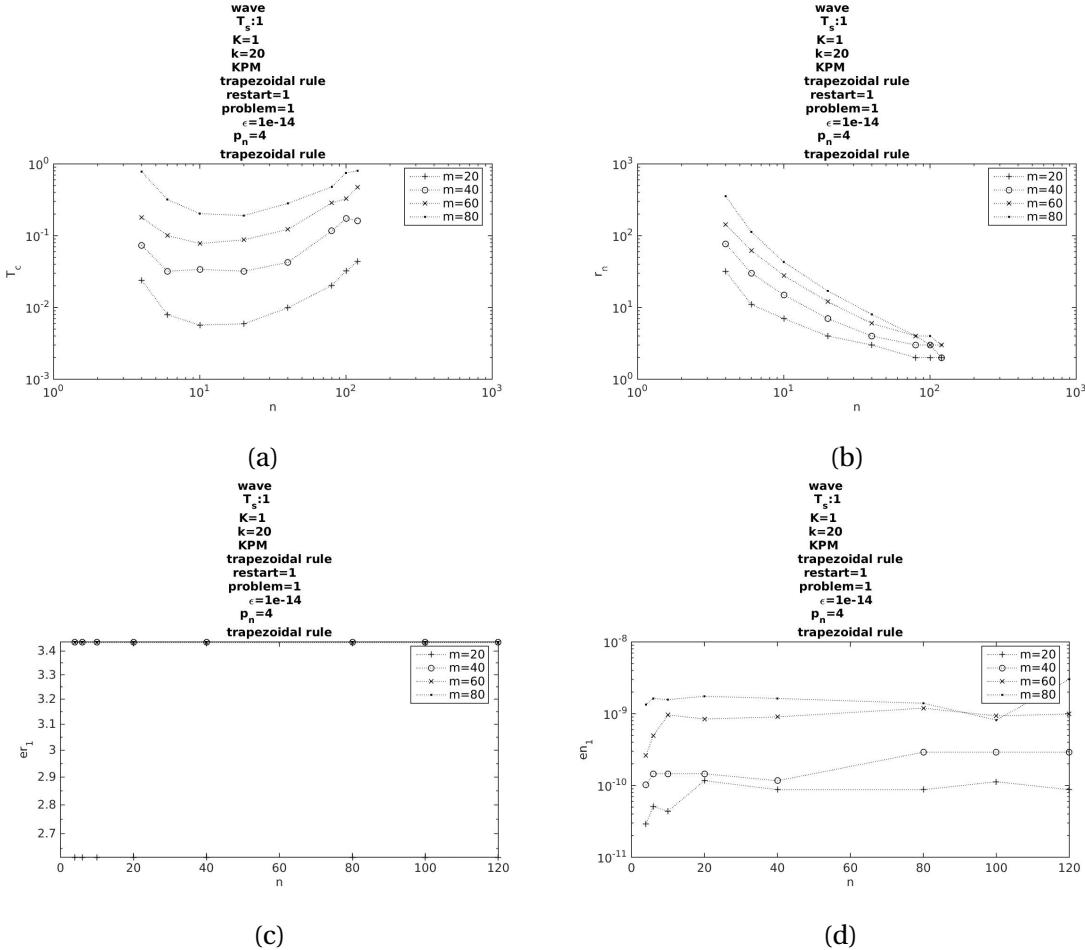


Figure 4.18: computation time, number of restarts, error and energy are all plotted here with different m and n .

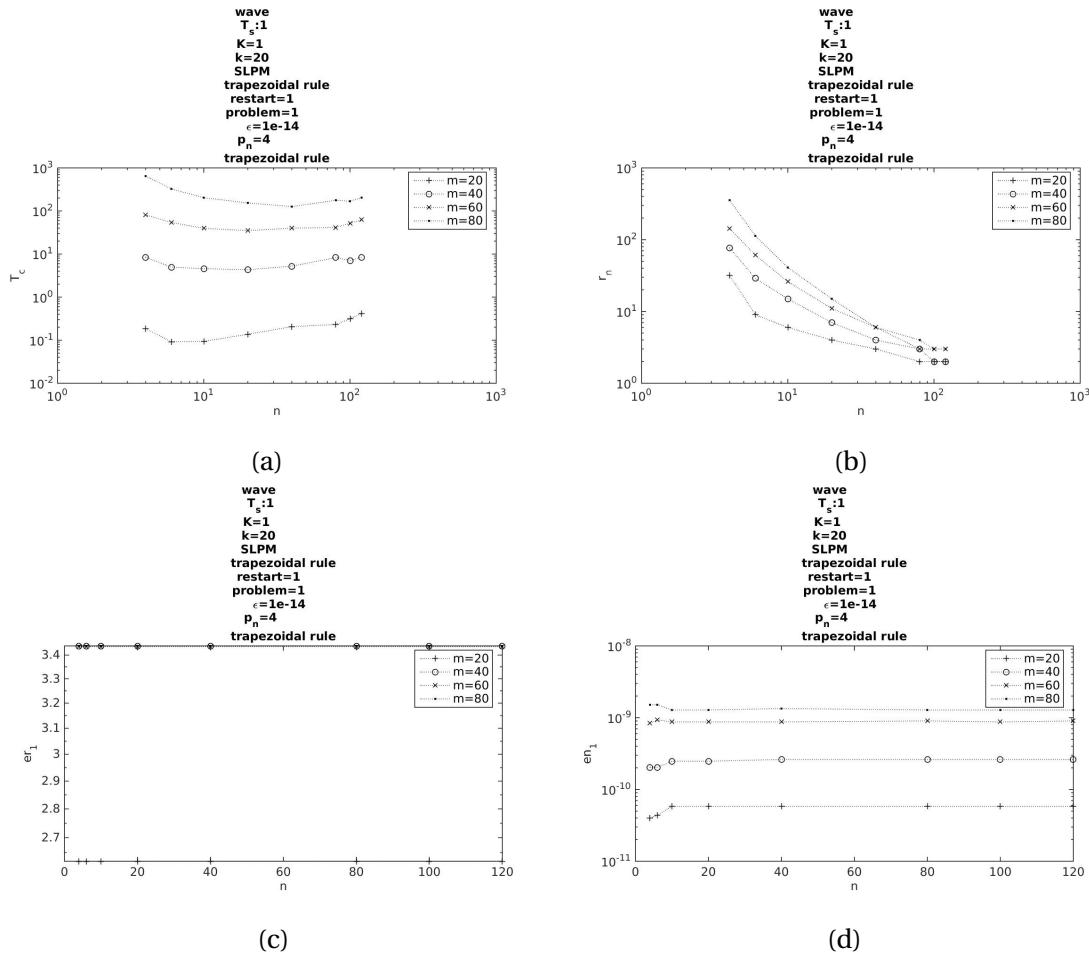


Figure 4.19: computation time, number of restarts, error and energy are all plotted here with different m and n .

!!!!!!!!!!!!!!Skriv noen kommentarer her!!!!!!!!!!!!!!

Varying energy

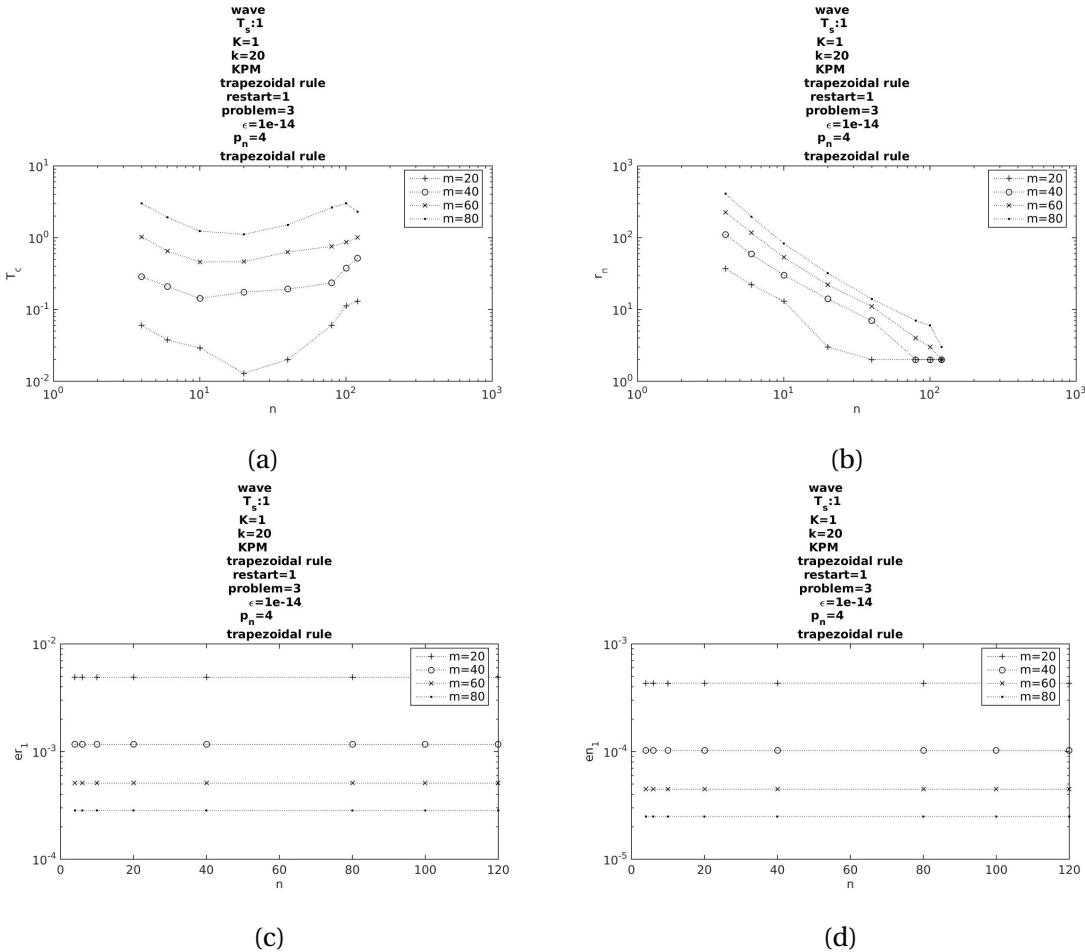


Figure 4.20: !!!!!!!SKRIV NOE HER!!!!!!

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

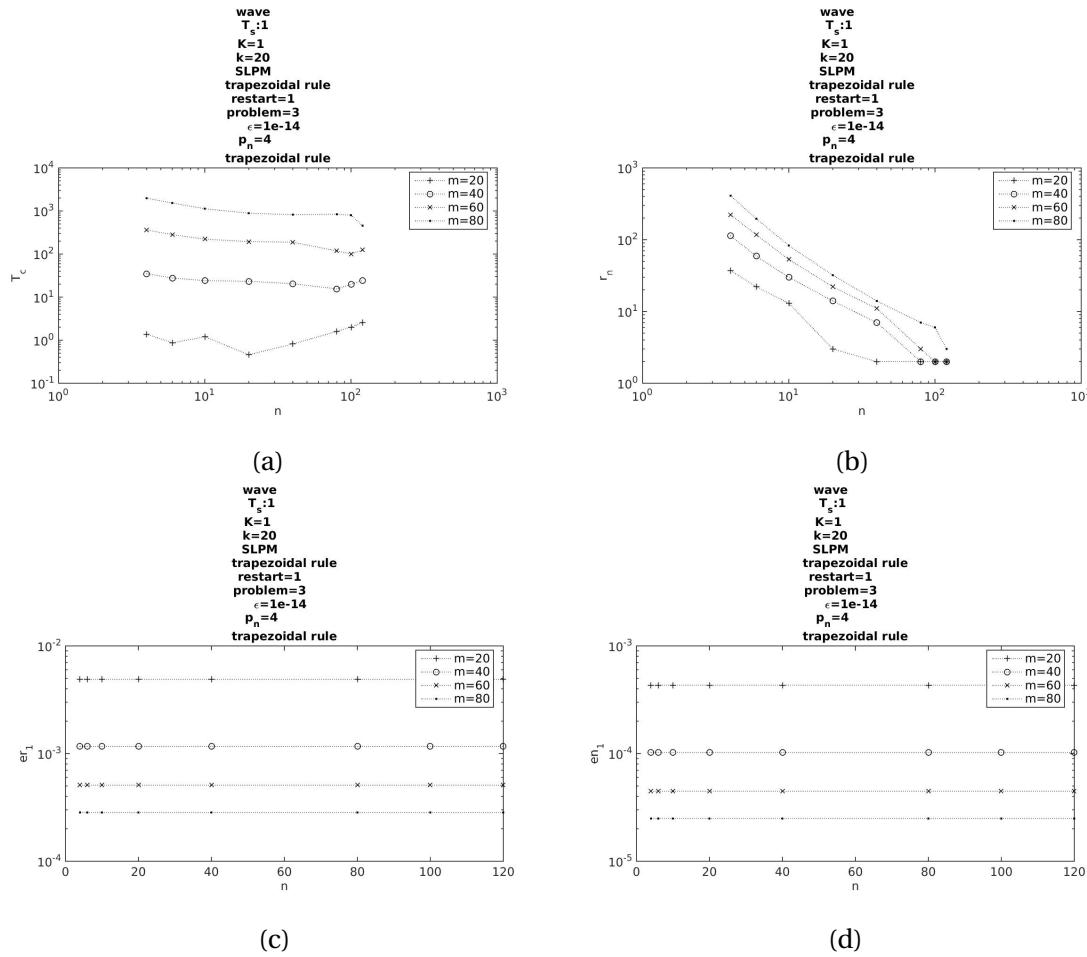


Figure 4.21: !!!!!!!SKRIV NOE HER!!!!!!

!!!!!!KOMmentar her!!!!!!

figure 4.20 shows a few different important things. First figure 4.20c 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 seems 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!!!!!!

4.7 Computation time

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

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

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

4.7.1 Constant energy

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

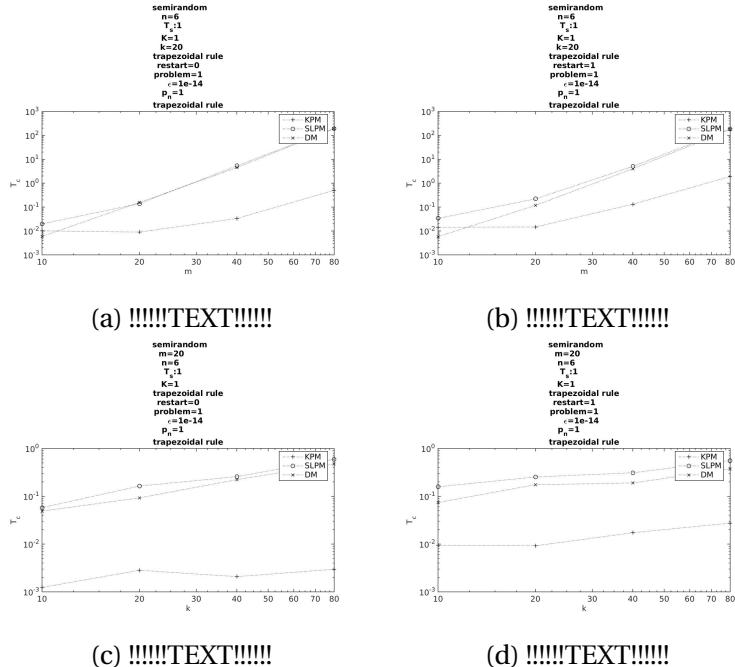


Figure 4.22:!!!!!!TEXT!!!!!!

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

4.7.2 Varying energy

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

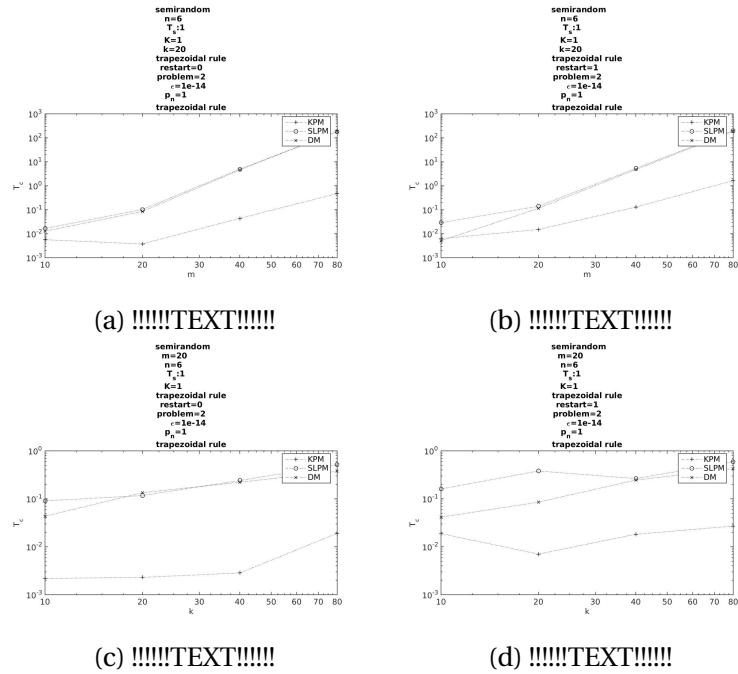


Figure 4.23: !!!!!TEXT!!!!!!

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

4.7.3 Best case for SLM and KPM



Figure 4.24: !!!!!TEXT!!!!!!

4.8 A different idea

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

4.8.1 Without restart

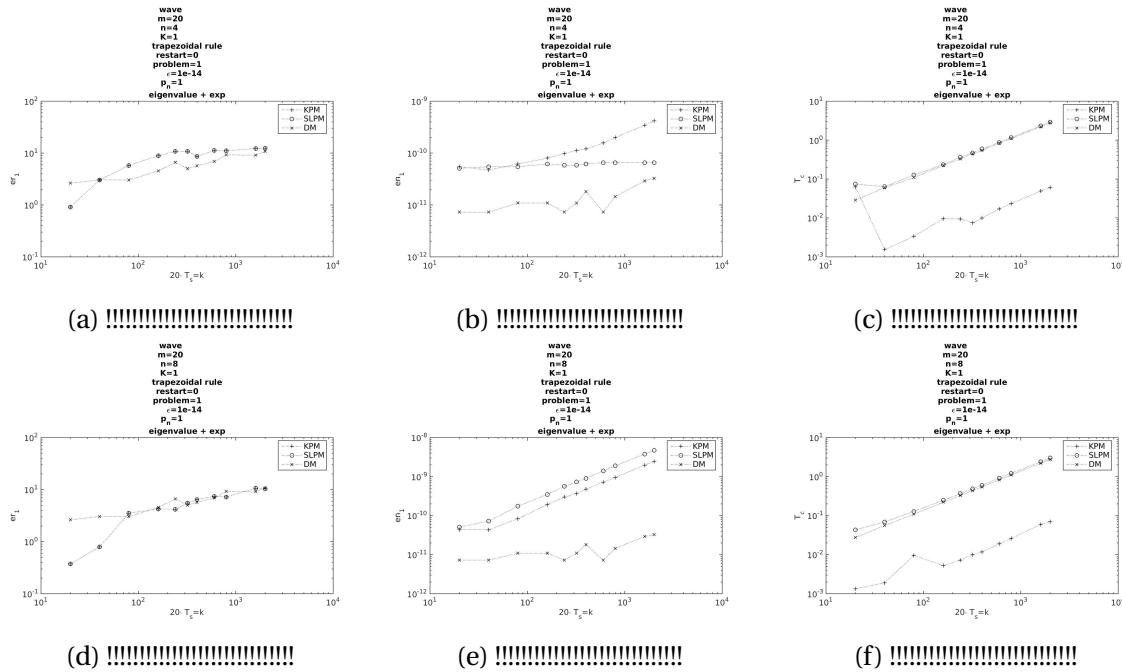


Figure 4.25: !!!!!!!

4.8.2 With restart

4.8.3 Without restart

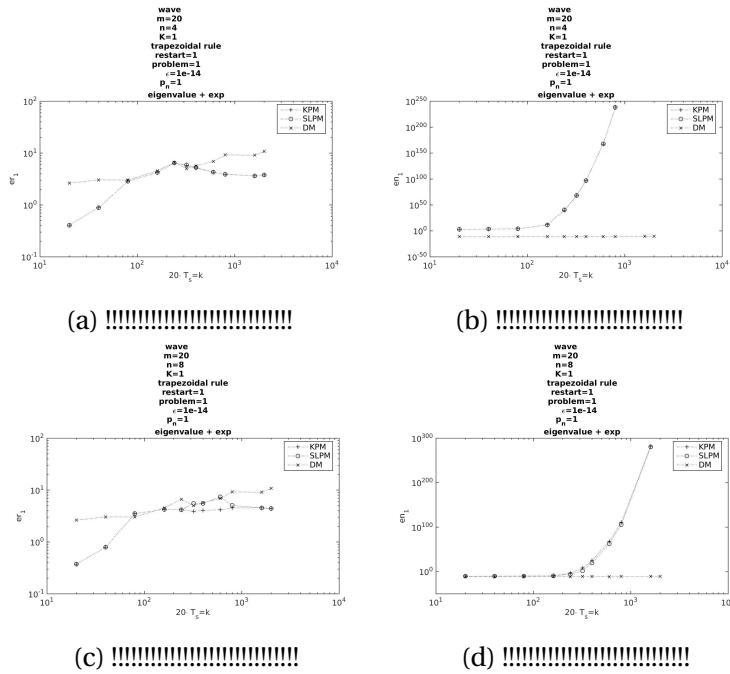


Figure 4.26: !!!!!!!!!!

4.9 Matlabs expm function

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

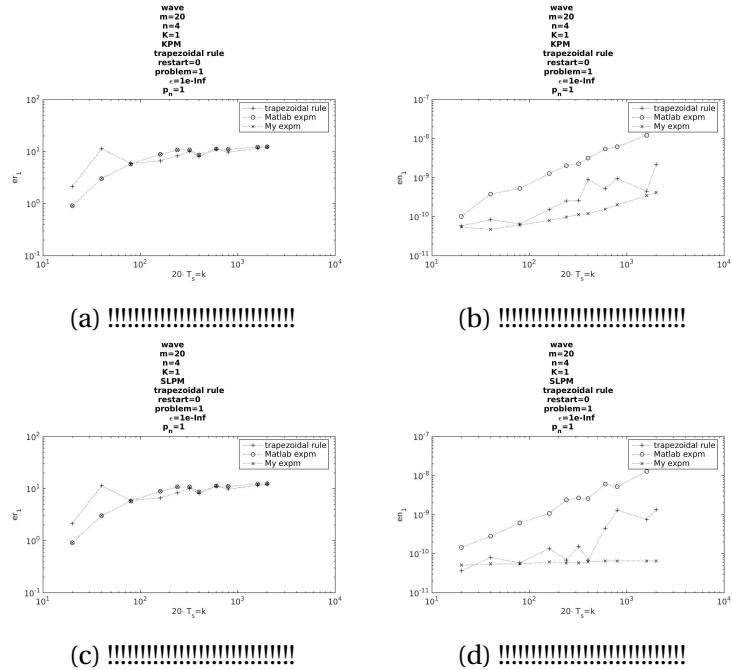


Figure 4.27: Comparison of numerical methods for wave problems.

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.