NTNU
Department of Mathematical Sciences

A Krylov projection method for the heat equation

Sindre Eskeland

September 13, 2015

PROJECT

Department of Mathematical Sciences
Norwegian University of Science and Technology

Supervisor: Professor Elena Celledoni

Preface

This is a specialization project as a part of the study program industrial mathematics. It was written during the summer of 2015. It is assumed that the reader is familiar with numerical difference methods and numerical linear algebra.

Acknowledgment

Thanks to Elena Celledoni for guiding me.

Summary and Conclusions

Solving partial differential equations with finite difference methods often requires performing operations on huge linear systems. The Krylov projection method allows the user to choose the size of the linear system by making the method iterative. The Krylov projection method was tested on the heat equation against other methods in convergence, memory demand and computation time.

The results shows that the Krylov projection method can reduce memory demand, and computation time if several processing units are used or some assumptions are met. Convergence for the Krylov projection method was found to be indistinguishable from other methods .

Contents

Chapter 1

Introduction

The aim of this report is to investigate how the Krylov projection method (KPM) can be used to solve linear differential equations on the form $q'(t) = Aq(t) + f(t)$. These problems arise for example when discretizing time dependent, linear partial differential equations with the method of lines, and have therefore a wide range of applications. KPM is an orthogonal projection technique, the method is explained in detail in section 2. See also *A krylov projection method for systems of ODEs* by E. Celledoni and I. Moret [2] for more information about the method.

In this report the heat equation will be solved with KPM. Convergence and computation time for KPM will be compared with an alternative solution technique. The alternative will be presented in section 2.5. The heat equation will be stated here with boundary conditions for future references.

$$\begin{aligned} \frac{\partial u(t,x,y)}{\partial t} - \nabla^2 u(t,x,y) &= p(t,x,y) & t \in [0,T] \\ u(0,x,y) &= 0 & , \\ u &= 0 & \text{on } \partial[0,L]\times[0,L] \end{aligned} \tag{1.1}$$

where $p(t,x,y)$ is a smooth function, and $u(t,x,y)$ is the solution sought.

Assume that the right hand side of equation (1.1) is separable, so that $p(t,x,y) = f(t)g(x,y)$, how this problem can be solved if this is not the case will be explained in section 2.4. Given the vector $v = [v_1, v_2, \cdots, v_m] \in \mathbb{R}^m$ with elements $g(x_i, y_j)$, $x_i, y_j \in [0, L]$ and the matrix A as an approximation of the Laplacian, the space-discrete version of the heat equation can be written as

$$\begin{aligned} q'(t) - Aq(t) &= f(t)v, \qquad t \in [0, T] \\ q(0) &= 0 \end{aligned} \tag{1.2}$$

where A is an $m \times m$ matrix assumed to be time independent, $f(t)$ is continuous on $[0, T]$, $v \in \mathbb{R}^m$, $q(t)$ is the sought solution vector on $[0, T]$. Note also that $f(t)$ is a scalar function, so that $f(t)v = [f(t)v_1, f(t)v_2, \cdots, f(t)v_m]$. The solution is

$$q(t) = \int_0^t \exp(A(t-s)) f(s) v \, ds.$$

More details about A, x_i, y_i and $q(t)$ will be given in section 3.

Chapter 2

Krylov subspace and methods

In this chapter some solution techniques will be derived. The Krylov subspace will be presented in section 2.1, KPM will be derived for the heat equation in section 2.2 and 2.3. In section 2.4 it is show how KPM can be used when p is not separable. The method we will compare KPM to will be introduced in section 2.5.

2.1 Krylov subspace

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 $n \leq m$. 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 1. Letting V_n be the $m \times n$ 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 $(h_{i,j})_{i,j=1,\cdots,n}$, the following holds [7]

$$AV_n = V_n H_n + h_{n+1,n} v_{n+1} e_n^\top \tag{2.1}$$

$$V_n^\top A V_n = H_n \tag{2.2}$$

$$v^{i\top} v^j = \delta_{i,j}. \tag{2.3}$$

Here, e_n is the nth canonical vector in $\mathbb{R}^n$ and $\delta_{i,j}$ is Kronecker's delta.

Algorithm 1 Arnoldi's algorithm[8]

Start with A, $v^1 = v/\|v\|_2$ and n
for $j = 1, 2, \cdots n$ **do**
  Compute $h_{i,j} = \langle Av^j, v^i \rangle$ 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 $h_{j+1,j} = 0$ **then**
    STOP
  end if
  $v^{j+1} = w_j / h_{j+1,j}$
end for

2.2 Krylov projection method

Let $z(t) = [z_1(t), z_2(t), \cdots, z_m(t)] \in \mathbb{R}^m$ be the vector satisfying $q(t) = V_m z(t)$, where $q(t)$ is from equation (1.2), and V_m is obtained by running algorithm 1 with $n = m$. We derive KPM by writing this into (1.2), that is

$$V_m z'(t) - AV_m z(t) = f(t)v$$
$$z(0) = 0.$$

Multiplying by $V_m^\top$ and using equation (2.2) gives

$$z'(t) - H_m z(t) = f(t) V_m^\top v$$
$$z(0) = 0.$$

Using equation (2.3) and $v = \|v\|_2 v^1$, we get

$$\begin{aligned} z'(t) - H_m z(t) &= \|v\|_2 e_1 f(t) \\ z(0) &= 0. \end{aligned} \tag{2.4}$$

By solving equation (2.4) for $z(t)$ and calculating $q(t) = V_m z(t)$ the solution is obtained. A step by step description is given in algorithm 2. The method will be denoted `KPM`.

Consider the residual of equation (1.2) at $q_n(t) = V_n z(t)$, that is

$$r_n(t) = f(t)v - q_n'(t) + Aq_n(t).$$

Algorithm 2 The Krylov projection method

Start with $A, f(t)$ and v.
Compute $[V_m, H_m] = \texttt{arnoldi}(A, v)$
Solve $z'(t) = H_m z(t) + f(t)\|v\|_2 e_1$ for $z(t)$
$q_m(t) \leftarrow V_m z(t)$

Since

$$r_n(t) = f(t)v - V_n z'(t) + AV_n z(t)$$

using equation (2.1) and (2.4) we get

$$r_n(t) = h_{n+1,n} e_n^\top z(t) v_{n+1}. \tag{2.5}$$

Since $h_{n+1,n} = 0$ for some $n \le m$, this shows the finite termination of the procedure.

2.3 Restarting the Krylov projection method

If $n < m$ so that $h_{n+1,n} \ne 0$, we need to restart the procedure described above. Consider first the following equation

$$\begin{aligned} (q-q_n)'(t) - A(q-q_n)(t) &= r_n(t) \\ (q-q_n)(0) &= 0 \end{aligned} \tag{2.6}$$

where $r_n(t)$ is as in equation (2.5). Solving this equation for $(q-q_n)(t)$ can improve the approximation of $q(t)$ via iterative refinement. Equation (2.6) is of the same form as equation (2.4). We derive KPM as before, by writing $q(t) = V_m z(t)$ and $q_n(t) = \tilde{V}_n \tilde{\zeta}(t)$. Let $\tilde{V}_n$ be equal to V_n, but with zeros-columns added to make it the same size as V_m. We then get

$$\begin{aligned} (V_m z - \tilde{V}_n \tilde{\zeta})'(t) - A(V_m z - \tilde{V}_n \tilde{\zeta})(t) &= h_{n+1,n} e_n^\top \tilde{\zeta}(t) v^{n+1} \\ (z - \tilde{\zeta})(0) &= 0. \end{aligned}$$

Multiplying by $V_m^\top$ and using equation (2.2) gives

$$\begin{aligned} (z - \tilde{\zeta})'(t) - \tilde{H}_n(z - \tilde{\zeta})(t) &= V_m^\top h_{n+1,n} e_n^\top \tilde{\zeta}(t) v^{n+1} \\ (z - \tilde{\zeta})(0) &= 0. \end{aligned}$$

Let $\tilde{\tilde{\xi}}(t) = (z - \tilde{\zeta})(t)$, and simplify

$$\tilde{\tilde{\xi}}'(t) - \tilde{H}_n \tilde{\tilde{\xi}}(t) = h_{n+1,n} e_n^\top \tilde{\zeta}(t)$$
$$\tilde{\tilde{\xi}}(0) = 0.$$

Drop all except the n first rows of $\tilde{\tilde{\xi}}(t)$, and we are left with

$$\begin{aligned} \xi'(t) - H_n \xi(t) &= h_{n+1,n} e_n^\top \zeta(t) \\ \xi(0) &= 0. \end{aligned} \tag{2.7}$$

Each restart we generate a new Krylov subspace $W_n(A, v^{n+1})$, solve equation (2.7) for $\xi(t)$ and obtain $q_n(t) = V_n \xi(t)$. By summing together $q_n(t)$, we converge towards $q(t)$. Note that the current value of $\zeta(t)$ equals the previous value of $\xi(t)$, and that $h_{n+1,n}$ is from the previous H_n. See algorithm 3 for a step by step description. We will call n a restart variable, and denote the method with `KPM`(n).

Algorithm 3 The Krylov projection method with restart

Start with $A, f(t), v$, n and $i = 0$
Compute $[V_n, H_n, h^i_{n+1,n}, v^{n+1}] = \texttt{arnoldi}(A, v)$
Solve $z'(t) = H_n z(t) + f(t)\|v\|_2 e_1$ for $z(t)$
$q_n(t) \leftarrow V_n z(t)$
$\xi_i(t) \leftarrow z(t)$
while convergence criterion not satisfied **do**
  $i \leftarrow i + 1$
  Compute $[V_n, H_n, h^i_{n+1,n}, v^{n+1}] = \texttt{arnoldi}(A, v^{n+1}, n)$
  Solve $\xi_i'(t) = H_n \xi_i(t) + h^{i-1}_{n+1,n} e_n^\top \xi_{i-1}(t)$ for $\xi_i(t)$
  $q_n(t) \leftarrow q_n(t) + V_n \xi_i(t)$
end while

2.4 When p is not seperable

Let $P(t)$ be a vector consisting of elements $p(t, x_i, y_j)$, so that $P(t) = [P_1(t), P_2(t), \cdots, P_m(t)]$. Writing equation (1.2) as

$$\begin{aligned} q_j'(t) - Aq_j(t) &= P_j(t)e_j \\ q_j(0) &= 0 \\ q(t) &= \sum_{i=1}^{m} q_j(t), \end{aligned} \tag{2.8}$$

where e_j is the jth canonical vector in $\mathbb{R}^m$, we can then solve the original equation without requiring separability. An important thing to note here is the need for parallel processing power, since m problems must be solved and not just one.

2.5 Direct method

Since the objective of this report is to measure how KPM performs, we need something to compare it to. For this, solve

$$q'(t) - Aq(t) = P(t) \tag{2.9}$$

for $q(t)$, without using KPM. Let $P(t)$ be as in section 2.4 and denote the method `DM` for direct method. Note that it is not possible to use more than one processing unit on this problem.

Chapter 3

Implementation

In this chapter the implementation of the methods will be explained. Discretizing in space and time will be done in section 3.1 and 3.2 respectively. In section 3.3 we present what to measure, and how, together with some information about computers and programs used to generate data. In section 3.4 some test problems are stated.

3.1 Discretization in space

Consider the spacial square $[0,1]\times[0,1]$ and divide each direction into $\rho+2$ pieces, with each piece having the length $h_s = 1/(\rho+1)$. Since the boundary conditions are known, there are only ρ unknown numbers in each direction. Let $x_i = h_s \cdot i$ and $y_j = h_s \cdot j$, with $i, j = 1, 2, \cdots, \rho+1$. Let $v_{i+\rho j} = g(x_i, y_j)$, and $P(t)_{i+\rho j} = p(t, x_i, y_j)$, where $i, j = 1, 2, \cdots, \rho+1$. The Laplacian will be approximated by the five point formula given as the matrix A,

$$A = \frac{1}{h_s^2}\begin{bmatrix} T & I & & & \\ I & T & I & & \\ & \ddots & \ddots & \ddots & \\ & & I & T & I \\ & & & I & T \end{bmatrix}, T = \begin{bmatrix} -4 & 1 & & & \\ 1 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 1 \\ & & & 1 & -4 \end{bmatrix}, I = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix} [4]. \tag{3.1}$$

This is a second order approximation. Notice that $m = \rho^2$.

3.2 Discretisation in time

Consider the time domain $t \in [0,1]$, and divide it in k pieces, giving each piece the length $h_t = 1/(k-1)$. Let $t_l = h_t \cdot l$, with $l = 0, 1, \cdots, k-1$.

Algorithm 2, 3 and equation 2.9 all contain a differential equation in time. The trapezoidal rule,

$$\int_a^b f(t)dt \approx \frac{h}{2}\sum_{l=1}^{N}(f(t_{l+1}) + f(t_l))[1] \tag{3.2}$$

will be used to solve these equations. The iteration scheme will only be derived for equation (1.2), but this it is easily generalizable to the other differential equations discussed. To obtain the iteration scheme, write q instead of f, use equation (1.1), and insert the numerical simplifications from earlier in this section and from section 3.1, that is

$$q(t_{l+1}) - q(t_l) = \int_{t_l}^{t_{l+1}} \frac{dq}{dt}dt \approx \frac{h}{2}(Aq(t_{l+1}) + f(t_{l+1})v + Aq(t_l) + f(t_l)v). \tag{3.3}$$

Solving for $q(t_{l+1})$ gives the Crank-Nicholson scheme for the heat equation:

$$q(t_{l+1}) \approx (I - h_t/2A)^{-1}(q(t_l) + \frac{h_t}{2}(Aq(t_l) + f(t_l)v + f(t_{l+1})v)) \tag{3.4}$$

This is a second order method. Because of this discretizing, $P(t)$ and $q(t)$ will be saved as $m \times k$ matrices, and $f(t)$ as a k vector, in the points t_l with $l = 0, 1, \cdots, k-1$. An implementation of equation (3.4) is given in algorithm 4.

Algorithm 4 Integration

Start with A, $f(t)$, v, h_t and k
$invmat = (I - h_t/(2A))^{-1}$
$q(t_1) = invmat \cdot (h_t/2 \cdot (f(t_1) + f(t_0)))$
for $l = 1, 2, \cdots k$ **do**
  $q(t_{l+1}) = invmat \cdot (q(t_l) + h_t/2(Aq(t_l) + f(t_l)v + f(t_{l+1})v))$
end for

Note that we have used MATLAB's invertion algorithm, and not the "\" operator. This is because performing one matrix inversion and k matrix vector products was faster than using the "\" operator k times.

3.3 Measurements and computers

Algorithms 1, 2, 3, together with equation 2.9 was implemented as solvers for equation 1.1, with varieties of algorithm 4 solving the differential equations.

Each solver was implemented in two versions, one where p was assumed separable, and one where p was assumed to be non-separable. Parallel computations was only implemented for KPM with p non-separable, because in this case we needed to solve m independent problems, and not just one.

All solvers and problems were implemented in MATLAB R2014b. The computer used runs Ubuntu 14.04 LTS with intel i7-4770 CPU, and has 16 GB ram.

The parallel computations was implemented with MATLAB's commands `parpool` and `parfor`, see [6] and [5] for more information, `nP` denotes the number of processing units used. Speedup and parallel efficiency will be used to investigate the gain by using several processing units. Speedup is defined as

$$S_{\mathtt{nP}} = \frac{\tau_1}{\tau_{\mathtt{nP}}}$$

and parallel efficiency as

$$\eta_{\mathtt{nP}} = \frac{S_{\mathtt{nP}}}{\mathtt{nP}}$$

where $\tau_{\mathtt{nP}}$ is run time with `nP` processors, $S_{\mathtt{nP}}$ is speedup with `nP` processors, and $\eta_{\mathtt{nP}}$ is parallel efficiency for `nP` processors. Speedup measures how much faster a program runs with `nP` processors, ideal speedup is when $S_{\mathtt{nP}} = \mathtt{nP}$. Parallel efficiency measures how well each processor is used. Perfect parallel efficiency occurs when $\eta_{\mathtt{nP}} = 1$.

Remember that KPM(n) is the method where the n first iterations of Arnoldi's algorithm is used to obtain V_n and H_n, and that KPM is short for KPM(m). Let n be called a restart variable. The number of restarts needed for convergence in algorithm 3 is denoted by γ. The convergence criterion used in algorithm 3 is to stop restarting when the maximum absolute difference in $q_n(t)$ is less than a given tolerance δ. The error is denoted as ϵ and is defined as the largest absolute difference between the correct solution and the approximation.

If nothing else is stated, assume that $\rho = 40$, $k = 40$, $n = 1$, $\delta = 10^{-15}$, these numbers was chosen as large as possible, while still giving an answer in a timely manner. All timed results are averaged over 2 runs, preferably this should have been higher, but would be too time consuming. A was implemented as a sparse matrix.

For separable p, computation time and error and how this scale with δ will be of interest. When p is not separable; convergence, computation time and parallel gain is of interest.

3.4 Test problems

Two test problems are implemented for the separable case. Equation (3.5) is a symmetric problem, and separable for each variable. It is denote as P1.

$$\begin{aligned}
u(t,x,y) &= \frac{t}{t+1}x(x-1)y(y-1) & & \\
f_1(t) &= \frac{1}{(t+1)^2} & g_1(x,y) &= x(x-1)y(y-1) \\
f_2(t) &= \frac{-t}{t+1} & g_2(x,y) &= 2x(x-1)+2y(y-1)
\end{aligned} \tag{3.5}$$

Equation (3.6) is not symmetric, and non-separable for x and y, it also has a combination of polynomials and exponential functions, just to make it test a more general case. This problem will be denoted as P2.

$$
\begin{aligned}
u(t,x,y) =& e^{xy} y(y-1)\sin(\pi x)\, t\cos(t) \\
f_1(t) =& \cos(t) - t\sin(t) \qquad\qquad g_1(x,y) = e^{xy} y(y-1)\sin(\pi x) \\
f_2(t) =& -t\cos(t) \\
g_2(x,y) =& (y-1)y^3 e^{xy}\sin(\pi x) \\
&+ e^{xy}(x^2(y-1)y + x(4y-2)+2)\sin(\pi x) \\
&+ 2\pi(y-1)y^2 e^{xy}\cos(\pi x) - \pi^2(y-1)ye^{xy}\sin(\pi x)
\end{aligned}
\tag{3.6}
$$

To obtain the solutions, solve for $f_i(t)g_i(x,y)$, $i=1,2$ and add the solutions together. Clearly parallel computations could be used to solve these, but this will not be done in this text. The reason for this is that MATLAB uses a significant amount of time to prepare parallel computation, often much more time than the computation itself.

`P1` will also be used in the non-separable case with $p(t) = f_1(t)g_1(x,y) + f_2(t)g_2(x,y)$. One additional test problem is implemented for non-separable p, this is a symmetric problem consisting of both polynomial and exponential functions, it is given in equation (3.7), and will be denoted as `P3`.

$$
\begin{aligned}
u(t,x,y) =& \sin(xyt)(x-1)(y-1) \\
p(t,x,y) =& t^2(x-1)(y-1)y^2\sin(txy) \\
&- 2t(y-1)y\cos(txy) + (x-1)x(y-1)y\cos(txy) \\
&- t(x-1)x(2\cos(txy) - tx(y-1)\sin(txy))
\end{aligned}
\tag{3.7}
$$

Even though both the test problems for non-separable p are symmetric, the method works equally well on non-symmetric problems.

Chapter 4

Computational complexity

This chapter will compare the computational and memory costs for the different methods discussed. This will be done in section 4.1 and 4.2, respectively.

4.1 Computational complexity

Matrix vector multiplication (full)	$\mathcal{O}(m^2)$ [3]
Matrix vector multiplication (sparse)	$\mathcal{O}(m)$ [3]
Matrix inversion	$\mathcal{O}(m^3)$ [3]
Arnoldi's algorithm	$\mathcal{O}(n^2 m)$ [9]
Integration	$\mathcal{O}(k)$

Table 4.1: Computational complexity for some operations. Dimension of the matrices is assumed to be $m \times m$ while n is the restart variable and k is the number of steps in time.

`DM` need to perform k matrix vector multiplications with a sparse matrix, and one matrix inversion.

KPM need to perform k matrix vector multiplications with a full matrix, and one matrix inversion. KPM also need to run Arnoldi's algorithm. If p is non separable, all these operations need to be done m times. If p is separable, one time i enough. `KPM`(n) uses smaller matrices and vectors, with size n. This reduces the cost of these operations, but the method needs to restart.

Remember that the number of restarts `KPM`(n) need to converge is denoted as γ.

An overview over the computational cost of these operations is given in table 4.1. A list of asymptotic computational complexity for the methods is given in table 4.2.

Method	Separable p	Non-separable p
`DM`	$\mathcal{O}(km+m^3)$	$\mathcal{O}(km+m^3)$
`KPM`	$\mathcal{O}(km^2+m^3)$	$\mathcal{O}(km^3+m^4)$
`KPM`(n)	$\mathcal{O}((kn^2+n^2m+n^3)\gamma)$	$\mathcal{O}((kn^2m+n^2m^2+n^3m)\gamma)$

Table 4.2: Computational complexity for the methods discussed, γ denotes the number of restarts needed to converge. Parallel computations will be done for KPM when p is non-separable.

Assume that `KPM`(m) and `KPM` has the same complexity, so that $\gamma = 1$ when $n = m$. KPM has a much higher estimated complexity when p is non-separable, this is the reason to divide between the two cases. The advantage with `DM` is the stable performance, and lower estimated complexity when p is non-separable.

4.2 Memory requirement

A	$\sim 10m$
$q(t)$	$m \times k$
$z(t)$ or $\xi(t)$ or $\zeta(t)$	$n \times k$
$P(t)$	$m \times k$
$f(t)$	k
v	m
V_n	$m \times n$
H_n	$n \times n$
Inverted matrix, with size $m \times m$	$m \times m$

Table 4.3: List over memory demanding variables. All variables are assumed to be discretized with k points in time, and m points in space. Let n be a restart variable.

DM need to store A, $q(t)$,$P(t)$, and an inverted matrix with size $m \times m$.

KPM needs to store A, $q(t)$, $z(t)$, V_n, H_n, and an inverted matrix, remember that for KPM; $n = m$. If p is separable we also need to store v and $f(t)$, if p is non-separable we need to store $P(t)$ instead.

An overview over the memory demand for the different variables is given in table 4.3. A list over memory demand for the different methods is given in table 4.4.

Method	Separable p	Non-separable p
DM	$m^2 + 2mk + 10m$	$m^2 + 2mk + 10m$
KPM	$2mk + 3m^2 + 11m + k$	$3mk + 3m^2 + 11m + k$
KPM(n)	$mk + 2n^2 + k + 11m + nm + nk$	$2(mk + n^2) + k + 11m + nm + nk$

Table 4.4: Memory requirements for the methods discussed. The values are not given asymptotically so that it is easier to distinguish between the different methods.

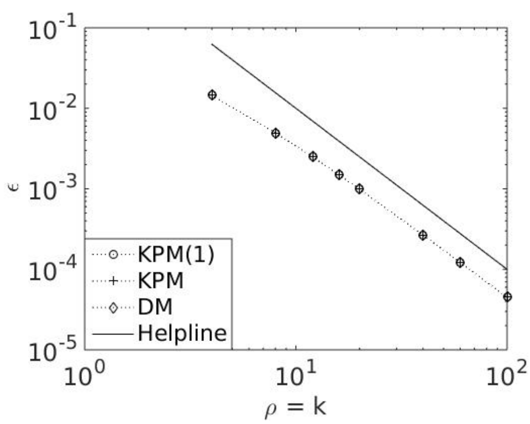
KPM and KPM(m) has the same memory demand, which makes sense. For KPM(1) the memory demand is proportional to mk, which is much lover than for the other methods. The difference in performance between separable and non-separable p is much smaller in this section than in section 4.1.

Chapter 5

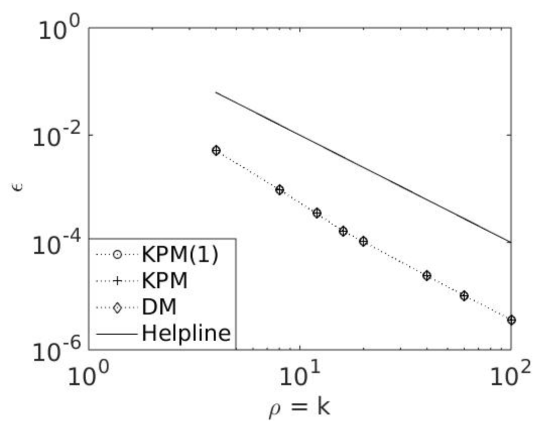
Results for separable p

In this whole section it is assume that p is separable, and that only one processing unit is used to obtain the results. Section 5.1 will show correctness of the methods with a convergence plot. Section 5.3 will see if there is any correlation between n and ρ. Computation times for the different methods will be compared to each other and their predicted computational complexity in section 5.4 and 5.5. How γ and ϵ scales with δ will be examined in section 5.6.

5.1 Convergence



(a) function P1

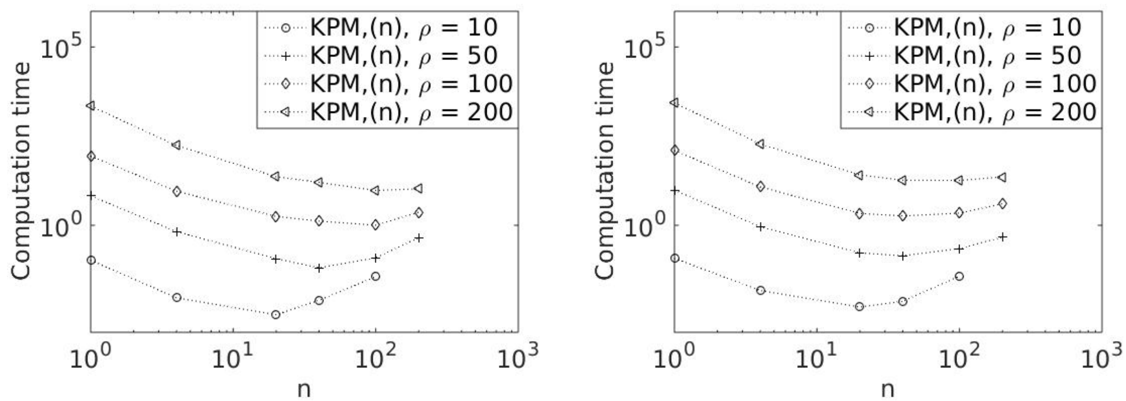


(b) function P2

Figure 5.1: A convergence plot for several methods with $\rho = k$. The helpline shows quadratic convergence.

As can be seen from figure 5.1, all method converges quadratically and overlap perfectly, this shows that all method preforms as expected regarding convergence.

5.2 Choosing restart variable

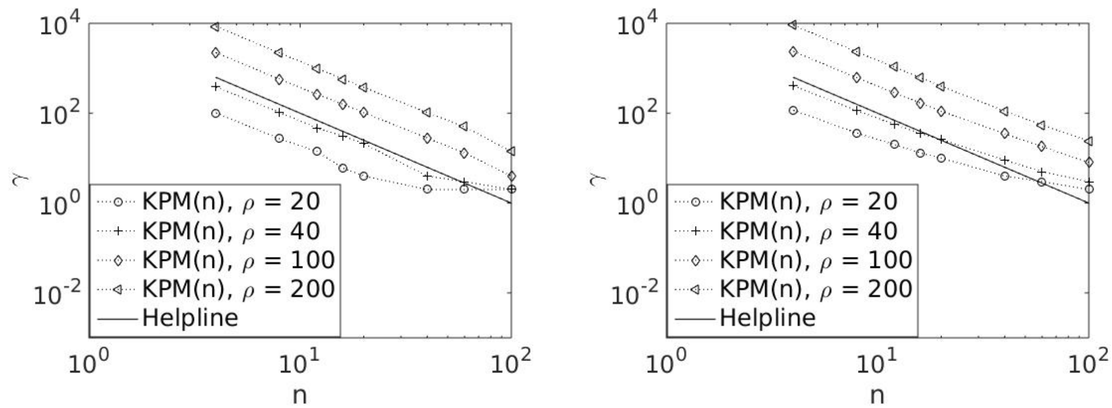


(a) function P1 (b) function P2

Figure 5.2: Computation time plotted against restart variable n.

As can be see from figure 5.2, the optimal restart variable changes as a function of ρ so that larger ρ needs larger n to preform optimally, $n = \rho$ seams to give the smallest computation time for the cases tested. One point is missing from KPM(n), $\rho = 10$, this is because the last point plotted is the same as KPM.

5.3 Comparing γ and n



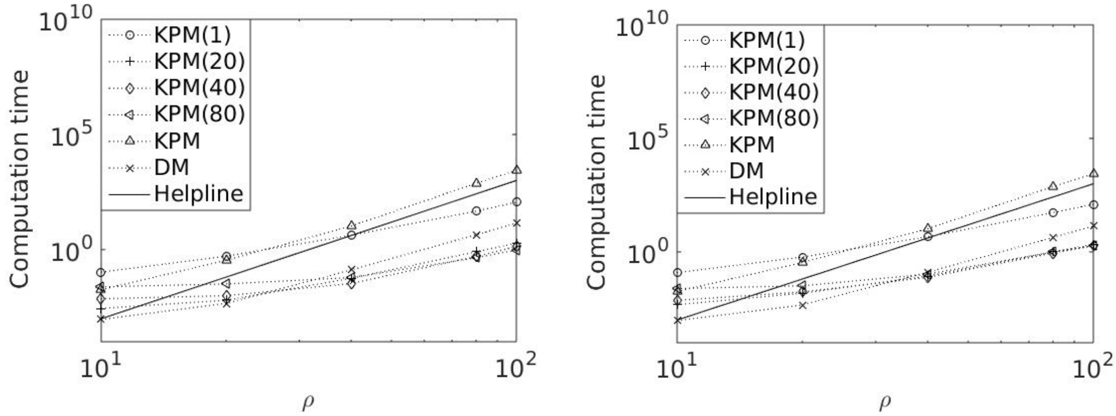
(a) function P1 (b) function P2

Figure 5.3: The number of restarts, γ needed for KPM(n) to converge as a function of the restart variable n. The helpline follows $1/n^2$.

The helpline from figure 5.3 shows that the number of restarts is proportional to $1/n^2$ for all tested cases, if we put this together with the assumption from section 4.1 we get that γ is proportional to m^2/n^2. Using table 4.2 we get that KPM(ρ) has a complexity of $\mathcal{O}(km^2 + m^3)$ for separable p, and $\mathcal{O}(km^3 + m^4)$ for non-separable p, which is the same as for KPM.

We also see that the number of restarts decrease quickest when $n < \rho$, and slower when $n > \rho$. This is the gain we observed in section 5.2. On each side of $n = \rho$ we can perform better, either by performing fewer restarts with larger matrices, or more restarts with smaller matrices.

5.4 Computation time with different ρ

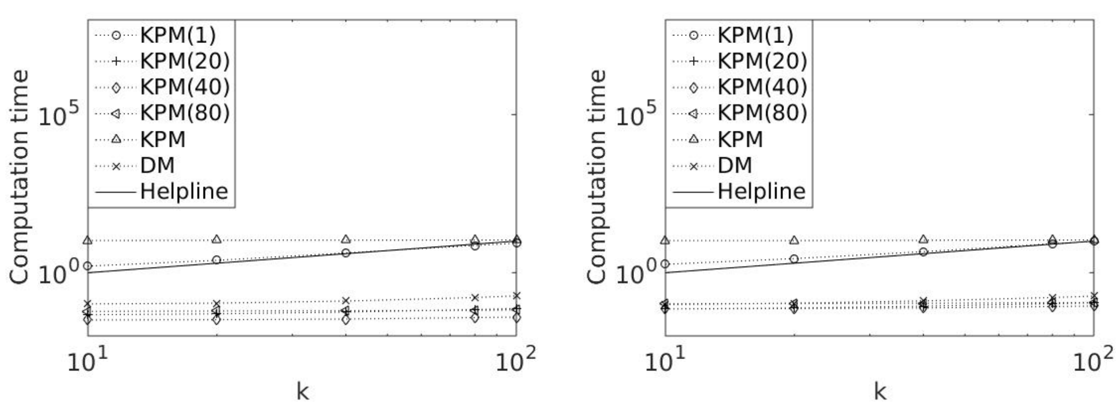


(a) function P1 (b) function P2

Figure 5.4: A plot of computation time as a function of ρ. The helpline increases with $\rho^6 = m^3$.

As we can see from figure 5.4, the computation time for KPM increases as expected, while DM and KPM(n) increases slower, perhaps due to MATLAB's efficient inversion algorithm or less memory demand. Even more interesting is it that KPM(n) is both asymptotically better, and faster than DM for large ρ. Clearly KPM(n) is better in some cases.

5.5 Computation time with different k



(a) function P1 (b) function P2

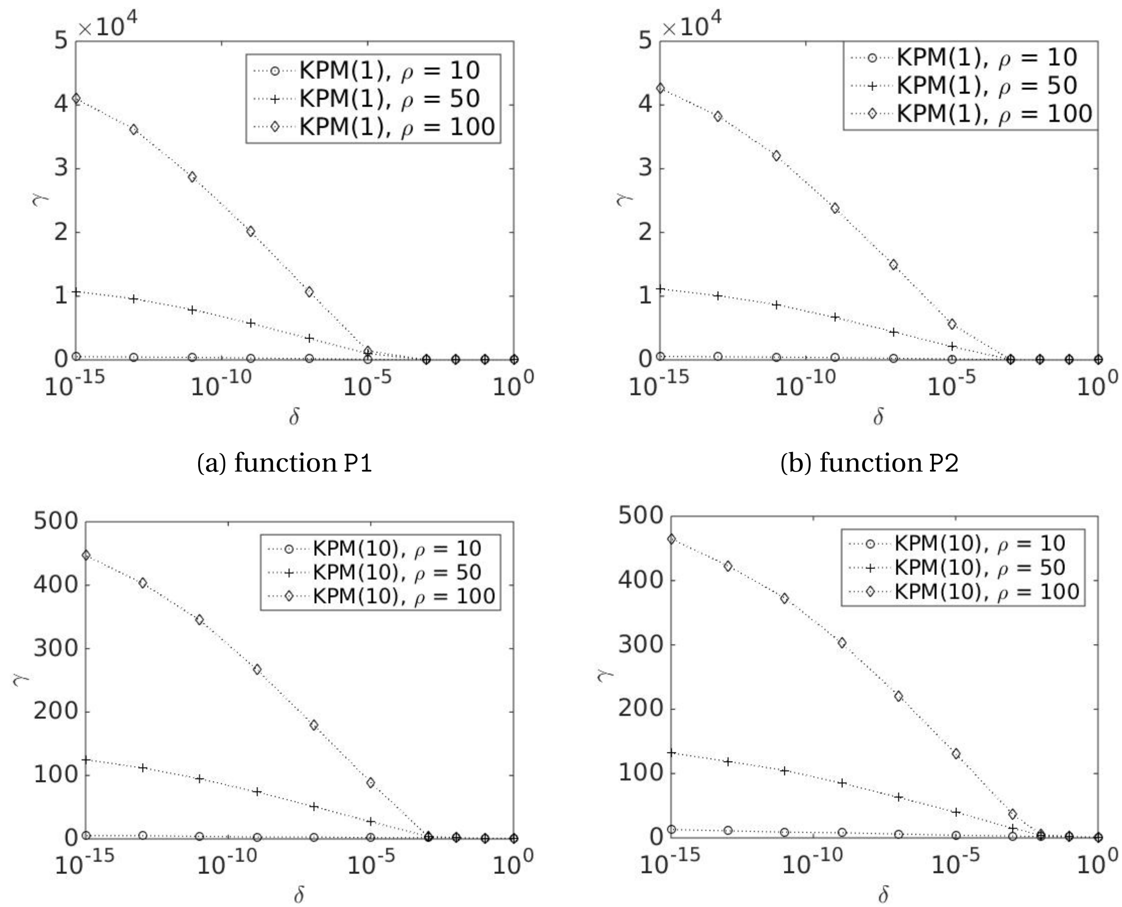
Figure 5.5: A plot of computation time as a function of k. The helpline increases with k.

As we can see from figure 5.5, the computation time for almost all tested methods are constant. The reason is that the time needed for initializing is much greater than the time it takes to integrate. The exception is `KPM(1)`, which increases close to linear because relatively more work is done while integrating, due to several restarts. We also see that `KPM(`n`)` is faster than `DM` for some n, but not asymptotically.

With larger n it is expected that all methods would follow the helpline.

5.6 Comparing δ, γ and ϵ

Remember that δ is tolerance, γ is the number of restarts, and ϵ is a measure for the error.

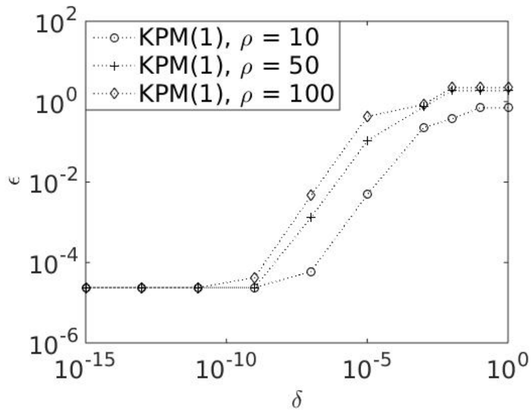


(a) function P1 (b) function P2

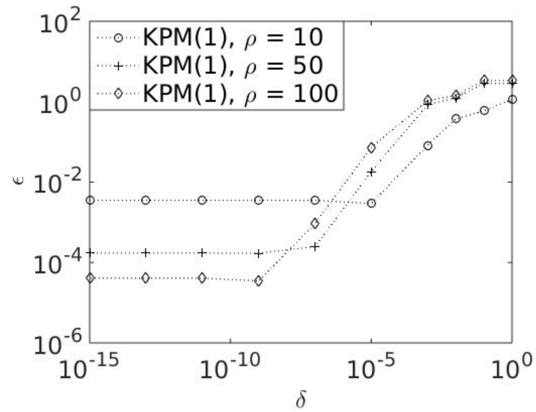
(c) function P1 (d) function P2

Figure 5.6: A plot of γ as a function of δ, with several ρ and n.

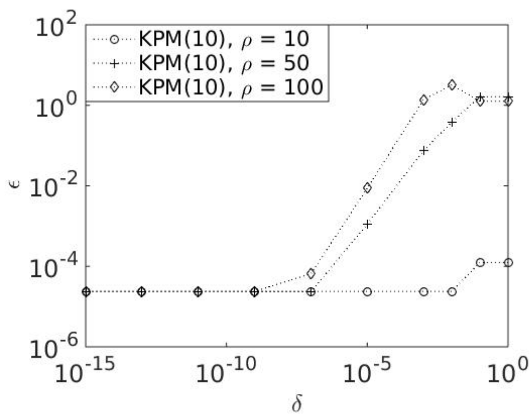
We see from figure 5.6 that γ changes significantly with ρ. The figure show a log linear dependence between γ and δ, and that γ is proportional to $1/n^2$. The constant part of the graph where $10^{-5} < \delta < 10^0$ shows that `KPM(1)` and `KPM(10)` does too few restarts to gain any accuracy when δ is too large, this is confirmed by figure 5.7.



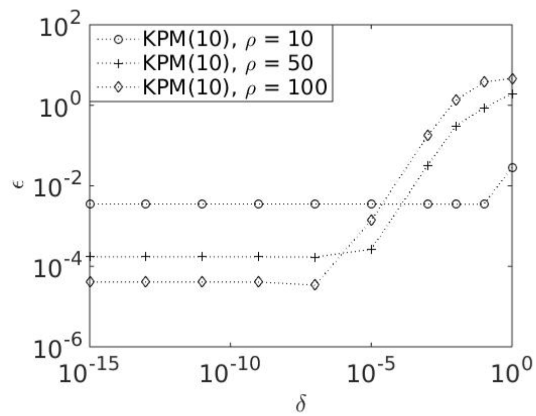
(a) function `P1`



(b) function `P2`



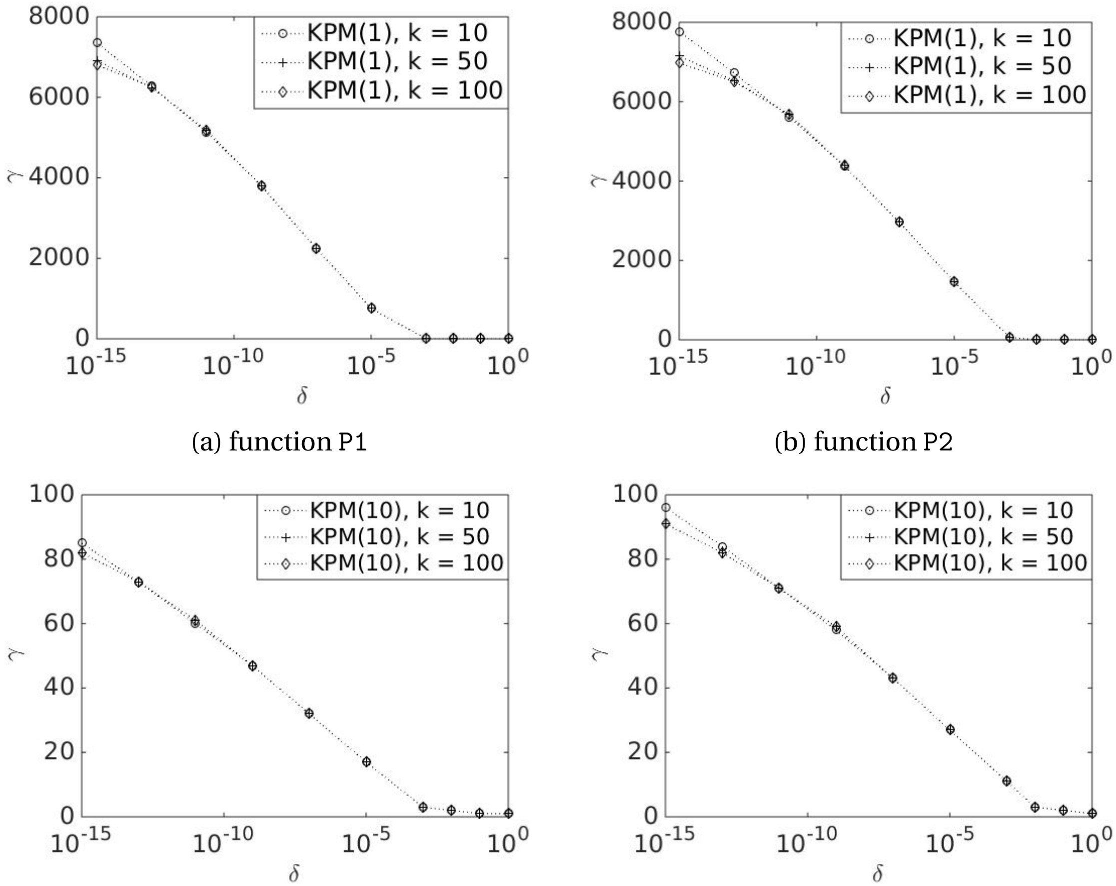
(c) function `P1`



(d) function `P2`

Figure 5.7: A plot of ϵ as a function of δ, with several ρ and n.

Figure 5.7 shows ϵ as a function of δ. In figure 5.7a and 5.7c all graphs has obtained the threshold precision possible with $k = 40$. In figure 5.7b and 5.7d the precision possible with the different ρ-s are obtained. It seams that the maximum precision is attained faster with larger n, this makes sense because if $n = m$, we should not need to restart in order to get threshold precision. This was also confirmed by figure 5.6, where `KPM(10)` needed fewer restarts than `KPM(1)` to converge.



(a) function P1

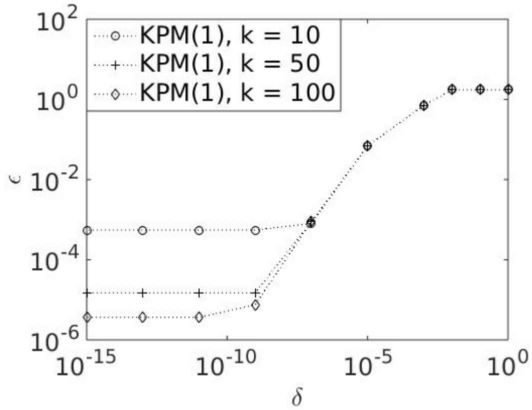
(b) function P2

(c) function P1

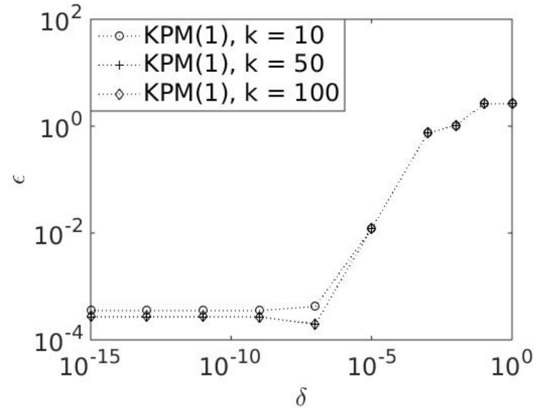
(d) function P2

Figure 5.8: A plot of γ as a function of δ, with several k and n.

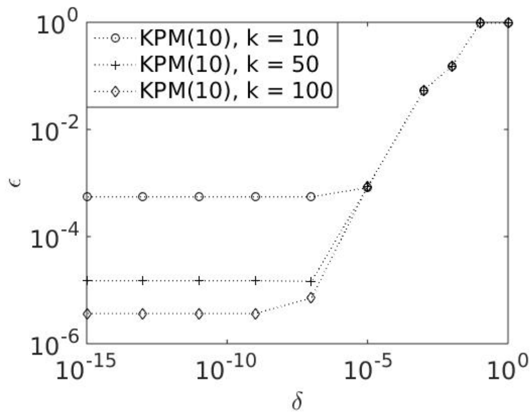
Figure 5.8 shows that γ is nearly independent of k. The figures also shows the log linear dependence between δ and γ.



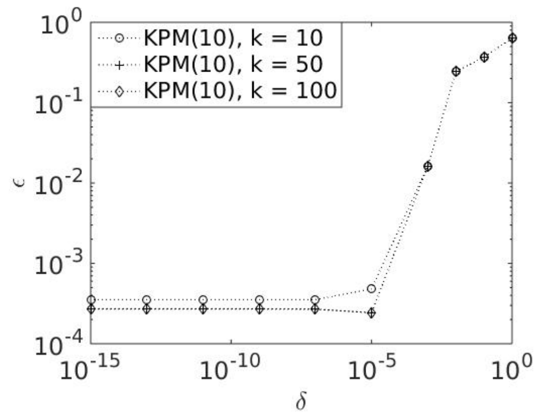
(a) function P1



(b) function P2



(c) function P1



(d) function P2

Figure 5.9: A plot of ϵ as a function of δ, with several k and n.

Figure 5.9 shows much the same as figure 5.7. Maximum precision for different k-s are shown in figure 5.9a and 5.9c. Threshold precision for $\rho = 40$ is obtained in figure 5.9b and 5.9d, but not for the graph where $k = 10$, this is the precision possible with $k = 10$. Again ϵ converges faster for `KPM(10)` than `KPM(1)`.

We see that there is no gain in precision with increasing one of either ρ or k or decreasing δ without changing the others appropriately.

In all cases a lot of time can be saved by choosing δ appropriate. If δ is to large we get inaccurate answers, if δ is to small we perform to many restarts to use the algorithm efficiently. There does not seam to be a simple rule to choose δ, since the results for P1 and P2 differs. The rule we will use is to start at $\delta = 10^{-3}$, with $\rho = k = 10$ and decrease δ with two order of magnitude each time k and ρ is doubled, the reason for this rule is perhaps easier to understand when looking at figure 5.1.

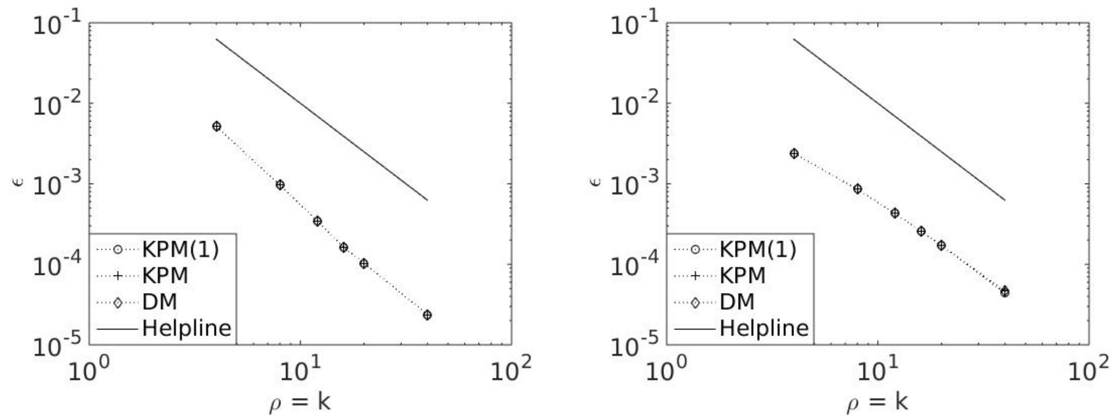
Chapter 6

Results for non-separable p

We now assume that p is non-separable, so that we need to solve m independent problems when using KPM. Several processing units will be used on KPM when possible, remember that it is not possible to use more than one processing unit on `DM`.

We start by showing correctness of the methods with a convergence plot in section 6.1, and proceed with looking into speedup and parallel efficiency in section 6.2. We end by investigating how computation time for the best possible case of KPM compares to `DM` in section 6.3.

6.1 Convergence



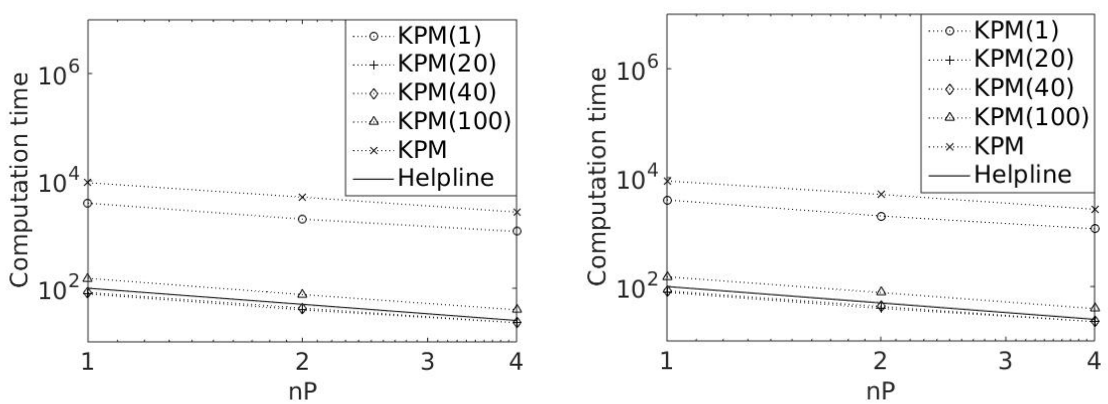
(a) function P1

(b) function P3

Figure 6.1: A convergence plot for several methods with $\rho = k$. The helpline shows quadratic convergence.

As can be seen from figure 6.1, all methods converges quadratically and identically, as in section 5.1. All methods therefore perform as expected regarding convergence.

6.2 Speedup



(a) function P1

(b) function P3

Figure 6.2: Computation time plotted against different number of processors, for several methods. The helpline shows perfect speedup.

☿	P1		P3	
	nP = 2	nP = 4	nP = 2	nP = 4
KPM	1.8556	3.5234	1.7702	3.3193
KPM(1)	1.9858	3.3740	1.9725	3.3924
KPM(20)	1.9883	3.4525	1.9756	3.4547
KPM(40)	1.9619	3.6667	1.9352	3.6642
KPM(100)	2.0083	3.8618	1.9437	3.8362

Table 6.1: Speedup for several cases of KPM.

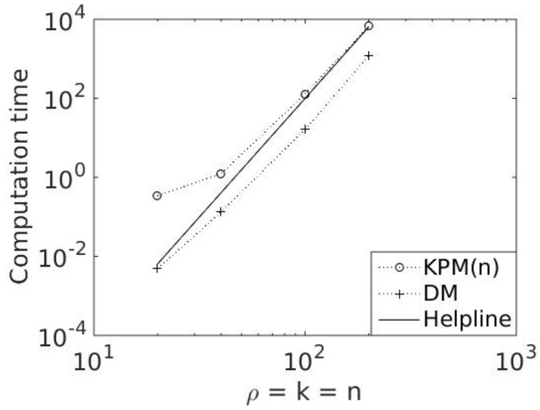
	P1		P3	
	nP = 2	nP = 4	nP = 2	nP = 4
KPM	0.9278	0.8809	0.8851	0.8298
KPM(1)	0.9929	0.8435	0.9862	0.8481
KPM(20)	0.9942	0.8631	0.9878	0.8637
KPM(40)	0.9809	0.9167	0.9676	0.9160
KPM(100)	1.0042	0.9655	0.9719	0.9591

Table 6.2: Parallel efficiency for several cases of KPM.

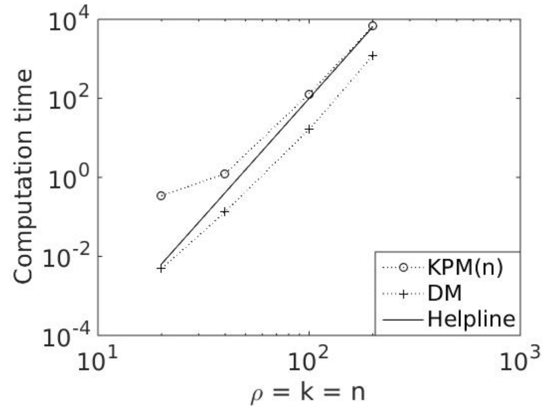
From figure 6.2 together with table 6.1 and 6.2 we can observe the gain by using several processing units. Parallel efficiency and speedup is high for all cases of KPM tested here, it is definitely efficient to use several processing units on this type of problem.

These experiments was only done with $m = k = 40$, this is because the experiments took much time on this computer. I see no reason why parallel gain should change significantly with other m or k.

6.3 Comparison



(a) function P1



(b) function P3

Figure 6.3: A plot of the computation time for KPM(n) and DM. We have used the values $n = \rho = k$. $\delta = 10^{-3}$ for the first point and decreases with two orders of magnitude for each additional point. Assume nP = 4 for KPM(n) and nP = 1 for DM. These values are choosen to make KPM(n) perform as efficiently as possible. The helpline increases with $\rho^6 = m^3$.

From figure 6.3 it is clear that DM is better in all cases simulated, but KPM(n) is not far behind. We know from section 5.4 and 5.5 that one iteration of KPM(n) is faster than DM, we therefore conclude that with enough processing units KPM(ρ) would have been faster than DM.

The other important thing to note here is that computation time for KPM(n) does not increase faster than for DM. It is difficult to say what would happen with larger ρ or k, but they would probably follow the helpline.

The first point of KPM(n) in both figures are very high compared to the trend, perhaps the problem size is to small to be used efficiently with several processing units, or perhaps δ is a little small, making the method restarts too many times.

Chapter 7

Discussion and conclusion

Theoretically all methods perform about the same when p is separable. If p is not separable `DM` has a clear advantage. If we assume that γ is proportional to m^2/n^2, which we concluded in section 5.3, and use $n = \rho$ as suggested in section 5.2, we get that `KPM`(ρ) performs asymptotically equal to `KPM`. This follows from table 4.2.

When p is separable, the results from section 5.4 and 5.5 shows that `KPM`(n) is faster and asymptotically better than `DM`. This contradicts the results from theoretical complexity, and might be due to the smaller memory demand.

The reason for the high parallel performance in section 6.2 is the natural independence in the method. The only communication needed between processors is when adding results, this can be done in $\log_2$(`nP`) additions. Note that a good restart variable also gives a good problem size to use with parallel computations.

In section 6.3 we used what we had learned about γ, n and δ to make `KPM`(n) run as fast as possible with p non-separable. `DM` was faster than `KPM`(ρ), but not asymptotically, as suggested by table 4.2. With more processors `KPM`(n) would have been faster because of the high parallel efficiency and the low cost of solving each of the m independent problems.

Regarding convergence all methods can perform equally well, the drawback is the need to choose an appropriate δ for `KPM`(n). With a larger δ `KPM`(n) is less accurate, but with smaller δ we restart too many times, making the methods inefficient. The rule of thumbs is start at $\delta = 10^{-3}$ with $\rho = k = n = 10$, and decrease δ with two orders of magnitude each time $\rho = k = n$ is doubled.

Table 4.4 shows that `KPM`(n) for $n \leq \rho$ uses the least amount of memory. This is one of the main reasons to use KPM instead of `DM` on this type of problems.

It is worth noting that `DM` did not work when $\rho > 300$ due to memory shortage, while `KPM`(n) had no problem with $\rho = 1000$ and $n = 40$.

Further work

To obtain better results KPM should be implemented in a more parallel friendly language, as for example `C`. It would also be a benefit to use a large computer, making it easier to get data with larger ρ and k.

It would also be interesting to see how KPM could be used to solve other equations than the heat equation.

My code

If you are interested in any of the code used here you can find it at:

https://github.com/sindreka/Prosjektoppgave

Bibliography

[1] CRUZ-URIBE D. and NEUGEBAUER C.J. Sharp error bounds for the trapezoidal rule and simpson's rule. *Journal of Inequalities in Pure and Applied Mathematics,*, Volume 3, Issue 4, Article 49, 2002.

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

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

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

[5] http://se.mathworks.com/help/matlab/ref/parfor.html, September 2015.

[6] http://se.mathworks.com/help/distcomp/parpool.html, September 2015.

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

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

[9] Saad Yousef. *Iterative Methods for Sparse Linear Systems*, volume SECOND EDITION, chapter 6.4.1, pages 160–161. Siam, 2003.