



Department of Mathematical Sciences

A Krylov projection method for the heat equation

Sindre Eskeland

September 7, 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 is used or some assumptions are met. No difference regarding convergence was found.

Contents

Preface	i
Acknowledgment	ii
Summary and Conclusions	iii
1 Introduction	2
2 Krylov subspace and methods	4
2.1 Krylov subspace	4
2.2 Krylov projection method	5
2.3 Restarting the Krylov projection method	6
2.4 When p is not seperable	8
2.5 Direct method	8
3 Implementation	9
3.1 Discretisation in space	9
3.2 Discretisation in time	10
3.3 Measurements and computers	10
3.4 Test problems	12
4 Computational complexity	14
4.1 Computational complexity	14
4.2 Memory requirement	15
5 Results for separable p	17
5.1 Convergence	17

5.2	Choosing restart variable	18
5.3	Comparing γ and n	19
5.4	Computation time with different ρ	20
5.5	Computation time with different k	20
5.6	Comparing δ , γ and ϵ	21
6	Results for non separable p	26
6.1	Convergence	26
6.2	Speedup	27
6.3	Comparison	29

Chapter 1

Introduction

In this report we will investigate how the Krylov projection method (KPM) can be used to solve linear differential equations in 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 (1) for more information about the method.

The main focus of this report is to solve the heat equation with KPM, and compare convergence and computational time with an alternative solution technique, the alternative will be presented in section 2.5. We state the heat equation here with boundary conditions for future references.

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

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

Let us for now assume that the right hand side of equation (1.1) is separable, so that $p(t, x, y) = f(t)g(x, y)$. Given the vector $v = [v_1, v_2, \dots, v_m] \in \mathbb{R}^m$ with elements $g(x_i, y_j)$, $x_i, y_j \in [0, L]$ and

the matrix A as an approximation of the Laplacian, we can obtain a space-discrete version of the heat equation as

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

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 unknown vector on $[0, T]$. Note also that $f(t)$ is a scalar function, so that $f(t)v = [f(t)v_1, f(t)v_2, \dots, f(t)v_m]$. 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 section we will derive some solution techniques. We present the Krylov subspace in section 2.1, derive KPM for the heat equation in section 2.2 and 2.3. In section 2.4 we show how we can use KPM when p is not separable, and introduce the method we will compare KPM to in section 2.5.

2.1 Krylov subspace

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 $n \leq m$. The vectors v^i together with $h_{i,j} = v^{i\top} Av^j$, are found by using Arnoldi's algorithm, shown in algorithm 1. Letting V_n be the $m \times n$ 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 $(h_{i,j})_{i,j=1,\dots,n}$, the following holds

(2)

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

$$V_n^\top AV_n = H_n \quad (2.2)$$

$$v^{i\top} v^j = \delta_{i,j} \quad (2.3)$$

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

Algorithm 1 Arnoldi's algorithm(2)

```

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

```

2.2 Krylov projection method

Let $z(t) = [z_1(t), z_2(t), \dots, z_m(t)] \in \mathbb{R}^m$ 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

$$\begin{aligned} V_m z'(t) - AV_m z(t) &= f(t)v \\ z(0) &= 0 \end{aligned}$$

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

$$\begin{aligned} z'(t) - H_m z(t) &= f(t)V_m^\top v \\ z(0) &= 0 \end{aligned}$$

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 $q(t) = V_m z(t)$ we obtain the solution. A step by step description is given in algorithm 2. We will denote the method KPM.

Let us now 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 $[V_m, H_m] = \text{arnoldi}(A, v)$
 Solve $z'(t) = H_m z + f(t) \|v\|_2 e_1$ for z
 $q_m(t) \leftarrow V_m z(t)$

Since

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

using equation (2.1) and (2.4) we get

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

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

2.3 Restarting the Krylov projection method

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

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

where r_n is as in equation (2.5). If we can solve this equation for $(q - q_n)$, we can improve the approximation of q via iterative refinement.

Equation (2.6) is of the same form as equation (2.4). We derive KPM as we did before, by writing $q(t) = V_m z(t)$ and $q_n = \tilde{V}_n \tilde{\zeta}$. Let \tilde{V}_n be the $m \times m$ matrix with the first n columns equal to the first n columns of V_m , and all additional elements equal to zero. The first n rows of $\tilde{\zeta}$ is equal to ζ , where $q_n = V_n \zeta$. The rest of the elements in $\tilde{\zeta}$ are equal to zero. 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 $\tilde{\xi}(t) = (z - \tilde{\zeta})(t)$, and simplify

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

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

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

Each time we restart we generate a new Krylov subspace $W_n(A, v^{n+1})$, solve equation (2.7) for $\xi(t)$ and approximate the solution q by $q_n = V_n \xi(t)$. By summing together q_n , we converge towards q . 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  $[V_n, H_n, h_{n+1,n}^i, v^{n+1}] = \text{arnoldi}(A, v)$ 
Solve  $z' = H_n z + f(t) \|v\|_2 e_1$  for  $z$ 
 $q_n \leftarrow V_n z$ 
 $\xi_i \leftarrow z$ 
while convergence criterion not satisfied do
     $i \leftarrow i + 1$ 
    Compute  $[V_n, H_n, h_{n+1,n}^i, v^{n+1}] = \text{arnoldi}(A, v^{n+1}, n)$ 
    Solve  $\xi_i'(t) = H_n \xi_i(t) + h_{n+1,n}^{i-1} e_n^\top \xi_{i-1}(t)$  for  $\xi_i$ 
     $q_n(t) \leftarrow q_n + V_n \xi_i(t)$ 
end while

```

2.4 When p is not seperable

If we let $P(t)$ be a vector consisting of elements $p(t, x_i, y_j)$, so that $P(t) = [P_1(t), P_2(t), \dots, P_m(t)]$, and write 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 j th canonical vector in \mathbb{R}^m , we can solve the original equation without requiring separability. An important thing to note here is the need for parallel processing power since we need to solve m problems and not just one.

2.5 Direct method

We need to compare KPM to a well known and easy to implement method. For this we will solve equation (2.9) straight forward with trapezoidal rule.

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

Let $P(t)$ be as in section 2.4. We denote the method DM for direct method.

Chapter 3

Implementation

This section will explain the implementation of the methods. We start by discretizing space and time in section 3.1 and 3.2 respectively. In section 3.3 we present what and how we want to measure interesting factors, together with some information about computers and programs used to generate data. In section 3.4 we state some test problems.

3.1 Discretisation in space

We consider the square $[0, 1] \times [0, 1]$ and divide each spacial direction into $\rho + 2$ piece, with each piece having the length $h_s = 1/(\rho + 1)$. Since the boundary conditions are known, we will only calculate with ρ numbers, leaving room on each side for the boundary. Let $x_i = h_s \cdot i$ and $y_j = h_s \cdot j$, with $i, j = 1, 2, \dots, \rho + 1$. v and $P(t)$ need to be found in an ordered fashion. We let $v_{i+\rho j} = g(x_i, y_j)$, and $P(t)_{i+\rho j} = p(t, x_i, y_j)$, where $i, j = 1, 2, \dots, \rho + 1$. The Laplacian will be approximated by the five point formula given in equation (3.1) as the matrix A . This is a second order approximation.

$$A = \frac{1}{h_s^2} \begin{bmatrix} T & I & & & \\ I & T & I & & \\ & \ddots & \ddots & \ddots & \\ & & I & T & I \\ & & & I & T \end{bmatrix}, T = \begin{bmatrix} -4 & 1 & & & \\ 1 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 1 \\ & & & 1 & -4 \end{bmatrix}, I = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & & & \\ & & & & \\ & & & & 1 \end{bmatrix} \quad (3.1)$$

Notice that $m = \rho^2$.

3.2 Discretisation in time

We will 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 $l = 0, 1, \dots, k-1$.

Algorithm 2, 3 and equation 2.9 all contain a differential equation in time, trapezoidal rule(5), given in equation (3.2) will be used to solve these equations.

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

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

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

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

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

This is a second order method. Because of this discretion, $P(t)$ and $q(t)$ will be saved as $m \times k$ matrices, and $f(t)$ as an $1 \times k$ vector.

3.3 Measurements and computers

Algorithm 1, 2, 3, together with equation 2.9 was implemented as solves for equation 1.1, with varieties of equation 3.4 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 p non separable, because in this case we need solve m independent problems.

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 (4) and (3) for more information, nP denotes the number of processing units used. We will use speedup and parallel efficiency to investigate the gain by using several processing units. Speedup is defined as

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

and parallel efficiency as

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

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

Remember that $KPM(n)$ is the method where we use the n first iterations of Arnoldi's algorithm to obtain V_n and H_n , and KPM is short for $KPM(m)$. We call n 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 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 , we will in general be looking at computation time and error, and how this scale with δ . When p is not separable we are interested in convergence, computation time and parallel gain.

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, we will denote it 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.

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

To obtain the solutions, we need to solve for $f_i(t)g_i(x, y)$, $i = 1, 2$ and add the solutions to-

gether. 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 larger 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.

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

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

We will in section 4.1 and 4.2 briefly compare the computational and memory costs of the different methods discussed.

4.1 Computational complexity

Matrix vector multiplication (full)	$\mathcal{O}(m^2)$ (6)
Matrix vector multiplication (sparse)	$\mathcal{O}(m)$ (6)
Matrix inversion	$\mathcal{O}(m^3)$ (6)
Arnoldi's algorithm	$\mathcal{O}(n^2 m)$ (2)
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 and KPM(n) needs to perform k matrix vector multiplications with a full matrix, and one matrix inversion. KPM and KPM(n) 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 $\text{KPM}(n)$ 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 non separable p .

Assume that $\text{KPM}(m)$ and KPM has the same complexity, so that $\gamma = 1$ when $n = m$. We see that KPM and $\text{KPM}(n)$ has a much higher estimated complexity if p is not separable, this is the reason we divide between the two cases. The advantage with DM is the stable performance.

4.2 Memory requirement

A	$\sim 10m$
q	$m \times k$
P	$m \times k$
f	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.

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

KPM and KPM(n) needs to store A , q , 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 , if p is non separable we need to store P instead.

An overview over the memory demand for the different variables is given in table 4.3. A list of 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	$mk + 3m^2 + 11m + k$	$2mk + 3m^2 + 11m + k$
KPM(n)	$mk + 2n^2 + k + 11m + nm$	$2(mk + n^2) + k + 11m + nm$

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 lower 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 we will assume that p is separable, thus we will only use one processing unit to obtain the results. We will start by showing correctness of the methods with a convergence plot in section 5.1, then see if we can find any correlation between n and ρ in section 5.3. We compare computation times for the different methods to each other and their predicted computational complexity in section 5.4 and 5.5. We will end by seeing how γ and ϵ scales with δ in section 5.6.

5.1 Convergence

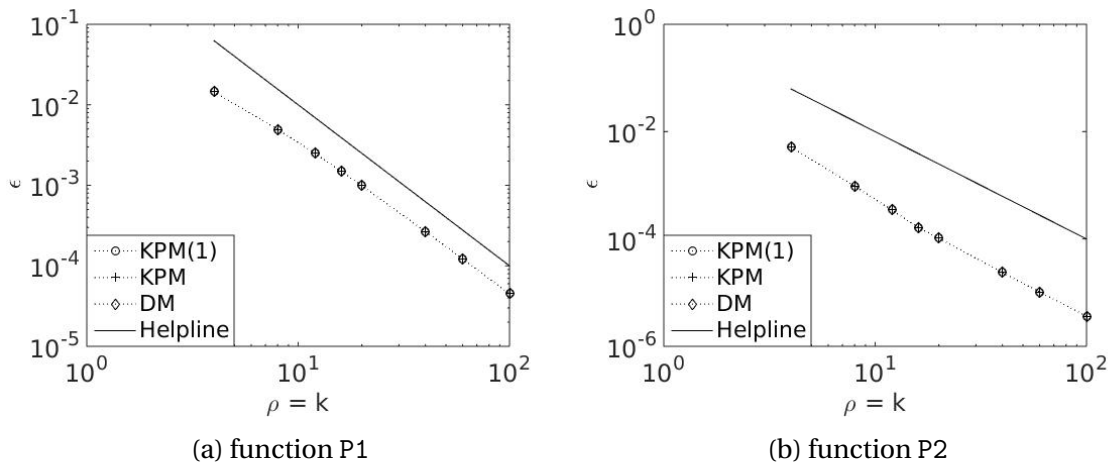


Figure 5.1: A convergence plot for several methods with $\rho = k$. The helpline shows quadratic convergence.

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



Figure 5.2: Computation time plotted against restart variable n .

As we can see from figure 5.2, the optimal restart variable changes as a function of ρ so that larger ρ needs larger n to perform optimally, $n = \rho$ seems 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

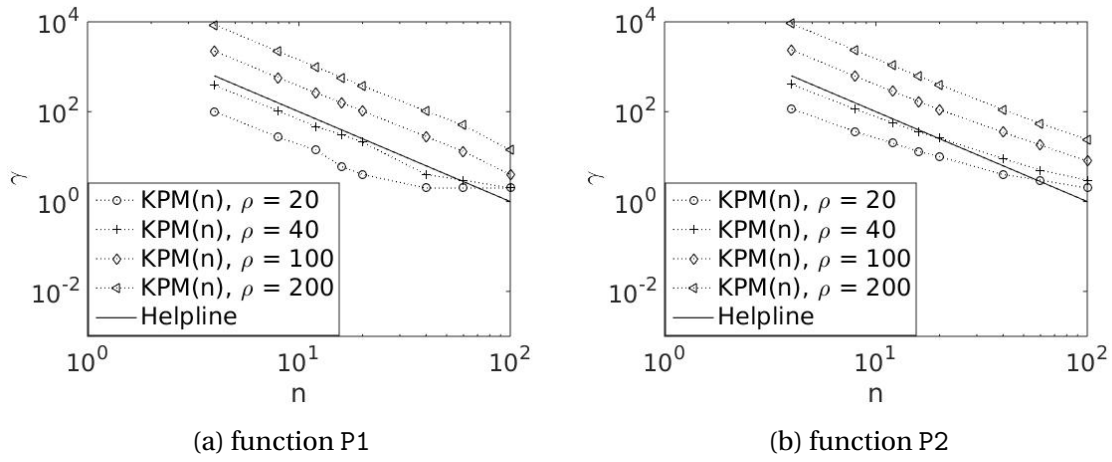


Figure 5.3: The number of restarts, γ needed for $\text{KPM}(n)$ to converge as a function of the restart variable n . The helpline follows $1/n^2$.

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 $\gamma \propto m^2/n^2$. If we put this into table 4.2 we get that $\text{KPM}(\rho)$ 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

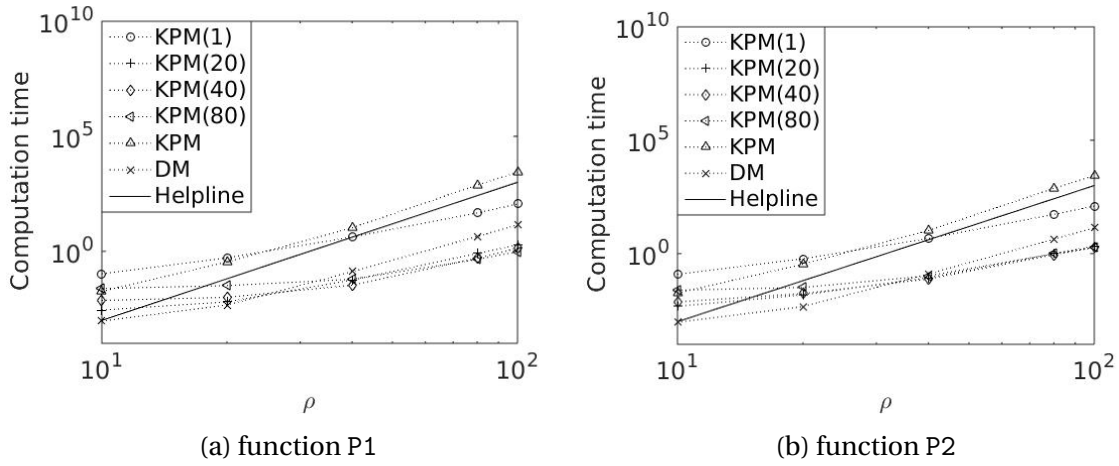


Figure 5.4: A plot of computation time as a function of ρ . The helpline increases with $\rho^6 = m^3$.

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

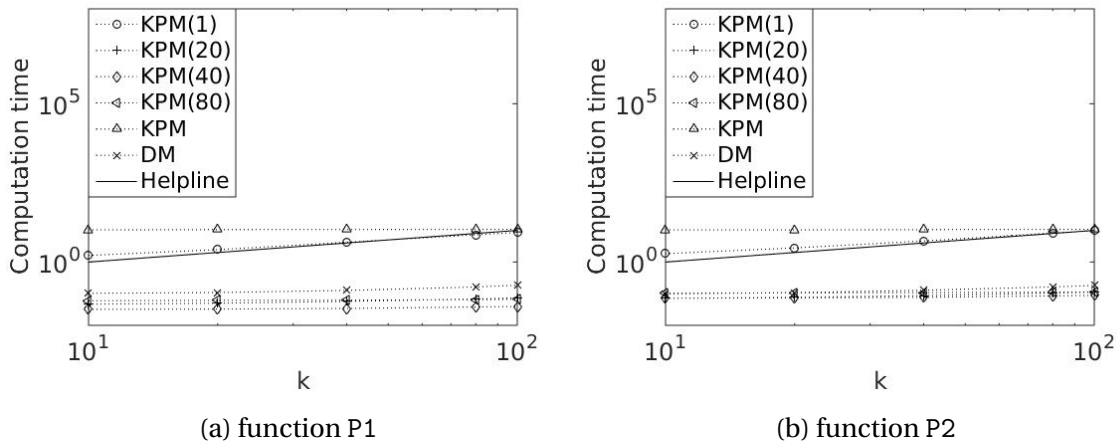


Figure 5.5: A plot of computation time as a function of k . The helpline increases with k .

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 I expect that all method 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.

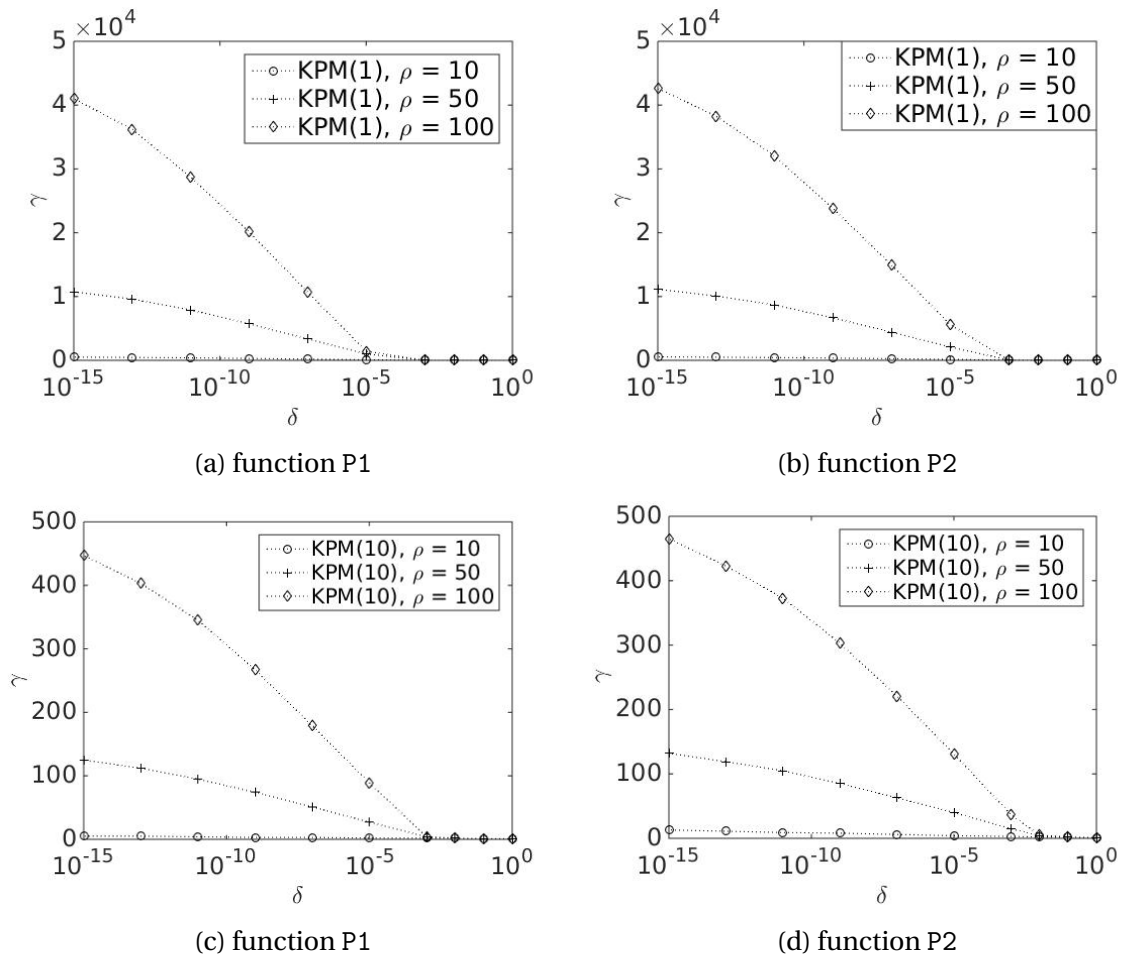


Figure 5.6: A plot of γ as a function of δ , with several ρ and n .

We see from figure 5.6 that γ changes significantly with ρ . The figure show a log linear depen-

dence between γ and δ , and that $\gamma \propto 1/n^2$. 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.

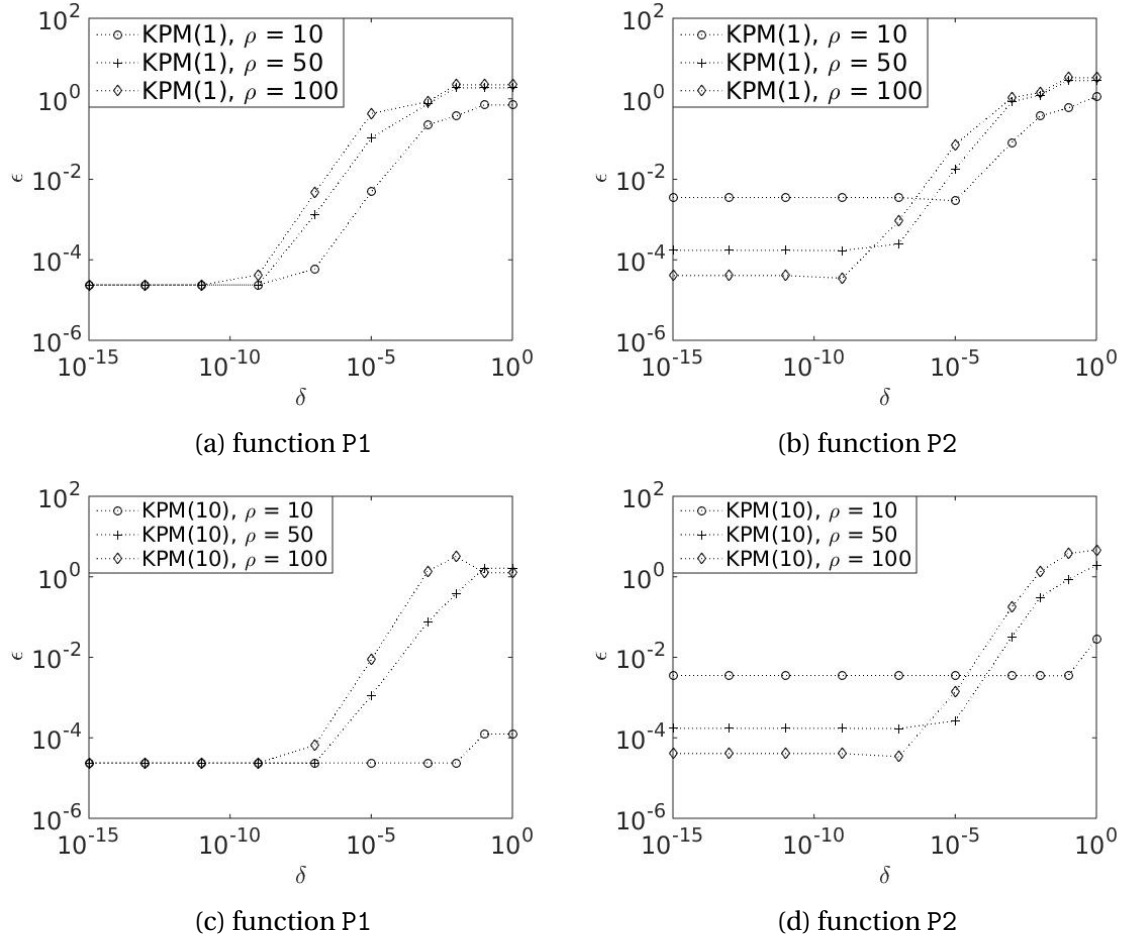


Figure 5.7: A plot of ϵ as a function of δ , with several ρ and n .

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 seems 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.

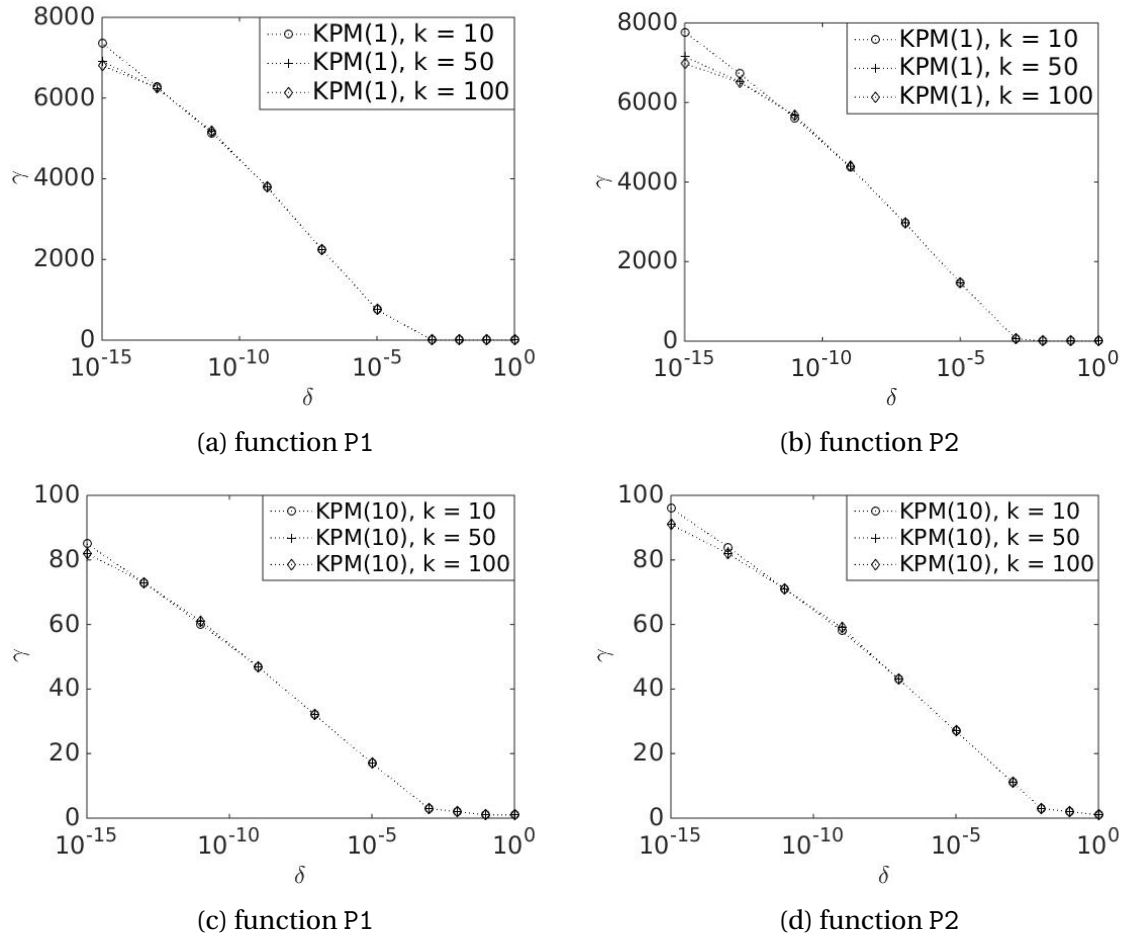


Figure 5.8: A plot of γ as a function of δ , with several k and n .

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

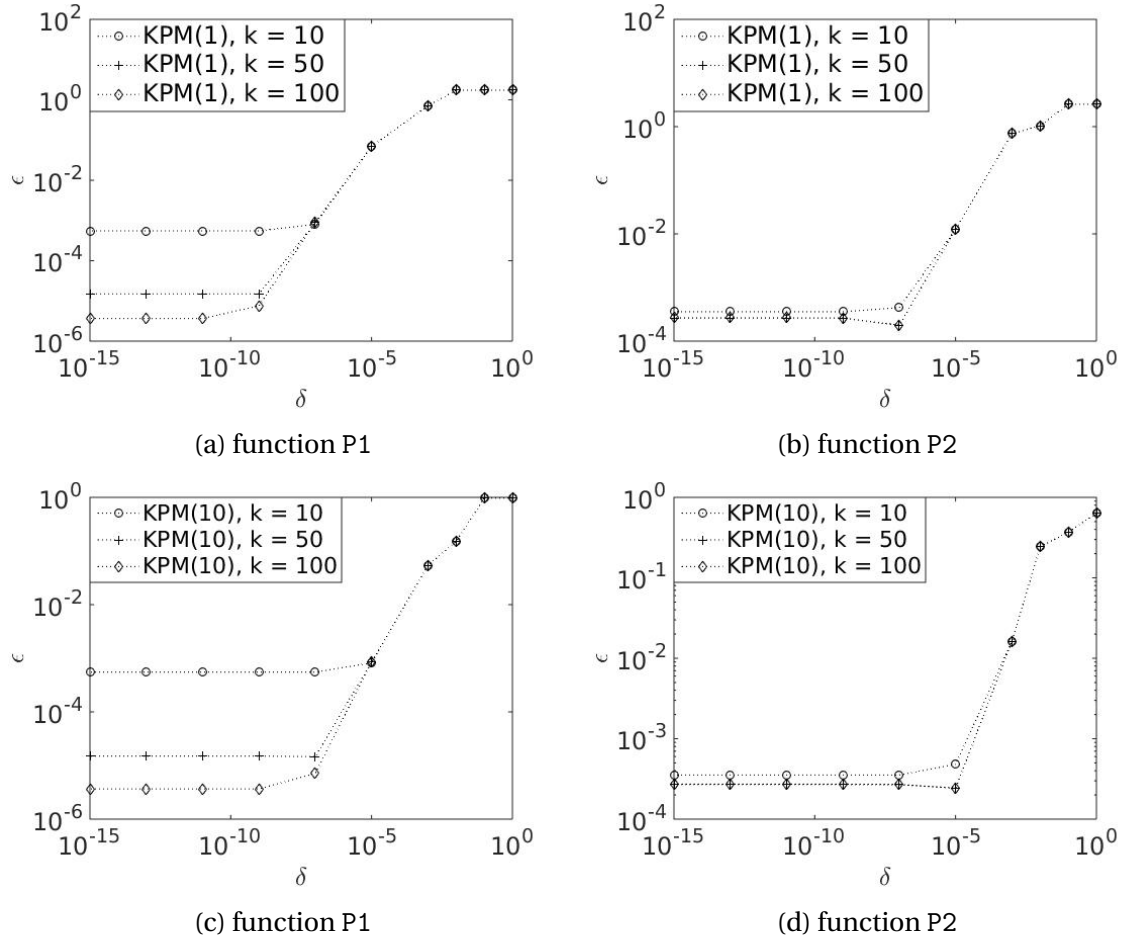
Figure 5.9: A plot of ϵ as a function of δ , with several k and n .

Figure 5.9 shows much the same as figure 5.7. Maximum precision for different k is 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 too large we get inaccurate answers, if δ is too small we perform too many restarts to use the algorithm efficiently. There does not seem to be a simple rule to choose δ , since the results for P1 and P2 differ. The rule I

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 will now try to use what we learned from the previous section in a parallel setting with p non separable and see if we can make KPM outperform DM. We start by showing convergence 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

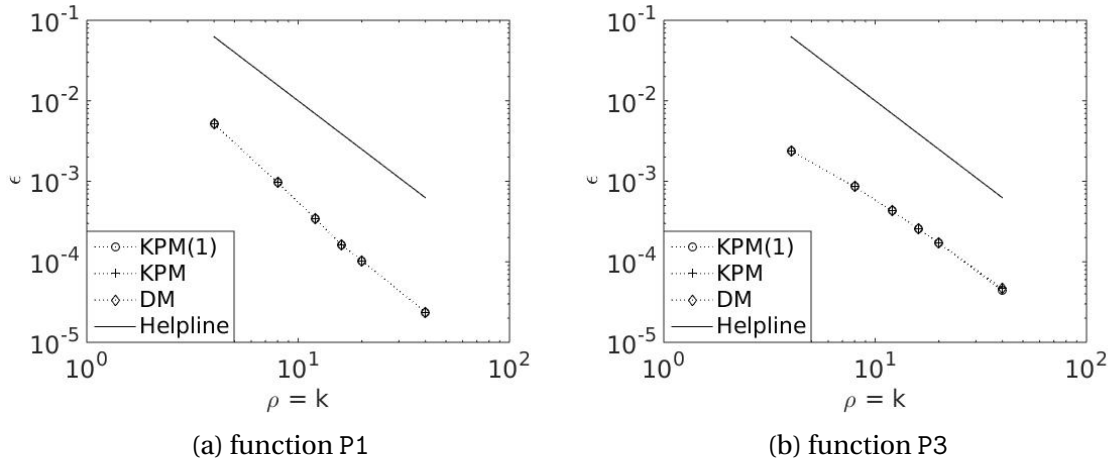


Figure 6.1: A convergence plot for several methods with $\rho = k$. The helpline shows quadratic convergence.

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

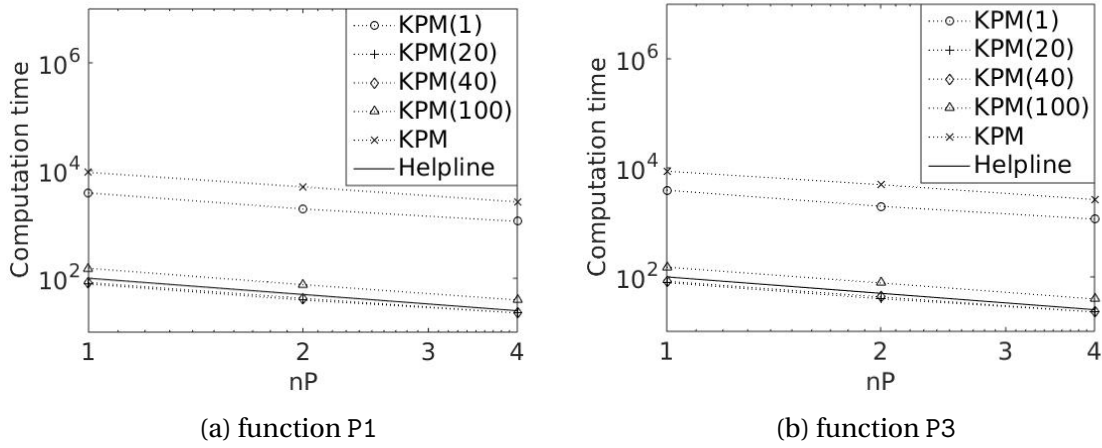


Figure 6.2: Computation time plotted against different number of processors, for several methods. The helpline shows perfect speedup.

Ψ	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

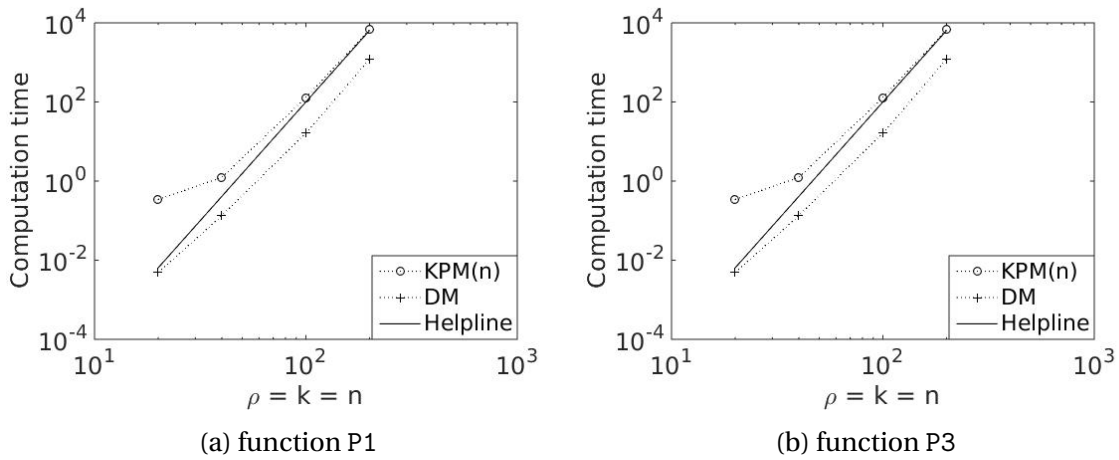


Figure 6.3: A plot of the computation time for $KPM(n)$ and DM. We have used the values $n = \rho = k$. $\delta = 10^{-3}$ for the first point, and decreases with two orders of magnitude for each additional point. Assume $nP = 4$ for $KPM(n)$ and $nP = 1$ for DM. These values are chosen to make $KPM(n)$ 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 $KPM(\rho)$ 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 too small to be used efficiently with several processing units.

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 $\gamma \propto m^2/n^2$, which we concluded in section 5.3, and use $n = \rho$ as suggested in section 5.2, we get that KPM(ρ) has the same complexity as DM and KPM if p is separable. If p is not separable DM has again an advantage over KPM and KPM(n).

Table 4.4 shows that KPM(ρ) uses the least amount of memory. This is one of the main reasons to use KPM instead of DM on this type of 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.

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 is 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 $\text{KPM}(n)$ run as fast as possible with p non separable. DM was faster than $\text{KPM}(\rho)$, but not asymptotically, as suggested by table 4.2. With more processors $\text{KPM}(n)$ would have been faster because of the high parallel efficiency and the low cost of solving each of the m independent problems.

It is worth noting that DM did not work when $\rho > 300$ due to memory shortage, while $\text{KPM}(n)$ 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 . 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] *Krylov projection method for system of ODEs*, E. Celledoni, I. Moret, Applied Numerical Mathematics 23 (1997) 365-378, 1997.
- [2] *Iterative Methods for Sparse Linear Systems*, Yousef Saad, siam, 2003, SECOND EDITION.
- [3] <http://se.mathworks.com/help/distcomp/parpool.html>.
- [4] <http://se.mathworks.com/help/matlab/ref/parfor.html>.
- [5] *SHARP ERROR BOUNDS FOR THE TRAPEZOIDAL RULE AND SIMPSON'S RULE*, D. CRUZ-URIBE AND C.J. NEUGEBAUER, Journal of Inequalities in Pure and Applied Mathematics, 2002, Volume 3, Issue 4, Article 49.
- [6] *Group-theoretic Algorithms for Matrix Multiplication*, Henry Cohn, Robert Kleinberg, Bal azs Szegedy, Christopher Umans, Proceedings of the 46th Annual Symposium on Foundations of Computer Science, 23-25 October 2005, Pittsburgh, PA, IEEE Computer Society, pp. 379-388, 2005.