Multiscale timestepping technique for ODEs and PDEs

MD RAJIBUL ISLAM

Master of Science Thesis
Stockholm, Sweden 2014

Multiscale timestepping technique for ODEs and PDEs

MD RAJIBUL ISLAM

Master's Thesis in Scientific Computing (30 ECTS credits)
Master Programme in Scientific Computing (120 credits)
Royal Institute of Technology year 2014
Supervisor at KTH was Olof Runborg
Examiner was Michael Hanke

TRITA-MAT-E 2014:40
ISRN-KTH/MAT/E--14/40--SE

Royal Institute of Technology
School of Engineering Sciences

KTH SCI
SE-100 44 Stockholm, Sweden

URL: www.kth.se/sci

Abstract

The focus of this thesis is to find efficient ways of solving certain types of *ODEs* and *PDEs*. We have implemented a time upscaling method called Multiscale timestepping technique for this problems. In this method discretization of *PDEs* are transformed into wavelet basis, which divides the solution and the discretized differential operator into coarse scales and fine scales. Larger time steps are then used for solving the fine scale elements. In numerical experiments we show that the accuracy of the solution is maintained but the computational cost is significantly reduced compared to standard methods.

Referat

Tidsstegning med flera skalor för ODE och PDE

Denna uppsats behandlar effektiva metoder för att lösa vissa typer av ODE och PDE. Vi har implementerat en metod för tidsuppskaling som är baserad på tidsstegning med flera skalor. Metoden utgår från en vanlig rumsdiskretisering av en PDE. Den transformeras till en wavelet-bas som delar upp lösningen och den diskretiserade differentialoperatorn i grova och fina skalor. Stora tidssteg används sedan för att approximera de element som motsvarar fina skalor. I numeriska experiment visar vi att noggrannheten i lösningen bibehålls, men att berä kningskostnaden jämfört med standardmetoder blir betydligt mindre.

Contents

Chapter 1

Introduction

This thesis is concerned with efficient ways of solving large ordinary differential equation (ODE) system and time dependent partial differential equations (PDE), in particular parabolic and hyperbolic equations. Hyperbolic equations arise where advective transport and wave propagation are important, for instance in areas like gas dynamics, optics, geophysics, acoustics, elastodynamics and biomechanics. Parabolic equations appear where heat transport or diffusion are involved. By using semi-discretization schemes, time-dependent PDE problems are transformed into ODE systems of the form

$$u_t = Au, \tag{1.1}$$

where A is the space discretization matrix, which is sparse, and u represents the solution vector. Time stepping methods for $ODEs$ are then used to solve the system.

In many areas of scientific and engineering research, scientists are using matrices with millions of unknowns. With increasing computers capability the interest is still rising to include larger and larger problems in their considerations. For simulations of time dependent problems with such large sizes, computational cost is one of the most important issue. Successful execution in limited amount of time for these types of problem is a great challenge. Definitely, decreasing the simulation times in this field will ensure a significant achievement.

By transforming a matrix and a solution vector into wavelet bases the information is divided into different scales [14]. For the matrices that arise from semi-discretization of $PDEs$ most of the information is gathered into a particular area of the matrix. Matrix elements far away from the diagonal and the lower part of the diagonal are in general smaller. Similarly, if the solution is smooth, elements in the solution vector that represent fine scales, will be small. Making simulations that emphasise a particular part or scale, can make the simulations more efficient. Obviously, in that case ensuring the accuracy is also a difficult task. There are many papers which aim is to reduce the computational cost for differential equations by using wavelets. Some wavelet based transformation techniques [1, 5, 6] are applied to matrices obtained by using finite difference and finite element methods to $PDEs$. Kumar & Mehra in [5] implemented Haar and Daubechies based wavelets precondition to achieve large sparse linear system from finite difference and finite elements discretization matrix of non-linear $PDEs$. In [15] dense matrices from discretization of integral equations are transformed into sparse matrices in wavelet bases to reduce costs.

Our goal is to reduce the computational cost by using so-called *time upscaling*. Normally, if $\triangle t$ is the time step and $\triangle x$ is the step size in space, the computational cost for solving an N-dimensional problem is $O(\frac{1}{\triangle t}\frac{1}{\triangle x^N})$. We want to reduce the cost to $O(\frac{|log \triangle t|}{\triangle x^N})$. This is called time upscaling [8]. In recent years different techniques were proposed to reduce the computational cost of solving linear $PDEs$ by time upscaling. For instance, repeated squaring of the solution operator is one technique used in [8, 13, 7].

The time upscaling technique used in this paper we call *Multiscale timestepping*. To solve (1.1), first of all we transform the matrix A in (1.1) and the solution vector u into a wavelet basis. Then we modify the wavelet based matrix (say $L \in \mathbb{R}^{(J+1)\times(J+1)}$) to the matrix H by multiplying the elements of L by positive integers. We also consider different levels $j = 0, 1, 2, \ldots, J$ of these matrices L_j, H_j. The transformation from L to H on level j is shown below.

$$L_j = \begin{pmatrix} l_{0,0} & l_{0,1} & \ldots & l_{0,j} & \ldots 0 \\ l_{1,0} & l_{1,1} & \ldots & l_{1,j} & \ldots 0 \\ \vdots & \vdots & \ddots & \vdots & \\ l_{j,0} & l_{j,1} & \ldots & l_{j,j} & \ldots 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots \\ 0, & 0 & \ldots & 0 & \ldots 0 \end{pmatrix} \Rightarrow H_j = \begin{pmatrix} l_{0,0} & 2^1 l_{0,1} & \ldots & 2^j l_{0,j} & \ldots 0 \\ 2^1 l_{1,0} & 2^1 l_{1,1} & \ldots & 2^j l_{1,j} & \ldots 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 2^j l_{j,0} & 2^j l_{j,1} & \ldots & 2^j l_{j,j} & \ldots 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots \\ 0, & 0 & \ldots & 0 & \ldots 0 \end{pmatrix};$$

After that we implement our Multiscale timestepping technique with some standard ODE methods. The Multiscale timestepping technique with the Forward Euler (FE) method for (1.1) is

$$v^{n+1} = v^n + \triangle t H_{j(n)} v^n. \tag{1.2}$$

Here v is the wavelet transform of u.

During simulations different sizes of matrices and vectors are used in different time steps. The matrices H_j are used in the order $j(n) = 0, 1, 0, 2, 0, 1, 0, 3$ and so on (more details in Section 2.2.3). Thus in every odd iteration we compute only a product of scalars (one element from the matrix and one element from the solution vector) and in even iterations matrices of different sizes are multiplied. In the second iteration a 2×2 matrix and $2-$vector are called. Similarly in the fourth step the simulation is executed for a 3×3 matrix and a $3-$vector, and so on. Finally, only in the last time step we consider the whole matrix H_J and the whole solution v.

In Multiscale timestepping technique, different elements of the solution vector are essentially computed with different time steps $\triangle t = \triangle t(j)$, the size of which depend on the element index j and time level n. More precisely, the k^{th} element of the solution vector in time level n is computed as follows when using H_j.

$$v_k^{n+1} = v_k^n + 2^k \triangle t(l_{k,0}v_0^n + l_{k,1}v_1^n + \ldots\ldots. + l_{k,k}v_k^n) + 2^{k+1}(\triangle t) l_{k,k+1} v_{k+1}^n + \ldots\ldots. + 2^j(\triangle t) l_{k,j} v_j^n,$$

where $j = j(n)$ and $k \leq j$. (This is row k in (1.2).) Hence in the right hand side, factors of the form $2^r \bigtriangleup t$ appear, which can interpreted as different time step sizes.

In particular for element $k = j(n)$ at time level n,

$$v_j^{n+1} = v_j^n + 2^j \bigtriangleup t(l_{j,0}v_0^n + l_{j,1}v_1^n + ... + l_{j,j}v_j^n).$$

Here the time step is $2^j \bigtriangleup t$.

Time upscaling techniques similar to our Multiscale timestepping technique were proposed for the wave equation [9] and for the advection and parabolic equations [11, 10]. M. B. Giles [4] used a geometric sequence of timesteps and reduced computational cost for stochastic differential equations in a similar way. Using almost similar technique like the one in this paper, Stolk [9] reduced the computational cost. After transforming the one-way formulation of wave equation into a wavelet based he integrates the fine scales with longer time steps than the coarse scale. O. Runborg [3] and J. Popovic in [12] reduced the computational cost for tracking interfaces in a time-varying velocity field. They consider longer time steps for finer scales and also apply it to time upscaling of Hamilton Jacobi equations.

Usually Multiscale computation is more complicated than standard computations and for smaller problems the cost may be higher. But if the problem size is large enough, this technique will be faster. Its computational complexity is $O(N(logN)^2)$ for N unknowns and N time steps, instead of $O(N^2)$ for standard methods, see Section 2.4. We show in numerical examples (see Chapter 4) that the accuracy is still maintained.

The organization of this paper is as follows. After this introduction we formulate some structured problems in Chapter 2 that are well suited for our Multiscale timestepping technique and state some standard ODE methods used in this paper. In Chapter 2 we also discuss the implementation procedures of our technique and compare the theoretical numerical cost with standard methods. We introduce some partial differential equations in Chapter 3 and discuss the advantages of using wavelets with partial differential equations in numerical analysis. After that all of the numerical experiments and their results are presented in Chapter 4. Finally in Chapter 5 we have an overall discussion and leave comments for future work.

Chapter 2

Multiscale timestepping for ODEs

2.1 Problem description

Consider the *ODE*

$$\overline{\mathbf{z}}_t = V\overline{\mathbf{z}}, \tag{2.1}$$

where the matrix V and the solution vector $\overline{\mathbf{z}}$ are defined as:

$$V = \begin{pmatrix} v_{0,0} & v_{0,1} & \cdots & v_{0,J} \\ v_{1,0} & v_{1,1} & \cdots & v_{1,J} \\ \vdots & \vdots & \ddots & \vdots \\ v_{J,0} & v_{J,1} & \cdots & v_{J,J} \end{pmatrix} \in \mathbb{R}^{(J+1)\times(J+1)}, \qquad \overline{\mathbf{z}}(t) = \begin{pmatrix} z_0(t) \\ z_1(t) \\ \vdots \\ z_J(t) \end{pmatrix} \in \mathbb{R}^{(J+1)}.$$

We want to solve (2.1) when the matrix elements and the initial vector elements satisfy

$$|v_{j,k}| \leq C2^{-q|j-k|}, \qquad |z_j(0)| \leq C2^{-jQ}, \tag{2.2}$$

where C is some constant and q, Q are some positive numbers used to describe the decay rate of the elements of the matrix and the initial vector. We call them the *matrix generator* number and *vector generator* number respectively. Larger q and Q indicate smaller elements of the matrix and the vector and vice-versa. For this structure, the matrix elements become smaller away from the diagonal. The initial vector decreases its value from top to bottom. We will also discuss a blocked version of the problem (2.1) later in Section 2.3.

The reason for beginning with this types of structured problem is the application of wavelets in solving *PDEs*. Similar decay properties like our structured matrix and the solution vector in (2.2) are observed when the discretization matrix of the *PDEs* and the solution vector are transformed into a wavelet basis. Hence, (2.1) and in particular the blocked version in Section 2.3 is considered as a model problem for such *PDE* discretizations.

2.2 Numerical method

2.2.1 Standard methods

We consider first some standard ODE methods for solving equation (2.1). Let $\mathbf{z}^n$ be the numerical approximation of $\mathbf{z}(n \triangle t)$ and $T = 2^J \triangle t$, where $\triangle t$ is the time step and J is used to define the problem size.

The **Forward Euler method** for problem (2.1) can be written as

$$\mathbf{z}^{n+1} = \mathbf{z}^n + \triangle t V \mathbf{z}^n.$$

According to the Forward Euler method in each time step it multiplies $(I + \triangle t V)$ with $\mathbf{z}^n$. Since V is a full matrix the cost is $O(J^2)$. So if the problem (2.1) is solved by Forward Euler method with N time steps, the total cost is $O(NJ^2)$. This time stepping method shows **first order of accuracy** in error reduction.

On the other hand the classical **Runge-Kutta** $4th$ **(RK4) order method** for equation (2.1) can be written as:

$$\mathbf{z}^{n+1} = \mathbf{z}^n + \frac{\triangle t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.3}$$

where

$$\begin{aligned} k_1 &= \triangle t V \mathbf{z}^n \\ k_2 &= \triangle t V (\mathbf{z}^n + \frac{k_1}{2}) \\ k_3 &= \triangle t V (\mathbf{z}^n + \frac{k_2}{2}) \\ k_4 &= \triangle t V (\mathbf{z}^n + k_3). \end{aligned}$$

If we look at k_1, k_2, k_3 and k_4 above, we can see that a matrix-vector multiplication is needed to calculate k_1. Moreover for each k_2, k_3 and k_4, a vector-vector addition and then matrix-vector multiplication is required. So for each one of k_1, k_2, k_3 and k_4 the cost is $O(J^2)$. Hence the total cost of this method in each time step is $O(J^2)$ and as above the cost to do N time steps is $O(NJ^2)$.

2.2.2 Introductory Multiscale timestepping example

We know that the computational cost for a matrix-vector multiply is proportional to the number of non-zero elements of the matrix and the vector. During the solution procedure with standard methods, in every time step the whole matrix $I + \triangle t V$ is applied to the whole vector $\mathbf{z}^n$. So in each time step the cost is calculated from all of the non-zero elements of the matrix and the vector. In Multiscale timestepping techniques, except for the very last time step, instead of operating the whole matrix and the whole vector, only a part of the matrix operates on a part of the vector in each time step. The most interesting point is that in every odd iteration this technique operates only a single element from the matrix and the vector. As a result the computational cost is significantly reduced.

To get an overview of the Multiscale timestepping technique, we start by considering a simple toy problem as an example. Suppose we want to solve

$$\overline{\mathbf{z}}_t = V\overline{\mathbf{z}}, \tag{2.4}$$

where V is a 4×4 matrix and $\overline{\mathbf{z}}$ is a $4-$vector i.e. $J = 3$.

$$V = \begin{pmatrix} v_{0,0} & v_{0,1} & v_{0,2} & v_{0,3} \\ v_{1,0} & v_{1,1} & v_{1,2} & v_{1,3} \\ v_{2,0} & v_{2,1} & v_{2,2} & v_{2,3} \\ v_{3,0} & v_{3,1} & v_{3,2} & v_{3,3} \end{pmatrix}, \qquad \overline{\mathbf{z}}^0 = \begin{pmatrix} z_0^0 \\ z_1^0 \\ z_2^0 \\ z_3^0 \end{pmatrix}.$$

Now step by step we solve equation (2.4) by Multiscale timestepping technique using the Forward Euler time stepper as follows:

- **Step 1**

$$z_0^1 = z_0^0 + \triangle t v_{0,0} z_0^0$$

After step 1 the updated solution becomes

$$\overline{\mathbf{z}}^1 = \begin{pmatrix} z_0^1 \\ z_1^0 \\ z_2^0 \\ z_3^0 \end{pmatrix}.$$

- **Step 2**

$$z_0^2 = z_0^1 + \triangle t v_{0,0} z_0^1 + 2 \triangle t v_{0,1} z_1^0$$
$$z_1^2 = z_1^0 + 2 \triangle t v_{1,0} z_0^1 + 2 \triangle t v_{1,1} z_1^0$$

So now the updated solution is

$$\overline{\mathbf{z}}^2 = \begin{pmatrix} z_0^2 \\ z_1^2 \\ z_2^0 \\ z_3^0 \end{pmatrix}.$$

- **Step 3**

$$z_0^3 = z_0^2 + \triangle t v_{0,0} z_0^2$$

Now the solution becomes

$$\overline{\mathbf{z}}^3 = \begin{pmatrix} z_0^3 \\ z_1^2 \\ z_2^0 \\ z_3^0 \end{pmatrix}.$$

- **Step 4**

$$z_0^4 = z_0^3 + \triangle t v_{0,0} z_0^3 + 2 \triangle t v_{0,1} z_1^2 + 4 \triangle t v_{0,2} z_2^0$$
$$z_1^4 = z_1^2 + 2 \triangle t v_{1,0} z_0^3 + 2 \triangle t v_{1,1} z_1^2 + 4 \triangle t v_{1,2} z_2^0$$
$$z_2^4 = z_2^0 + 4 \triangle t v_{2,0} z_0^3 + 4 \triangle t v_{2,1} z_1^2 + 4 \triangle t v_{2,2} z_2^0$$

After this step the solution vector is

$$\overline{\mathbf{z}}^4 = \begin{pmatrix} z_0^4 \\ z_1^4 \\ z_2^4 \\ z_3^0 \end{pmatrix}.$$

- **Step 5**

$$z_0^5 = z_0^4 + \triangle t v_{0,0} z_0^4$$

So now the updated solution is

$$\overline{\mathbf{z}}^5 = \begin{pmatrix} z_0^5 \\ z_1^4 \\ z_2^4 \\ z_3^0 \end{pmatrix}.$$

- **Step 6**

$$z_0^6 = z_0^5 + \triangle t v_{0,0} z_0^5 + 2 \triangle t v_{0,1} z_1^4$$
$$z_1^6 = z_1^4 + 2 \triangle t v_{1,0} z_0^5 + 2 \triangle t v_{1,1} z_1^4$$

Now the updated solution becomes

$$\overline{\mathbf{z}}^6 = \begin{pmatrix} z_0^6 \\ z_1^6 \\ z_2^4 \\ z_3^0 \end{pmatrix}.$$

- **Step 7**

$$z_0^7 = z_0^6 + \triangle t v_{0,0} z_0^6$$

After step 7 the solution vector becomes

$$\overline{\mathbf{z}}^7 = \begin{pmatrix} z_0^7 \\ z_1^6 \\ z_2^4 \\ z_3^0 \end{pmatrix}.$$

- **Step 8**

$$\begin{aligned}
z_0^8 &= z_0^7 + \triangle t v_{0,0} z_0^7 + 2 \triangle t v_{0,1} z_1^6 + 4 \triangle t v_{0,2} z_2^4 + 8 \triangle t v_{0,3} z_3^0 \\
z_1^8 &= z_1^6 + 2 \triangle t v_{1,0} z_0^7 + 2 \triangle t v_{1,1} z_1^6 + 4 \triangle t v_{1,2} z_2^4 + 8 \triangle t v_{1,3} z_3^0 \\
z_2^8 &= z_2^4 + 4 \triangle t v_{2,0} z_0^7 + 4 \triangle t v_{2,1} z_1^6 + 4 \triangle t v_{2,2} z_2^4 + 8 \triangle t v_{2,3} z_3^0 \\
z_3^8 &= z_3^0 + 8 \triangle t v_{3,0} z_0^7 + 8 \triangle t v_{3,1} z_1^6 + 8 \triangle t v_{3,2} z_2^4 + 8 \triangle t v_{3,3} z_3^0
\end{aligned}$$

So our solution vector is

$$\bar{\mathbf{z}}^8 = \begin{pmatrix} z_0^8 \\ z_1^8 \\ z_2^8 \\ z_3^8 \end{pmatrix}.$$

The above example shows that in each of the odd iteration only a product of two scalars is calculated. Moreover, just in the last time step the whole matrix and the whole vector is used. During all other time steps smaller sub-matrices are used.

2.2.3 Method

If we look into the solution procedure of the previous example we can see that to solve the equation (2.4) instead of using the matrix

$$V = \begin{pmatrix} v_{0,0} & v_{0,1} & v_{0,2} & v_{0,3} \\ v_{1,0} & v_{1,1} & v_{1,2} & v_{1,3} \\ v_{2,0} & v_{2,1} & v_{2,2} & v_{2,3} \\ v_{3,0} & v_{3,1} & v_{3,2} & v_{3,3} \end{pmatrix},$$

we used a modified version of V defined as:

$$H = \begin{pmatrix} 2^0 v_{0,0} & 2^1 v_{0,1} & 2^2 v_{0,2} & 2^3 v_{0,3} \\ 2^1 v_{1,0} & 2^1 v_{1,1} & 2^2 v_{1,2} & 2^3 v_{1,3} \\ 2^2 v_{2,0} & 2^2 v_{2,1} & 2^2 v_{2,2} & 2^3 v_{2,3} \\ 2^3 v_{3,0} & 2^3 v_{3,1} & 2^3 v_{3,2} & 2^3 v_{3,3} \end{pmatrix}.$$

The example also shows that during the time steps, instead of applying the whole matrix H, we used smaller size sub-matrices of H and different sizes of the solution vector from $\bar{\mathbf{z}}$. Only in the last time step did we use the whole matrix H and the whole vector $\bar{\mathbf{z}}$. The sizes for different sub-matrices and initial vectors in different time steps change as $1, 2, 1, 3, 1, 2, 1, 4$. This ensures that the j^{th} component of the solution vector is updated every $2^{j-1}th$ step. It is thus effectively approximated using a time step of size $2^{j-1} \triangle t$.

We will now describe the general procedure. We create new matrices H_j from the matrix V, where

$$H_j = \begin{pmatrix} v_{0,0} & 2^1 v_{0,1} & \dots & 2^j v_{0,j} & \dots 0 \\ 2^1 v_{1,0} & 2^1 v_{1,1} & \dots & 2^j v_{1,j} & \dots 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 2^j v_{j,0} & 2^j v_{j,1} & \dots & 2^j v_{j,j} & \dots 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots \\ 0, & 0 & \dots & 0 & \dots 0 \end{pmatrix}; \qquad j = 0, 1, 2, ..., J. \tag{2.5}$$

We also define the ordering $j(n)$ as

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...	J-1	J
$j(n)$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	...	0	J

The general formula of j is

$$j(n) = \max\{k \in \mathbb{Z}^+ : n+1 \text{ is divisible by } 2^k\}. \tag{2.6}$$

In every time step we use a different size of sub-matrices as well as solution vectors and the solution vector is updated according to that order. The size of sub-matrices and solution vectors for different time step is determined according to the value of $j(n)$ (defined above). So simply we can write the Multiscale technique for equation (2.1) with Forward Euler method as

$$\mathbf{z}^{n+1} = \mathbf{z}^n + \triangle t H_{j(n)} \mathbf{z}^n \tag{2.7}$$

Similarly Multiscale timestepping technique for Backward Euler method (BE), Crank-Nicolson (CN) method, Runge-Kutta $2nd$ (RK2) order method and Runge-Kutta $4th$ order method for equation (2.1) can be written as:

- **Backward Euler method**

$$\mathbf{z}^{n+1} = \mathbf{z}^n + \triangle t H_{j(n)} \mathbf{z}^{n+1} \tag{2.8}$$

- **Crank-Nicolson method**

$$\implies (1 - H_{j(n)} \frac{\triangle t}{2}) \mathbf{z}^{n+1} = (1 + H_{j(n)} \frac{\triangle t}{2}) \mathbf{z}^n \tag{2.9}$$

- **Runge-Kutta 2^{nd} order method**

$$\mathbf{z}^{n+1} = \mathbf{z}^n + k_2, \tag{2.10}$$

where

$$\begin{aligned} k_1 &= \triangle\, t H_{j(n)} \mathbf{z}^n \\ k_2 &= \triangle\, t H_{j(n)} (\mathbf{z}^n + \frac{k_1}{2}) \end{aligned}$$

- **Runge-Kutta 4^{th} order method**

$$\mathbf{z}^{n+1} = \mathbf{z}^n + \frac{\triangle t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.11}$$

where

$$\begin{aligned} k_1 &= \triangle\, tH_{j(n)}\mathbf{z}^n \\ k_2 &= \triangle\, tH_{j(n)}(\mathbf{z}^n + \frac{k_1}{2}) \\ k_3 &= \triangle\, tH_{j(n)}(\mathbf{z}^n + \frac{k_2}{2}) \\ k_4 &= \triangle\, tH_{j(n)}(\mathbf{z}^n + k_3) \end{aligned}$$

2.2.4 Motivation

The Multiscale timestepping essentially use a larger time step for the components in $\overline{z}$ with larger j. We can think of it as using time step $\triangle t_j = 2^j \triangle t$ for $z_j(t)$, where $\triangle t = 2^{-(J+1)}$.

We know that if we solve an ODE with a time stepping method, the error depends on the time step $\triangle t$ and also the higher derivatives of the solution. Large derivative means a bigger error for a fixed $\triangle t$. Therefore if higher order derivatives of the solution are small we can solve the problem by taking larger time steps. We therefore need to study the sizes of the derivatives of $z_j(t)$, and how they depend on j. We have already talked about the structure of the matrix V and the initial condition $\overline{z}(0)$ of our considered problem in Section 2.1. Now we look theoretically at the size of its derivatives at time zero.

Lemma 2.2.1. *Suppose $\overline{z}(t) \in \mathbb{R}$ solves (2.1) with initial data and the matrix V satisfying (2.2) with min(q,Q) > 0. Then*

$$|z_j'(0)| \le C_0 2^{-j\min(q,Q)}(j+1)$$

and

$$|z_j''(0)| \le C_1 2^{-j\min(q,Q)}(j+1)^2$$

Proof: We have

$$\begin{aligned}
|z_j'(0)| =& |\sum_{i=1}^{J} V_{ji} z_i(0)| \leq \sum_{i=0}^{J} |V_{ji}||z_i(0)| \leq C\sum_{i=0}^{J} |2^{-|i-j|q}| \cdot |2^{-iQ}| \leq C\sum_{i=0}^{J} 2^{-(|i-j|+i)\ min(q,Q)} \\
=& C\sum_{i=0}^{j} 2^{-j\min(q,Q)} + C\sum_{i=j+1}^{J} 2^{-(2i-j)\ min(q,Q)} = C\sum_{i=0}^{j} 2^{-j\ min(q,Q)} + C\sum_{i=0}^{J-j-1} 2^{-(2i+j+1)\ min(q,Q)} \\
=& C\sum_{i=0}^{j} 2^{-j\ min(q,Q)} + 2^{-j\ min(q,Q)} C \sum_{i=0}^{J-j-1} 2^{-(2(i+1))\min(q,Q)} \\
\leq& C\sum_{i=0}^{j} 2^{-j\ min(q,Q)} + 2^{-j\ min(q,Q)} C \leq C2^{-j\ min(q,Q)}(j+1)
\end{aligned}$$

In the next to last step we used the fact that $\sum_{i=0}^{J-j-1} 2^{-2i\ min(q,Q)} < \frac{1}{1-2^{-2\ min(q,Q)}} < \infty$.
Thus we get

$$|z_j'(0)| \leq C_0 2^{-j\min(q,Q)}(j+1).$$

Now we will find the second derivative of the solution i.e. $\overline{z}''$. In this case we multiply the matrix V by the first derivative of $\overline{z}$.

Since $\overline{z}'' = V\overline{z}'$, we have

$$\begin{aligned}
|z_j''(0)| =& |\sum_{i=1}^{J} V_{ji} z_i'(0)| \leq \sum_{i=0}^{J} |V_{ji}||z_i'(0)| \leq C\sum_{i=0}^{J} |2^{-|i-j|q}| \cdot i|2^{-i\ min(q,Q)}| \\
\leq& C\sum_{i=0}^{J} i2^{-(|i-j|+i)\ min(q,Q)} = C\sum_{i=0}^{j} i2^{-j\min(q,Q)} + C\sum_{i=j+1}^{J} i2^{-(2i-j)\ min(q,Q)} \\
=& C\sum_{i=0}^{j} i2^{-j\ min(q,Q)} + C\sum_{i=0}^{J-j-1} i2^{-(2i+j+1)\ min(q,Q)} \\
=& C\sum_{i=0}^{j} i2^{-j\ min(q,Q)} + 2^{-j\ min(q,Q)} C\sum_{i=0}^{J-j-1} i2^{-(2(i+1))\min(q,Q)} \\
\leq& C\sum_{i=0}^{j} i2^{-j\ min(q,Q)} + 2^{-j\ min(q,Q)} C\sum_{i=0}^{\infty} i2^{-(2(i+1))\min(q,Q)} \\
\leq& C\sum_{i=0}^{j} i2^{-j\ min(q,Q)} + 2^{-j\ min(q,Q)} C\sum_{i=0}^{\infty} i2^{-(2(i+1))\ min(q,Q)} \leq C \\
=& C2^{-j\ min(q,Q)}(j+1)^2
\end{aligned}$$

Hence we get

$$|z_j''(0)| \leq C_1 2^{-j\min(q,Q)}(j+1)^2.$$

This proves the lemma. □
Similarly for the third derivative we will get

$$|z_j'''(0)| \leq C 2^{-j\ min(q,Q)}(j+1)^3.$$

Thus we can see that with increasing of the derivative the exponential of $(j+1)$ is increasing. But with increasing j the factor $2^{-j\ min(Q,q)}$ dominates and the values decay rapidly with j. This means that higher derivatives of the solution indeed decay with j at least at time $t=0$. It indicates that the Multiscale timestepping technique might be applicable for this type of problems. A rigorous proof of this is still missing, however.

2.3 Block-version

In this Section we consider similar types of problem as described in the problem description part in Section 2.2 but here the elements $V_{i,j}$ of $\mathbf{V}$ are sparse matrices and the initial vector elements $\overline{Z_j}$ of $\overline{\mathbf{Z}}$ are also vectors. The elements in a particular matrix V_{ij} are all of the same size. Also inside a vector Z_j the elements sizes are similar.

More precisely for the blocked-version we consider *ODEs*

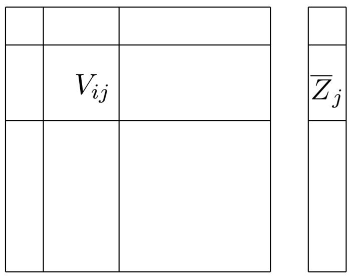
$$\overline{\mathbf{Z}}_t = \mathbf{V}\overline{\mathbf{Z}}, \tag{2.12}$$

where the matrix $\mathbf{V}$ and the initial vector $\overline{\mathbf{Z}}$ are defined as

$$\mathbf{V} = \begin{pmatrix} V_{0,0} & V_{0,1} & \dots & V_{0,J} \\ V_{1,0} & V_{1,1} & \dots & V_{1,J} \\ \vdots & \vdots & \ddots & \vdots \\ V_{J,0} & V_{J,1} & \dots & V_{J,J} \end{pmatrix} \in \mathbb{R}^{2^{J+1}\times 2^{J+1}}, \qquad \overline{\mathbf{Z}}(t) = \begin{pmatrix} \overline{Z_0}(t) \\ \overline{Z_1}(t) \\ \vdots \\ \overline{Z_J}(t) \end{pmatrix} \in \mathbb{R}^{2^{J+1}},$$

where $V_{i,j} \in \mathbb{R}^{2^i\times 2^j}$, $\overline{Z}_j \in \mathbb{R}^{2^j}$, $i,j = 0,1,2,\dots,J$. We assume that the number of non-zero elements in $V_{i,j}$ is $O(\max(2^i,2^j))$. One can then show that the cost of applying V to a vector is $O(J2^J)$.

The structure of the blocked matrix $\mathbf{V}$ and the block vector $\overline{\mathbf{Z}}$ look like the structure given below.



We also assume that the matrix elements and initial vector elements satisfy

$$[|V_{ij}|] \leq C2^{-|i-j|q}, \qquad |\overline{Z_j}(0)|_\infty \leq C2^{-jQ}.$$

Where C is some constant and q, Q are some numbers used to describe the decay rate of the elements of the matrix and initial vector, in the same way as in the non-blocked case.

The matrix norm is defined as

$$[|A|] = \max_{ij} |a_{ij}|; \quad a_{ij} \text{ are the elements of } A.$$

We have already talked about the cost of standard time stepping methods like Forward Euler method for the non-blocked and dense case in Section 2.2.1. It is $O(NJ^2)$, where N is the total number of time steps in the simulations and J is the problem size. For the blocked problem, because of the sparsity, it reduces to $O(N2^J)$, here 2^{J+1} is the problem size. Hence for the non-blocked and dense problems numerical cost $\approx$ "*total number of timesteps* $\times$ (*problem size*)^2" whereas for the blocked and sparse problem it reduces to "*total number of timesteps* $\times$ *problem size*". This is because of the large number of non-zero elements in the system, as matrix-vector multiplication cost is only calculated from the operations of the non-zero elements.

2.3.1 Multiscale technique

To define the Multiscale timestepping technique to this blocked-version, we use the same technique as in Section 2.2.3 for non-blocked case, replacing v_{ij} by V_{ij}.

The H matrix derived from $\mathbf{V}$ looks like

$$H_j = \begin{pmatrix} V_{0,0} & 2^1 V_{0,1} & \dots & 2^j V_{0,j} & \dots 0 \\ 2^1 V_{1,0} & 2^1 V_{1,1} & \dots & 2^j V_{1,j} & \dots 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 2^j V_{j,0} & 2^j V_{j,1} & \dots & 2^j V_{j,j} & \dots 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots \\ 0, & 0 & \dots & 0 & \dots 0 \end{pmatrix}; \qquad j = 0, 1, 2, ..., J. \tag{2.13}$$

Then the Multiscale timestepping technique with Forward Euler method is

$$\mathbf{Z}^{n+1} = \mathbf{Z}^n + \triangle t H_{j(n)} \mathbf{Z}^n,$$

where $j(n) = 0, 1, 0, 2, ..$ is defined in Section 2.2.3 by the equation (2.6).

2.4 Costs

In the introductory example in Section 2.2.2 we chose the time step such that there were 2^J time steps required to achieve the solution at $t = 2^J \triangle = T$. In half of the total time steps, in every odd step, we use only the first top diagonal element matrix H_0 and the top vector element Z_0. In every fourth iteration i.e. one-fourth of our total time steps we are calling the H_1 matrix and the first three elements of the vector and so on. Finally we applied the whole matrix to the whole vector only in the last time step.

To compute the total cost of our Multiscale timestepping technique for a general J in the blocked case we consider,

i) $N =$ Number of time steps. Here $N = 2^J$.

ii) C_j is the cost of applying $(I + \triangle t H_j)$. Here $C_j \approx O(j2^j)$, as the application of $(I+\triangle t H_j)$ is a matrix-vector multiplication with a sparse matrix of size $2^{j+1} \times 2^{j+1}$.

So the total cost of our technique by using Forward Euler method looks like

$$\begin{aligned}
Cost &= C_0\frac{N}{2} + C_1\frac{N}{4} + C_2\frac{N}{8} + C_3\frac{N}{16} + \cdots + C_{J-1}\frac{N}{2^{J-2}} + C_J \\
&= C_0\frac{N}{2\cdot 2^0} + C_1\frac{N}{2\cdot 2^1} + C_2\frac{N}{2\cdot 2^2} + C_3\frac{N}{2\cdot 2^3} + \cdots + C_{J-1}\frac{N}{2\cdot 2^{J-1}} + C_J \\
&= \sum_{j=0}^{J-1} C_j\frac{N}{2\cdot 2^j} + C_J = \sum_{j=0}^{J-1} C_j\frac{2^J}{2\cdot 2^j} + C_J = \frac{2^J}{2}\sum_{j=0}^{J-1}\frac{C_j}{2^j} + C_J \\
&= \frac{2^J}{2}\sum_{j=0}^{J}\frac{C_j}{2^j} = \frac{2^J}{2}\sum_{j=0}^{J}\frac{O(j2^j)}{2^j} = \frac{2^J}{2}\sum_{j=0}^{J} j = O(J^2 2^J) = O(N(log_2 N)^2).
\end{aligned}$$

This is much faster than the cost for standard time stepping method derived above, namely $O(N2^J) = O(N^2)$.

2.5 Non-linear structured problem

In this part we have worked with the structured blocked version problem described in Section 2.3 and added a non-linear part. We define the time dependent non-linear problem as:

$$\mathbf{z}_t = \underbrace{V\mathbf{z}}_{\text{linear part}} + \underbrace{\mathbf{z}\mathbf{z}^T b}_{\text{non-linear part}} \implies \mathbf{z}_t = \mathbf{F}(\mathbf{z}). \tag{2.14}$$

Where V is some structured matrix and $\mathbf{z}$ is the solution vector as described in the problem description Section 2.1 and b is either a constant vector or a vector depending on t. This represents the simplest quadratic non-linear equation.

2.5.1 Multiscale timestepping technique for non-linear structured problems

For the non-linear structured problem we used the same technique as the blocked version problem in Section 2.3.1. The difference between this non-linear problem and the linear problem in Section 2.3 is that for the non-linear equation (2.14) we add the non-linear part with the equation (2.12) in every time step. We use the same H_j matrix as in Section 2.3 and the non-linear function $\mathbf{H}(\mathbf{z}, j(n))$ defined as follows.

Let $F(z) = zz'b = (F_0(z), F_1(z), \dots, F_J(z))^T$. Then we set

$$\mathbf{H}(\mathbf{z}, 0) = \; F_0(\mathbf{z}_0, 0, 0, \dots) = \mathbf{z}_0\mathbf{z}_0 b_0$$

$$\mathbf{H}(\mathbf{z}, 1) = \begin{cases} F_0(\mathbf{z}_0, 2\mathbf{z}_1, 0, 0, \dots, 0) = \mathbf{z}_0(\mathbf{z}_0 b_0 + 2\mathbf{z}_1^T b_1) \\ 2F_1(\mathbf{z}_0, \mathbf{z}_1, 0, 0, \dots, 0) = 2\mathbf{z}_1(\mathbf{z}_0 b_0 + \mathbf{z}_1^T b_1) \end{cases}$$

$$\mathbf{H}(\mathbf{z}, 2) = \begin{cases} F_0(\mathbf{z}_0, 2\mathbf{z}_1, 4\mathbf{z}_2, 0, \dots, 0) = \mathbf{z}_0(\mathbf{z}_0 b_0 + 2\mathbf{z}_1^T b_1 + 4\mathbf{z}_2^T b_2) \\ 2F_1(\mathbf{z}_0, \mathbf{z}_1, 2\mathbf{z}_2, 0, \dots, 0) = 2\mathbf{z}_1(\mathbf{z}_0 b_0 + \mathbf{z}_1^T b_1 + 2\mathbf{z}_2^T b_2) \\ 4F_1(\mathbf{z}_0, \mathbf{z}_1, \mathbf{z}_2, 0, \dots, 0) = 4\mathbf{z}_2(\mathbf{z}_0 b_0 + \mathbf{z}_1^T b_1 + \mathbf{z}_2^T b_2) \end{cases}$$

and so on.

For this non-linear structured problem the Multiscale timestepping technique with Forward Euler method is

$$\mathbf{Z}^{n+1} = \mathbf{Z}^n + \triangle t H_{j(n)} \mathbf{Z}^n + \triangle t \mathbf{H}(\mathbf{Z}^n, j(n)),$$

where $j(n) = 0, 1, 0, 2, ..$ is defined in Section 2.2.3 by the equation (2.6).

Chapter 3

Application to PDEs

To apply our Multiscale timestepping technique to $PDEs$ we have worked with the linear advection equation and advection diffusion equation. First of all we have discretized these equations only in space by a well known semi-discretization method. After that we have transformed the discretization matrix and the solution vector into a wavelet basis. The structure of this wavelet basis matrix and the solution vector is similar to the structured matrix and solution vector described in Section 2.3.

3.1 Advection equation

The one-dimensional linear advection equation is defined as

$$u_t + au_x = 0; \tag{3.1}$$

with initial condition

$$u(x, 0) = u_0(x)$$

and periodic boundary condition

$$u(0, t) = u(1, t); \quad t > 0,$$

where a is the advection coefficient, $u(t, x)$ is the solution that depends on time as well as the space variables. According to the semi-discretization method, we discretize only in space by using the central finite differences scheme. The semi-discrete scheme for the advection equation (3.1) is as follows:

$$\partial_t u_j = -a\frac{u_{j+1} - u_{j-1}}{2h} \tag{3.2}$$

Then we have got a time dependent problem like

$$u_t = Au, \tag{3.3}$$

where

$$A = \frac{a}{2h}\begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 & -1 \\ -1 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 0 & 0 & 0 & \dots & -1 & 0 \end{pmatrix}, \quad u = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{pmatrix}.$$

A is the space discretization matrix from equation (3.2).

Note that Forward Euler cannot be used here, as Forward Euler and central differences in space leads to unstable discretization. Moreover, it turns out that implicit methods are needed for the MT technique. But the computational cost of an implicit method is higher compared to an explicit method. So for our comparison we will use the explicit Runge-Kutta 4^{th} order method for the direct solution and the implicit Crank-Nicolson method for both the direct and the MT method.

The Crank-Nicolson method and the Runge-Kutta 4^{th} order method for equation (3.3) are given below.

- **Crank-Nicolson method**

$$u^{n+1} = u^n + \frac{\triangle t}{2}(Au^{n+1} + Au^n) \tag{3.4}$$

- **Runge-Kutta 4^{th} order method**

$$u^{n+1} = u^n + \frac{\triangle t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{3.5}$$

where

$$\begin{aligned} k_1 &= \triangle\, tAu^n \\ k_2 &= \triangle\, tA(u^n + \frac{k_1}{2}) \\ k_3 &= \triangle\, tA(u^n + \frac{k_2}{2}) \\ k_4 &= \triangle\, tA(u^n + k_3). \end{aligned}$$

3.2 Advection diffusion equation

The linear advection diffusion equation is represented by

$$u_t = \epsilon u_{xx} + u_x; \tag{3.6}$$

with initial condition

$$u(x,0) = u_0(x)$$

and periodic boundary condition

$$u(0,t) = u(1,t); \quad t > 0,$$

Here ϵ is the diffusion coefficient and the solution $u(t,x)$ depends on the time as well as the space variables as for the advection equation. For this equation we have used the same procedure as the advection equation. In the semi-discretization scheme we used central finite differences for space discretization and then we have implemented the well known Crank-Nicolson method for time stepping.

The semi-discrete scheme for the advection-diffusion equation (3.6) is

$$\partial_t u_j = \epsilon \frac{u_{j+1} + 2u_j + u_{j-1}}{h^2} + \frac{u_{j+1} - u_{j-1}}{2h} \tag{3.7}$$

From equation (3.7) we can formulate a time dependent problem

$$u_t = Au, \tag{3.8}$$

where

$$A = \frac{1}{h^2}\begin{pmatrix} 2\epsilon & \epsilon + h/2 & 0 & 0 & \dots & 0 & \epsilon - h/2 \\ \epsilon - h/2 & 2\epsilon & \epsilon + h/2 & 0 & \dots & 0 & 0 \\ 0 & \epsilon - h/2 & 2\epsilon & \epsilon + h/2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \epsilon + h/2 & 0 & 0 & 0 & \dots & \epsilon - h/2 & 2\epsilon \end{pmatrix}, \quad u = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{pmatrix}.$$

.

The **Crank-Nicolson** method of equation (3.8) looks like

$$u^{n+1} = u^n + \frac{\triangle t}{2}(Au^{n+1} + Au^n) \tag{3.9}$$

3.3 Application of wavelets to PDEs

In Section 2.1 we assumed that elements of the matrix V decay away from the diagonal. When solving PDE problems we get a discretization matrix that is banded. Also the diagonal elements and the initial data sizes are of roughly the same size. So to implement our Multiscale timestepping technique, we need to transform the discretizatiion matrix in such a way that we get a matrix that has similar decay properties as the problem in Section 2.1. Using wavelets we can transform the discretization matrix A to a matrix that almost follow the decay properties and the initial data follow precisely the decay properties.

The implementation procedure is to apply the Discrete Wavelet Transform (W) to the semi-discrete problem

$$u_t = Au. \tag{3.10}$$

The matrix W is orthonormal, $W^TW = I$, so we get

$$\begin{aligned} u_t =& Au \\ \implies Wu_t =& WAW^TWu \\ \implies w_t =& Vw; \qquad \text{where we defined } Wu = w \text{ and } WAW^T = V. \end{aligned}$$

(See Figure 3.1 for an example of of w in the Haar wavelet case.)
Note that the eigenvalues of $A = WAW^T$ are the same as those of A which implies that the condition number of A will not be affected by the Discrete Wavelet transform [5].

The new variable w is the wavelet transform of u, the pointwise approximation of the continuous solution. Instead of pointwise values, w contains differences of local averages on different levels, corresponding to the contribution to the function on different scales. As an example we consider the **Haar** wavelet transform which can be roughly expressed as:

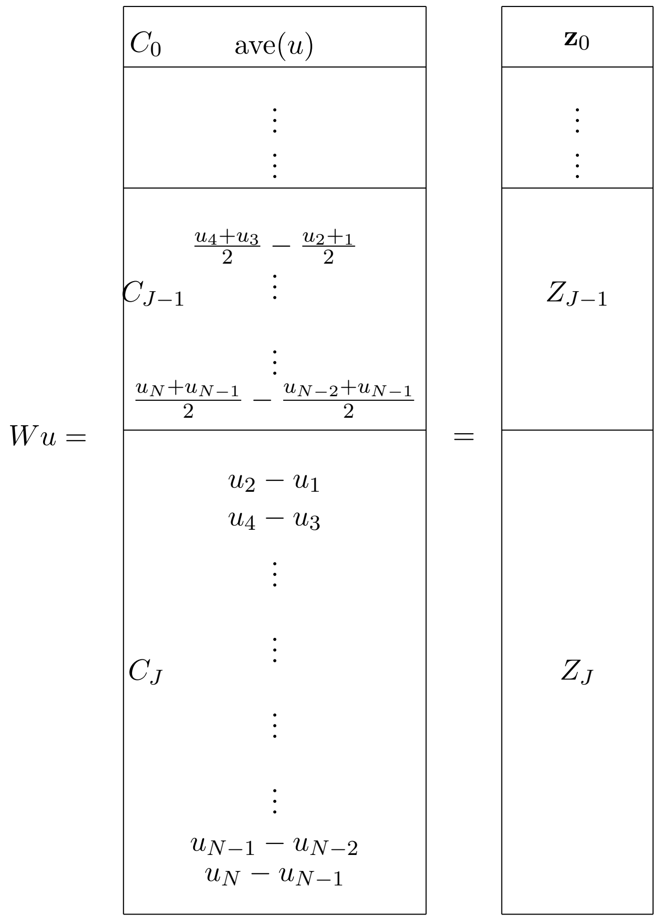


Figure 3.1. Haar wavelet transformation of the solution vector.

Types of wavelet	q	Q
Haar	0.50	1.5
Daubechies, 2 vanishing moments	1.05	2.5
Daubechies, 3 vanishing moments	1.59	3.5
Daubechies, 4 vanishing moments	2.13	4.5
Daubechies, 5 vanishing moments	2.49	5.5
Daubechies, 6 vanishing moments	2.70	6.5
Daubechies, 7 vanishing moments	2.97	7.5
Daubechies, 8 vanishing moments	3.28	8.5
Daubechies, 9 vanishing moments	3.61	9.5
Daubechies, 10 vanishing moments	3.88	10.5
Coifman, 2 vanishing moments	1.55	2.5
Coifman, 4 vanishing moments	2.27	4.5
Coifman, 6 vanishing moments	2.81	6.5

Table 3.1. Wavelets transform and its (q, Q)-values for advection equation.

By the wavelet transformation the matrix blocks of $V = WAW^T$ can be estimated as

$$[|V_{j,k}|] \leq C2^{-|j-k|q+Pmin(j,k)}, \tag{3.11}$$

where P is the order of the PDE (advection $P = 1$, diffusion $P = 2$) for wavelets which belong to the Sobolev space $H^q(\mathbb{R})$, see [2]. The initial vector elements can similarly be estimated as

$$|Z_j| \leq C2^{-jQ},$$

where $Q - \frac{1}{2}$ is the number of vanishing moments of the wavelet.

By explicit computation of some large V matrices based on the advection equation we find experimentally the q-values listed in Table 3.1 for different wavelet types. In the experiment, those values do not change much when changing the size of the V matrix.

As for the sparseness of V, if a banded matrix such as those used in Section 3.1, 3.2 is transformed with wavelet basis then the number of non-zero elements of the transformed matrix is slightly increased compared to the original matrix. The number of non-zero elements depend on the width of the wavelet mask. A transform matrix with a larger mask has more non-zero elements and vice-versa. Higher order wavelets with more vanishing moments have in general a wider mask.

The 'spy' view of the discretization matrix A and its wavelet based transformation matrices V are given in Figure 3.2-4. The figures show how much the number of non-zero elements are increased when changing a diagonal sparse matrix to wavelet basis and how much it increases with higher order wavelets.

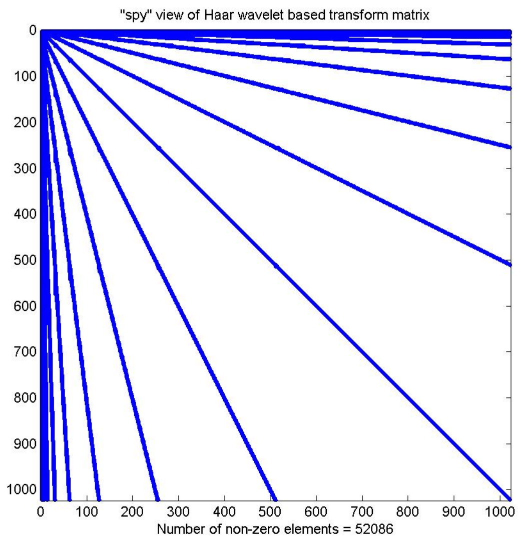


Figure 3.3. 'Spy' view of Haar wavelet based transform matrix of A..

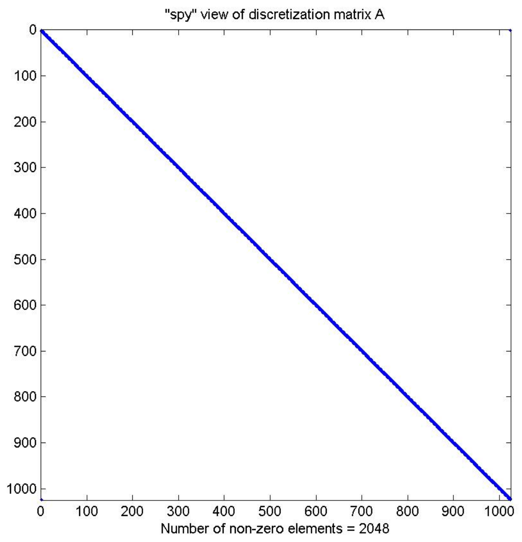


Figure 3.2. 'Spy' view of the matrix $A(1024 \times 1024)$ of advection equation 3.3.

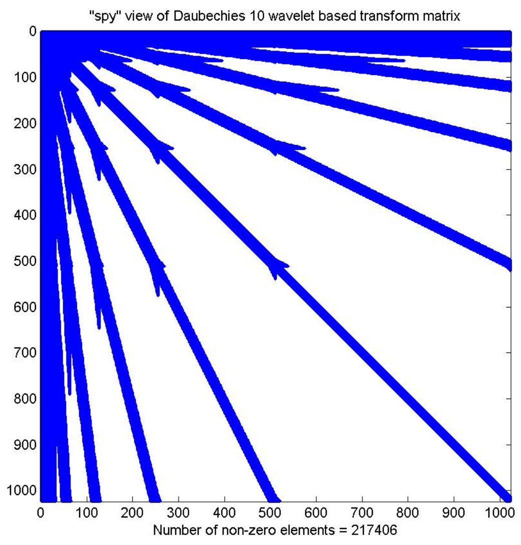


Figure 3.4. ‘Spy’ view of Daubechies 10 wavelet based transform matrix of A.

Chapter 4

Numerical experiments

We will numerically evaluate our Multiscale timestepping technique for *ODEs* and *PDEs* to find out the order of accuracy and the computational cost. For all experiments in this paper we compare the solution obtained by a standard method such as Forward Euler, Backward Euler, Runge-Kutta *2nd* order method, Runge-Kutta *4th* order method, Crank-Nicolson method with the solution obtained by using the Multiscale timestepping technique. For example the solution obtained by the FE method is compared with the solution obtained by MT-FE method. We consider the max norm of the difference as the error. In all experiments the accuracy is considered in terms of the time step $\triangle t$.

To evaluate the cost, we have calculated the CPU time by using the “tic - toc" Matlab commands. For the advection diffusion equation we have taken the average time of five simulations and for all other experiments we have taken the average time of ten simulations.

Our goal is to implement our Multiscale timestepping technique for *PDEs*, but we start with the simple structured problem described in Chapter 2.

4.1 Structured problem

In this section we will make experiments with the simple structured problems (2.1) and (2.12) described in Chapter 2 in the problem description part of Section 2.1 and in blocked version part of Section 2.3 respectively. In the non-blocked case the matrix elements and the initial condition elements are scalars. Since the discretization matrices of the *PDEs* in Chapter 3 are blocked matrices similar to those of Section 2.3 in Chapter 2, our main focus on experiments in this section is on the blocked-version problem but at the beginning we make some experiments with the non-blocked problem.

The matrix structure depends on the values q and Q respectively. It means that for different sets of (q, Q) values the method shows different characteristics of the solution. So it is important and interesting part of the experiment, how the solution changes with changing the (q, Q) values.

In all experiments considered we take $v_{j,k} = 2^{-q|j-k|}$ in the non-blocked case and each element in $V_{j,k}$ has the value $2^{-q|j-k|}$ in the blocked case. Moreover for initial data we take $z_j(0) = 2^{-jQ}$ in the non-blocked case and each element in $Z_j(0)$ has the value 2^{-jQ} for the blocked version. The difference between the blocked version and the non-blocked version elements is that for the non-blocked case each element value is different, whereas for the blocked version, inside the block the elements value are the same but different blocked elements contain different values. With this choice of elements, when the size of the problem (J) increases, old values are kept the same. For example for $J = 8$ and $J = 9$ the elements over the matrix area for $J = 8$ are the same for that area when $J = 9$, the only difference is that for larger problem new elements are added in new positions.

4.1.1 Non-blocked problem

In the non-blocked case matrices V_J are dense. We have discussed the structure of the elements $|v_{j,k}| \leq C2^{-|j-k|q}$ of V in problem (2.1) in Section 2.1. In numerical experiments with the non-blocked problem we have considered smaller problem with maximum size $J = 19$, *i.e.* V_J is a 20×20 matrix. Multiscale timestepping technique with only the Forward Euler time stepping method i.e. only MT-FE technique are applied in this section. For all experiments with this non-blocked problem we have used the time step $\triangle t = 2^{-J}$ and run the simulations until $T_{end} = 1$.

Before all of the experiments, we want to see the solution and its derivatives as a function of time. In this experiment we have chosen $(q, Q) = (1.5, 1.5)$, $V_{j,k} = 2^{-|j-k|q}$ is a 10×10 matrix and the initial condition $z_j(0) = 2^{-jQ}$ is a vector of dimension 10.

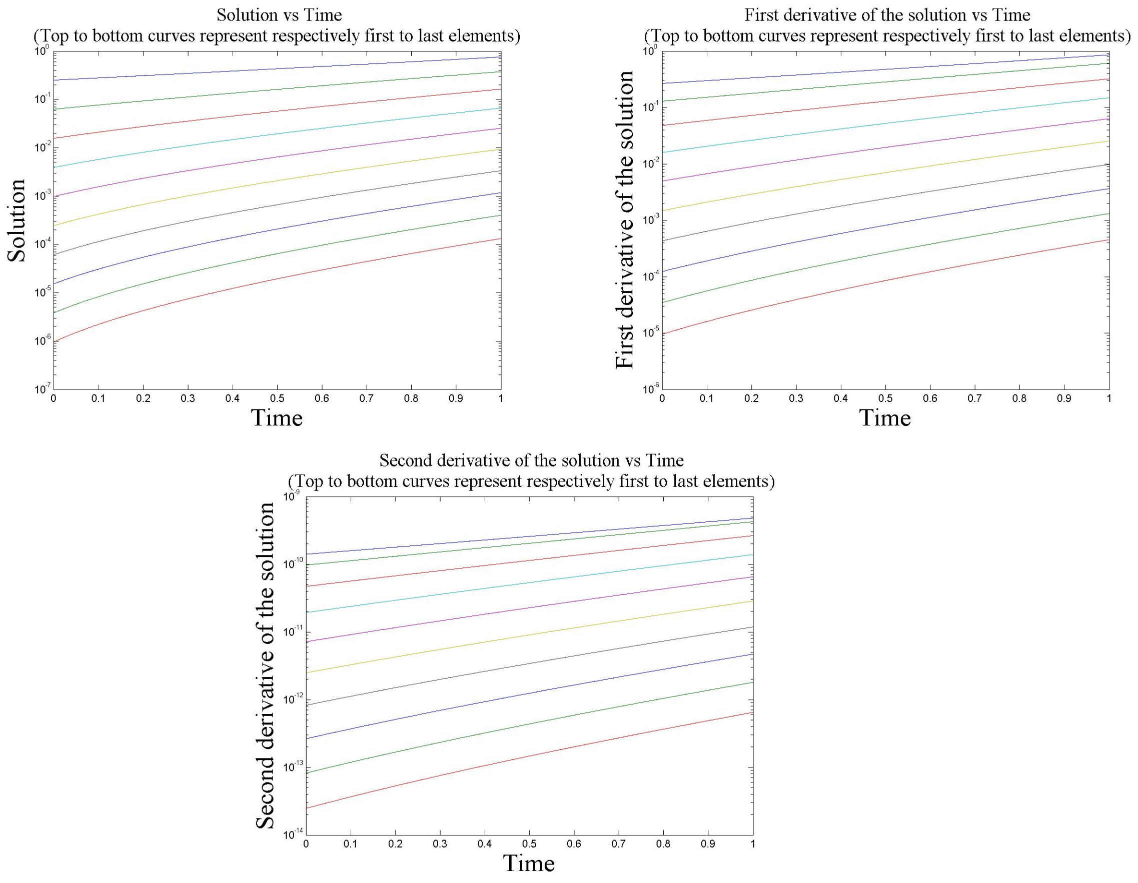


Figure 4.1. Solution vector and time derivatives of the solution for each $\triangle t$.

Though by observing the Figure 4.1 it is difficult to measure the slope exactly but it seems that slopes (time derivatives) become smaller for increasing index j (lies further down). We proved in Lemma 2.2.1 that it was true at $t = 0$, but here we see that it holds for all time $t > 0$. We know that if the solution changes slowly with time, we can use a larger time step. The Multiscale timestepping technique is designed in such a way that the elements of the solution vector with larger index j, are calculated with larger time steps. So Multiscale timestepping technique might be applicable for these type of problems.

Now we will observe how the solutions look like at $t = 1$ and compare with initial data. In this case we have chosen $(q, Q) = (1.5, 1.5)$ and $J = 19$, V is 20×20 matrix.

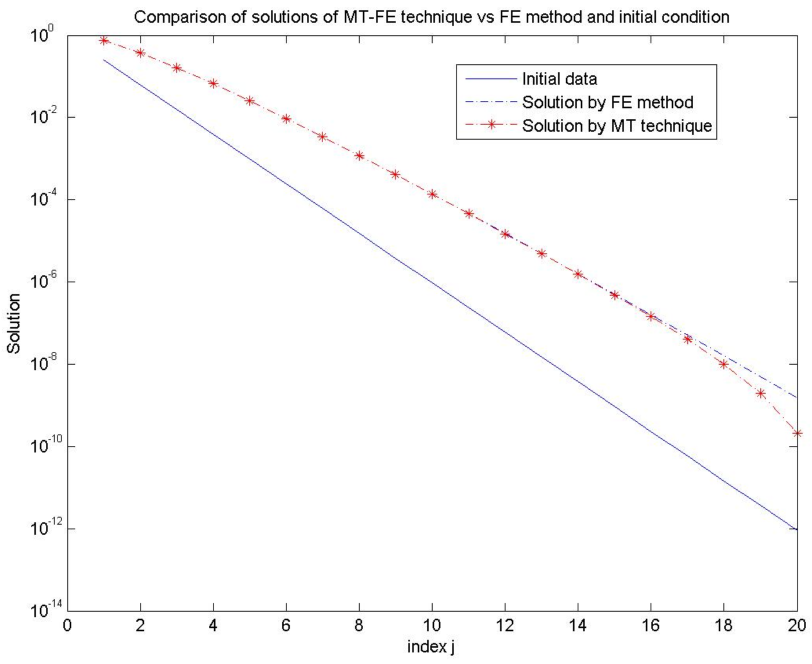


Figure 4.2. Semilogy plot of the solutions and the initial condition.

Figure 4.2 shows the comparison of the solution of the MT-FE technique with the FE method. It also compared the solutions with the initial condition. If we look toward the end of the solution curves, some differences between the solutions are observed and it is because of the properties of the MT technique. We know that for updating the last element of the solution the MT technique iterates only one time. For the second last it iterates only two times and so on, whereas the FE method updates the whole solution vector by iterating with all elements. So in general there will be differences in the last couple of elements of the solutions.

We have already mentioned that the (q, Q) values play an important role for the structured problems. Now we make experiments with the equation (2.1) for different set of (q, Q) values. In these experiments our aim is to note the effect of (q, Q) values on the solutions. We plot the difference between the solutions of $MT - FE$ & FE in max norm, at $t = T_{end} = 1$ as a function of $J = log_2 N$. (Note that here we will consider the max norm of the difference $|u_{MT-FE}(1) - u_{FE}(1)|_\infty$ as error. We know that since $|u_{exact}(1) - u_{FE}(1)|_\infty = O(h) = O(2^{-J})$, then $|u_{MT-FE}(1) - u_{exact}(1)| = O(2^{-J})$ if $|u_{MT-FE}(1) - u_{FE}(1)| = O(2^{-J})$ by the triangle inequality.). The results are given in Figure 4.3.

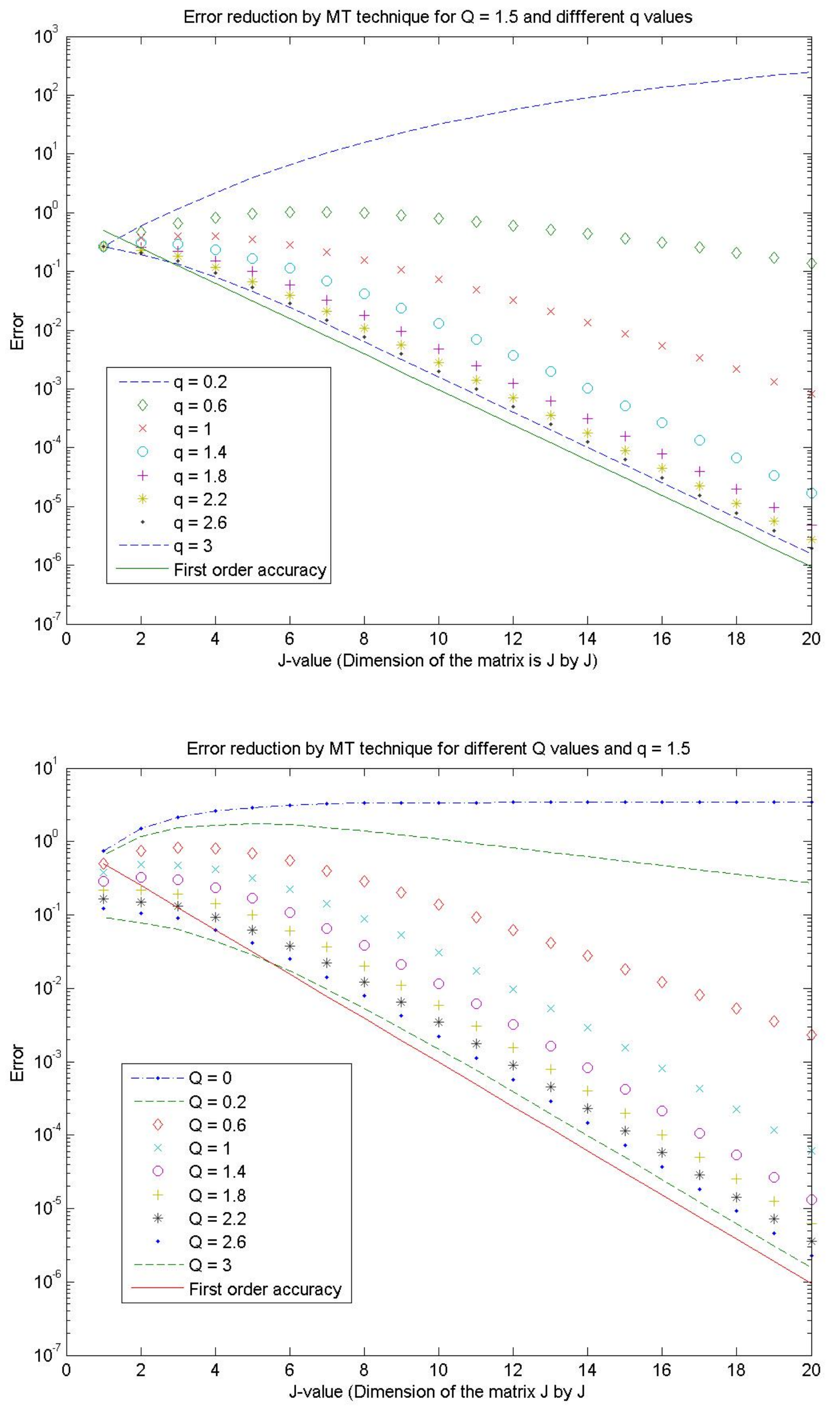


Figure 4.3. Error reduction for different set of (Q, q) values.

From both plots in Figure 4.3 we can see that for this non-blocked toy problem the MT-FE technique reduces the error in all of the cases except if the q values are very small (about $q < 0.5$). The second plot shows that results are not as sensitive to Q values as to q. The reason is that the larger number of elements of the system, the matrix elements are generated by q whereas Q is used to formulate the initial vector elements. The Figure 4.3 also indicates that for this small problem the MT-FE technique ensures about the first order of accuracy in error reduction for $(q, Q) \geq (1, 1)$. (Note that $\triangle t = 2^{-J}$.)

One of the most interesting points of the MT technique is to reduce the computational cost. Now we examine the computational cost of MT-FE technique for this very small problem. The evaluated result is given in Figure 4.4.

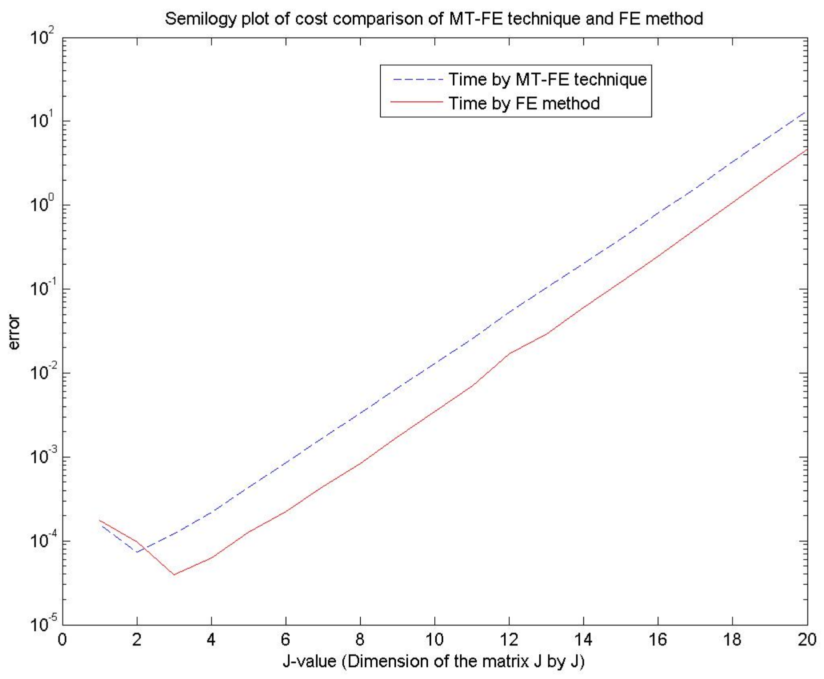


Figure 4.4. Comparison of numerical cost between the standard FE method and the $MT - FE$ technique.

From Figure 4.4 we can see that solving this non-blocked problem the MT-FE technique takes bit more time than the FE method. According to the hypothesis of the Multiscale timestepping technique it should be opposite (although for non-blocked problem the difference is small). We can guess one reason behind it. In the Matlab programs inside the MT-FE technique inner loop in every time from a vector we are picking up a number to fix the size of the problem for that step. It increases the computational cost. Moreover when the size of the problem is not large enough then there is not enough significant difference between the computational cost of the MT-FE technique and the FE method. As this non-blocked tiny problem is a toy problem we leave this point for the next Section.

4.1.2 Blocked problem

In this section we work with the equation (2.12), where the matrix and the initial vector are structured as described in Section 2.3. We have already talked about various characteristics of the solution's for different sets of (q, Q) value. In this section for all of the experiments we use time step $\triangle t = 2^{-J+1}$ for the explicit FE, RK2 and RK4 methods and for the implicit BE method $\triangle t = 2^{-J}$. Also for the BE method we run the simulation until $T_{end} = 0.5$ and for other methods $T_{end} = 1$. So for all of the four cases we run simulations with same number of iterations.

To begin with, we consider $(Q, q) = (1, 1)$. In the previous Section in non-blocked case we have seen the solution's behaviour for different sets of (q, Q) value. Here we start by observing the errors between solutions of the MT technique and some standard *ODE* methods. The computed results are in Figure 4.5.

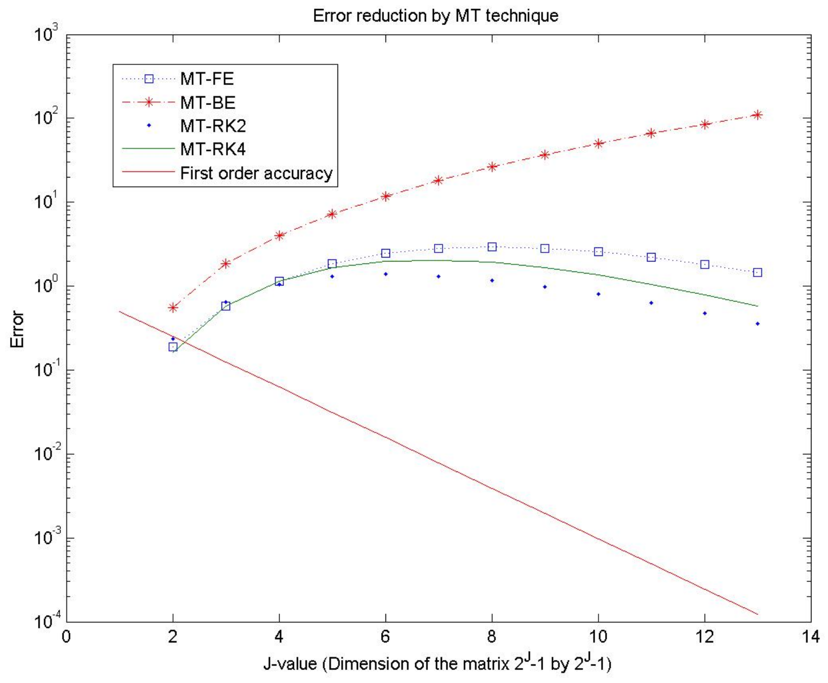
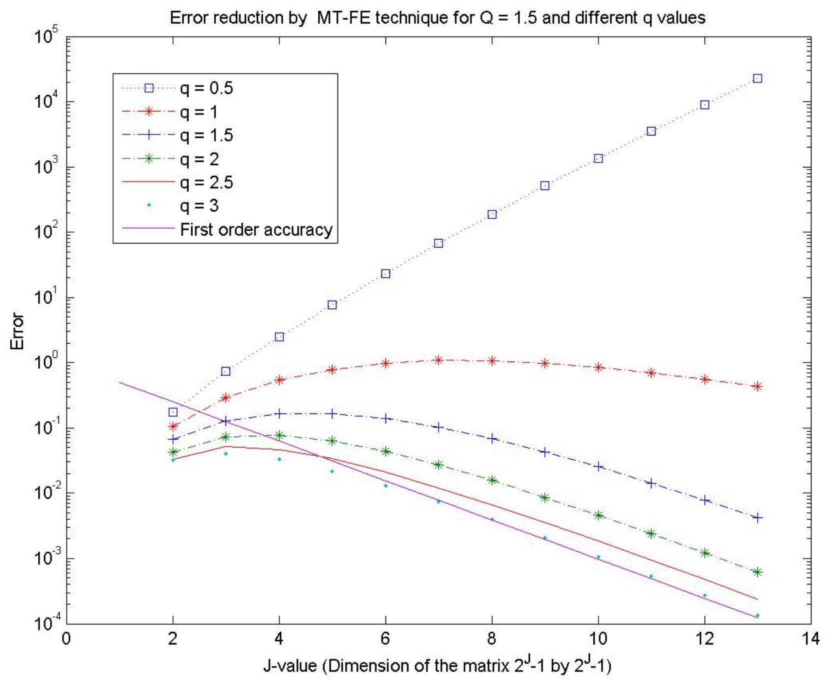


Figure 4.5. Error reduction by MT technique with different standard methods.

In Figure 4.5 we can see that for $(q, Q) = (1, 1)$, with explicit methods the MT technique shows some error reduction whereas as far we observe, with the implicit Euler the error increases.

Now we work with different sets of (q, Q) values. As for the non-blocked case, in one case we consider q constant and different Q values and in another set we consider the opposite, Q is constant and q values are different. Here we implement our MT technique with all of the four standard methods, the FE, BE, RK2 and RK4 and we call them respectively MT-FE, MT-BE, MT-RK2, MT-RK4 technique. For different methods the examined results are given below in Figure 4.6-9.



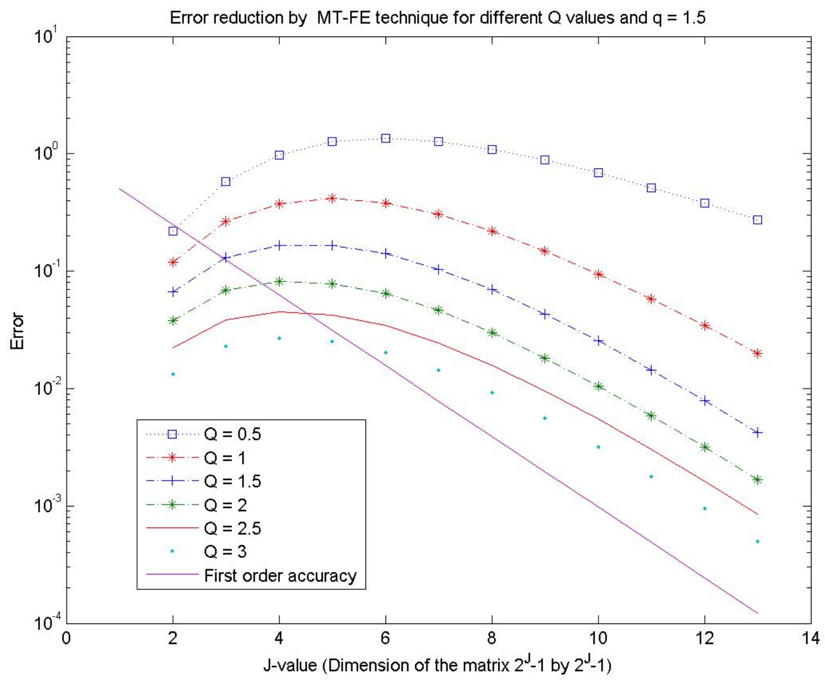
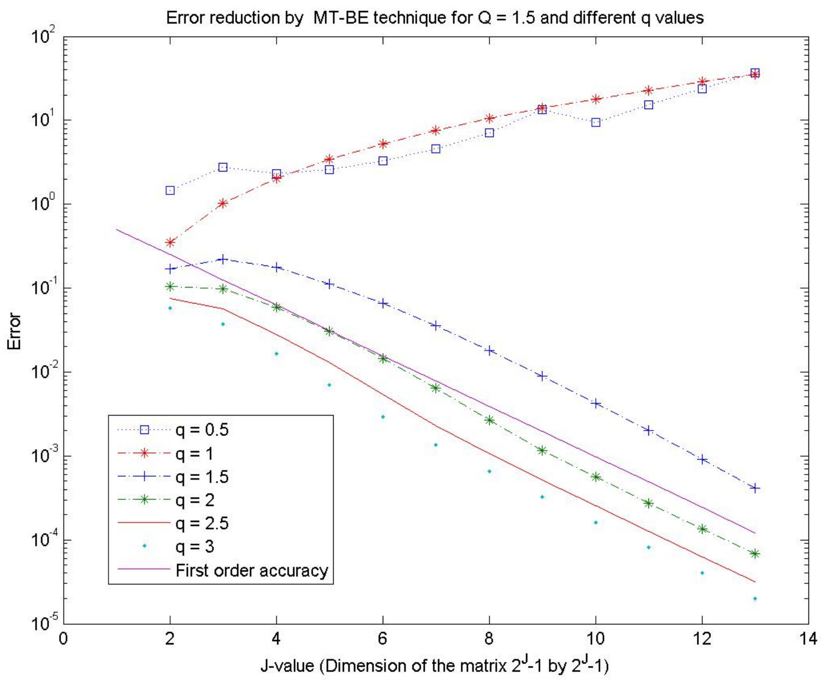


Figure 4.6. Error reduction by $MT - FE$ technique for different set of (Q, q) values.



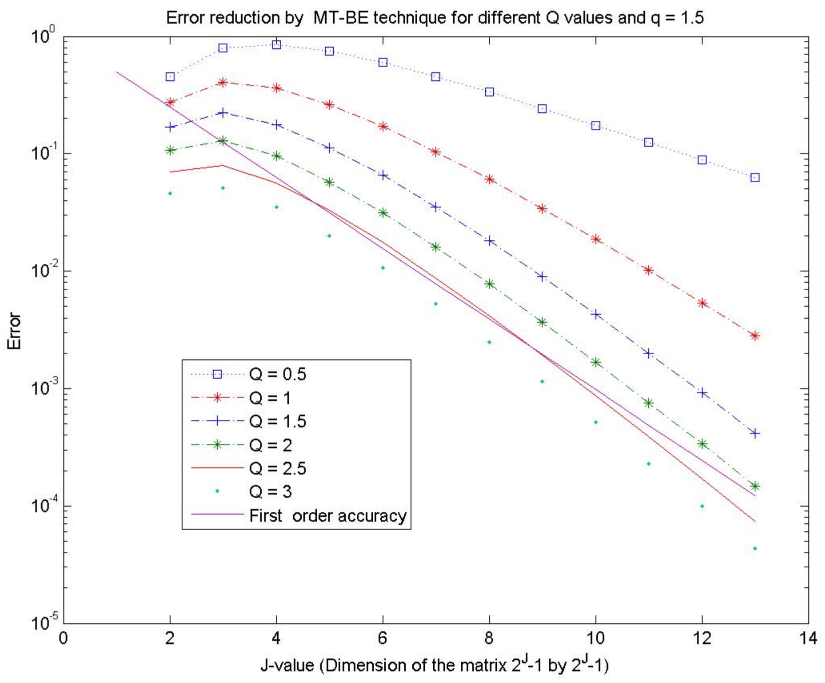
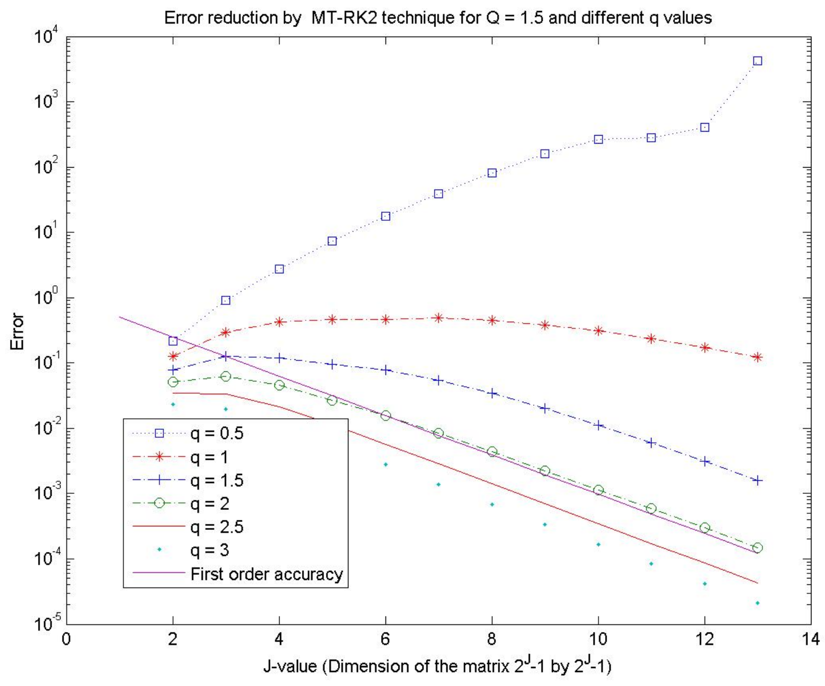


Figure 4.7. Error reduction by $MT-BE$ technique for different set of (Q, q) values.



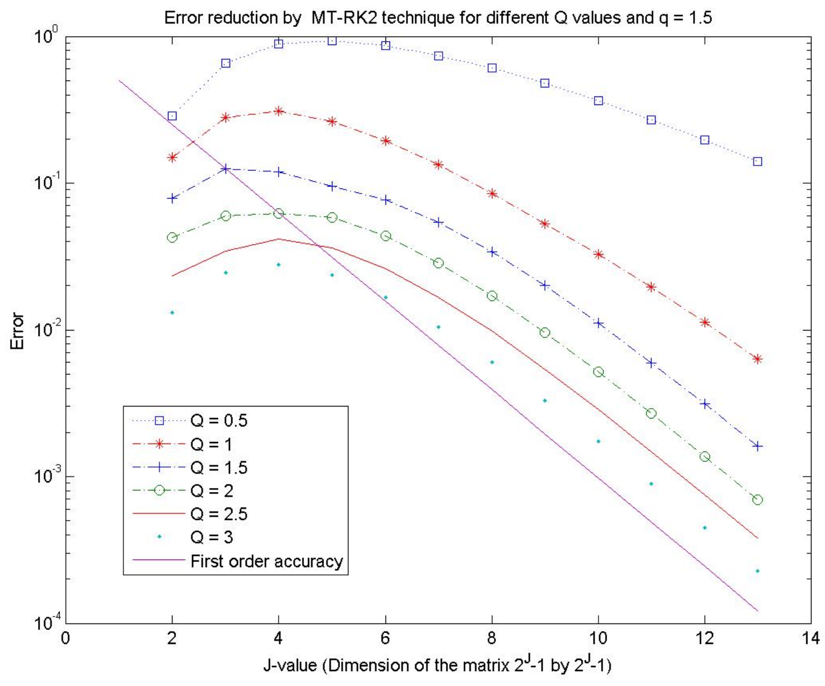
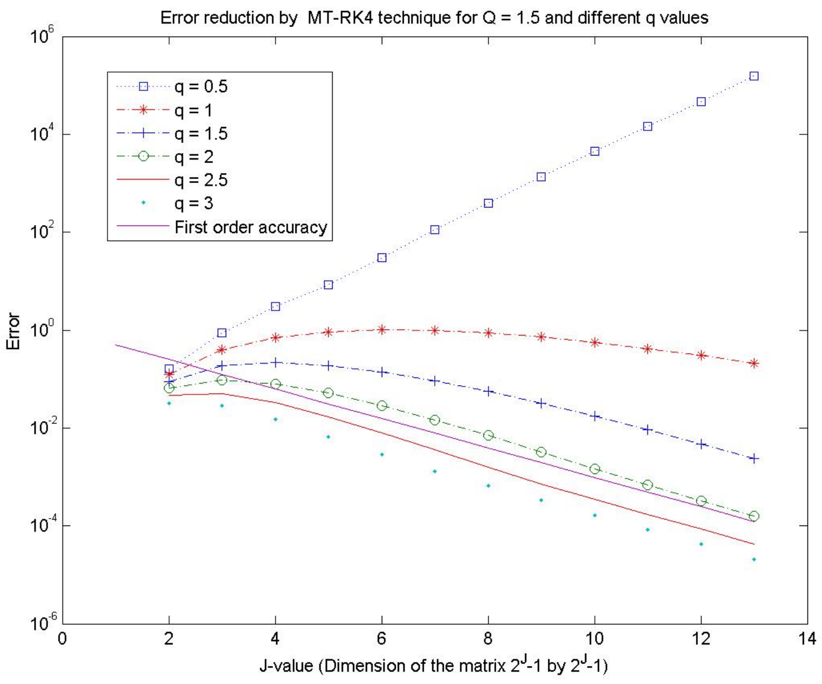


Figure 4.8. Error reduction by $MT - RK2$ technique for different set of (Q, q) values.



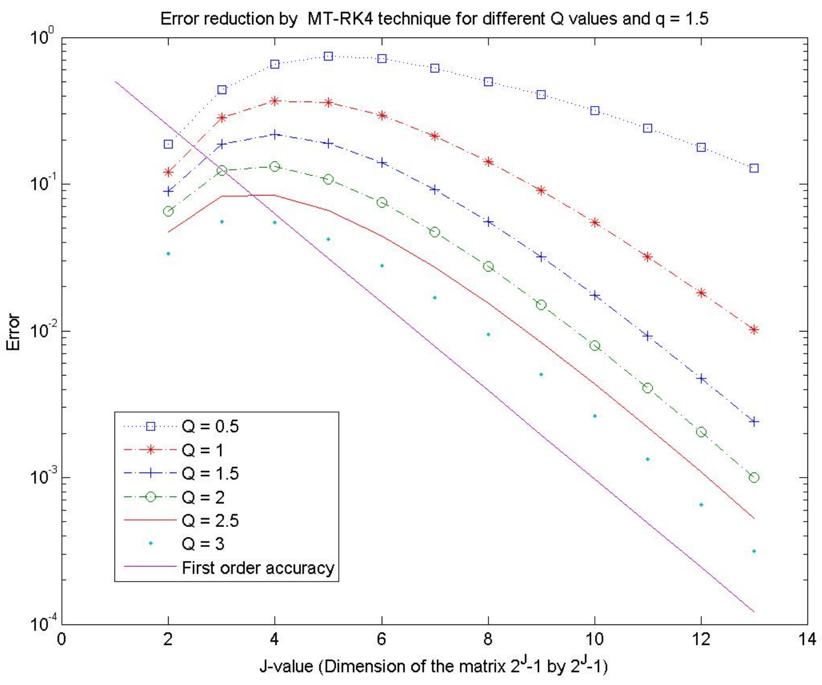


Figure 4.9. Error reduction by $MT - RK4$ technique for different set of (Q, q) values.

We can see that in all of the cases in Figure 4.6-9, the solution is more sensitive to the matrix generator number q compared with the vector generator number Q. The same feature was observed in the previous section for the non-blocked case. This characteristic is expected and the reason mentioned in the previous Section, is that q has more impact on generating the problem compared with Q. Also if we compare in each method the bottom plots show the error reduction in all of the cases whereas on the top plots error reduction is observed only for $q \geq 1.5$. Moreover when $Q = 1.5$ then for any $q > 1.5$, there is a sharp decrease of errors in all of the cases. Also from the top four figures we can see that the matrix generator number q is more sensitive when MT technique is applied with the implicit method.

Cost estimation In this section we evaluate the computational cost of the MT technique with all of the four methods FE, BE, RK2 and RK4 and compare with the standard methods. The numerical results are given in Figures 4.10-13.

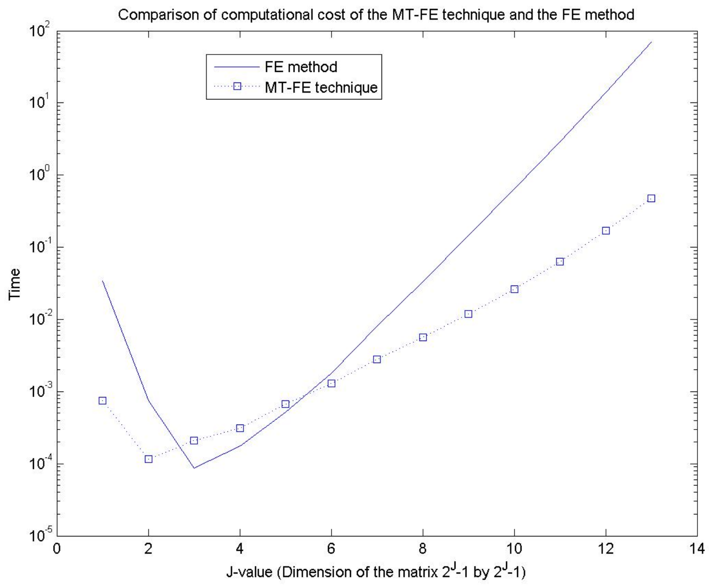


Figure 4.10. Cost evaluation of $MT - FE$ technique and the standard Forward Euler method

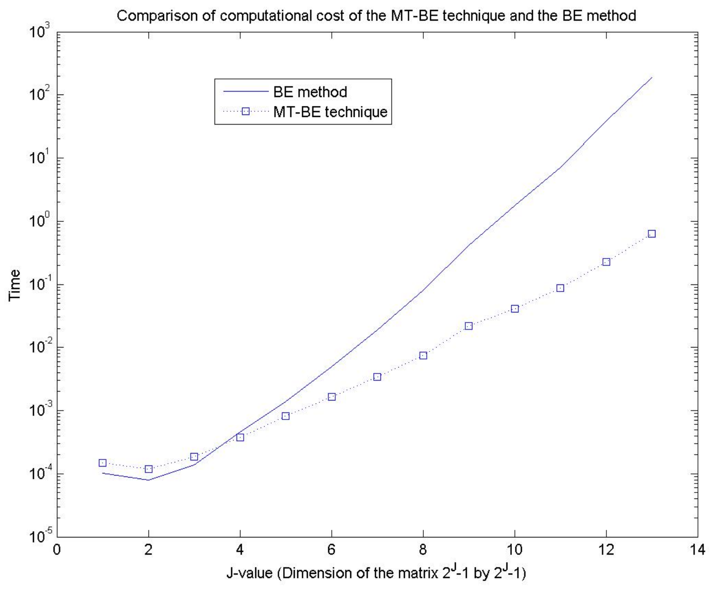


Figure 4.11. Cost evaluation of $MT - BE$ technique and the standard Backward Euler method

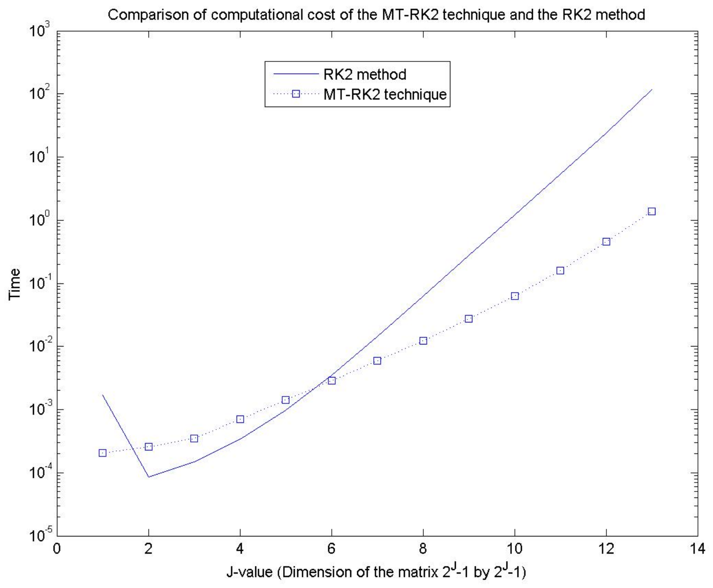


Figure 4.12. Cost evaluation of $MT - RK2$ technique and the Runge-kutta $2nd$ order method

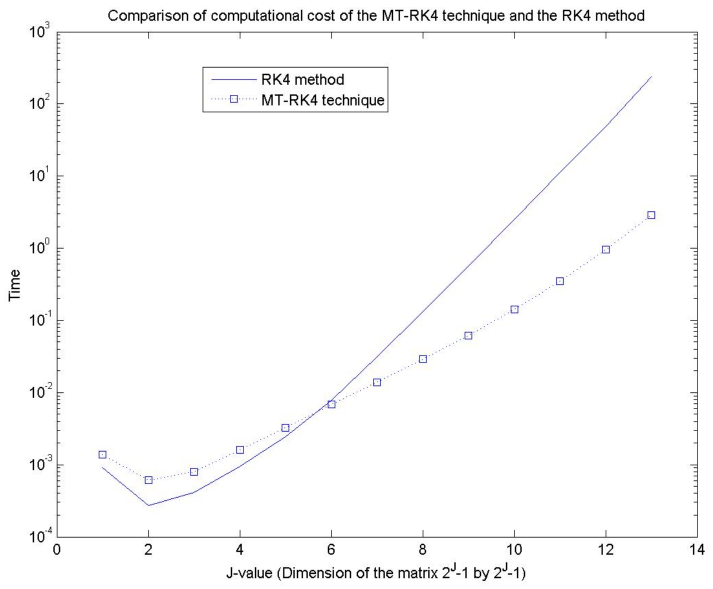


Figure 4.13. Cost evaluation of $MT-RK4$ technique and the standard Runge-kutta $4th$ order method

From Figures 4.10-13 we can see that when the problem sizes are small then the MT technique takes a bit more time compared with the standard methods but after a certain problem size it reverses character. From the Figures we can also see when $J = 13$, with all of the standard methods the MT technique cuts a significant computational cost. The distance between the cost line of MT technique and the cost line of standard methods increased geometrically with increasing problem size.

To find out exactly how much faster the MT technique is, compared with its standard method, for all of the four cases we have calculated the ratio of the computational costs. The ratio is calculated as "computational cost by standard method" divided by "computational cost by MT technique with that standard method".

The cost ratio for all of the four cases are given in Figure 4.14-17.

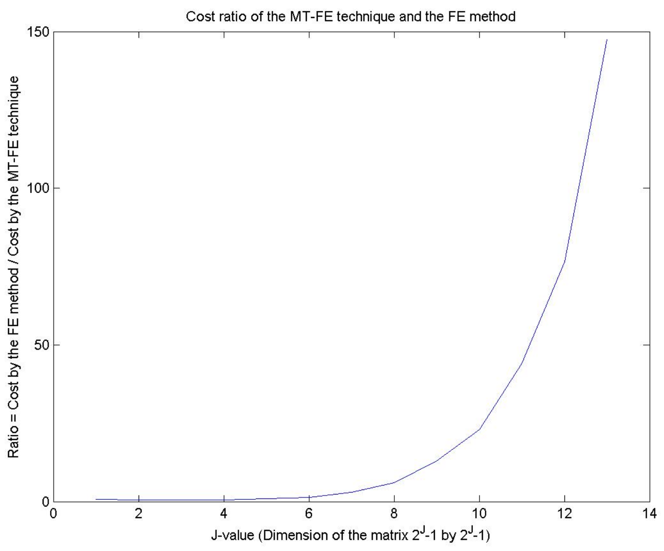


Figure 4.14. Performance achievement by $MT - FE$ technique compared with the standard Forward euler method.

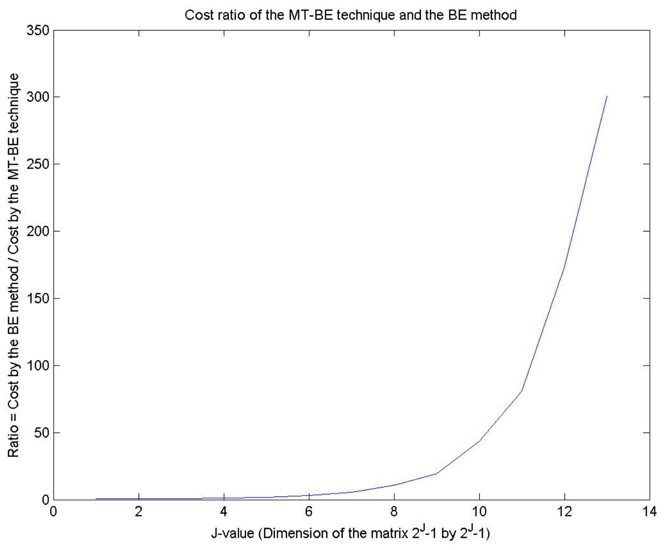


Figure 4.15. Performance achievement by $MT - BE$ technique compared with the standard Backward euler method.

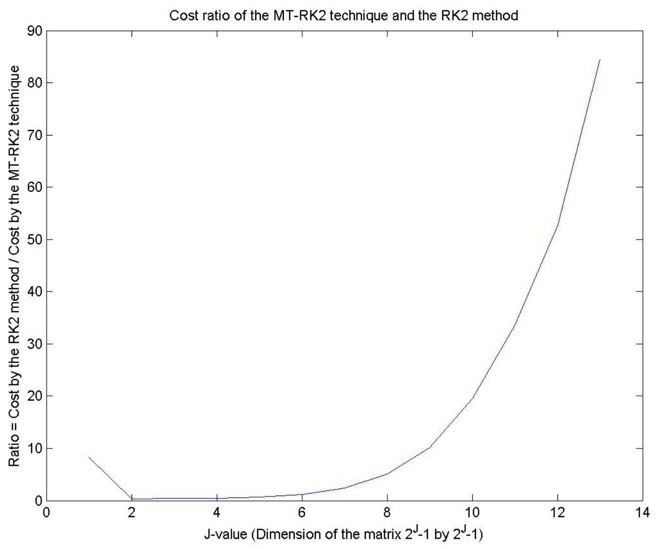


Figure 4.16. Performance achievement by $MT - RK2$ technique compared with the standard Runge-kutta $2nd$ order method.

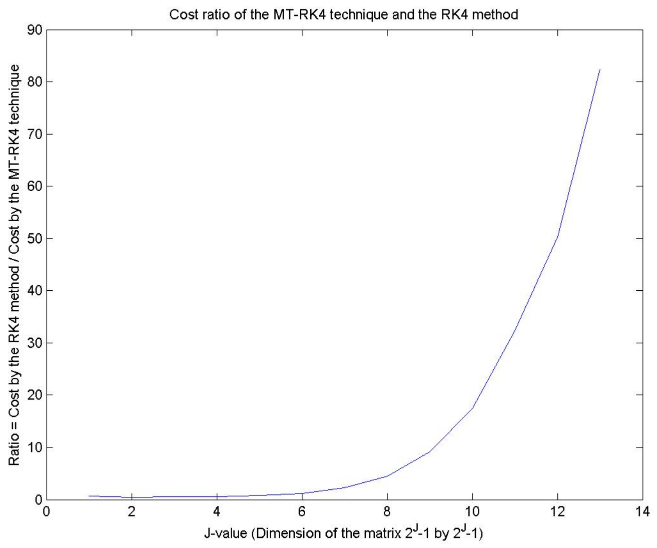


Figure 4.17. Performance achievement by $MT - RK4$ technique compared with the standard Runge-kutta $4th$ order method.

For all of the four cases in Figure 4.14-17, a significant cost reduction by MT technique are observed. Here among all of the four techniques the MT-BE technique cuts the most computational cost. Moreover if we look the the right sides of all the four curves we can see that after $J = 10$, the curves move almost vertically. The most interesting point is that in all of the four cases if we compare for $J = 12$ and $J = 13$ the curves gone about half way of the whole path. It indicates that the performance gain increases with $O(2^J) = O(N)$. This agrees with the theory in Section 2.4.

4.2 Linear PDEs

In this section we have implemented our Multiscale timestepping technique for the linear advection equation as well as the linear advection diffusion equation. To solve these equations, first of all we have discretized the equations by central differences in space. Then we have transformed the matrices and the initial conditions into a wavelet basis. Finally we have solved the transformed problems by implementing our Multiscale timestepping technique based on $RK4$ and CN. [See Chapter 3.]

The wavelet transform converts the matrices and the vectors into matrices and vectors with properties similar to those that are discussed for the structured problems in Section 2.3. Also, according to the wavelet properties, if we transform the spatial discretization matrices into a wavelet basis matrices, then the number of non-zero elements of the wavelet basis matrices increase with the number of vanishing moments of the wavelets. This is discussed and shown by 'spy' view pictures in Section 3.3. It means that for higher order of wavelets, information is more distributed over the whole domain of the matrices. So for different order of wavelets the solutions show different characteristics. For example if we choose the Daubechies wavelets with most vanishing moments, the information is the most distributed.

In this section for both of the $PDEs$ we make numerical experiments with Daubechies wavelets with different number of vanishing moments. In our experiments we also consider Coifman wavelets of order 4. Moreover for these experiments we have chosen a suitable exponential (almost) periodic function $u = e^{-100(x-0.5)^2}$, $x \in [0, 1]$ as initial condition. Also we have used the size of step in space $\triangle x = \frac{1}{N}$ and time step size $\triangle t = 2 * T_{end}/N$, where $N = 2^J - 1$ is used to represent the size of the problem. Also for these tests we choose $T_{end} = 0.5$.

4.2.1 Advection equation

We know that the advection equation is preferably solved by explicit methods. However for our Multiscale timestepping technique we must use implicit methods, as different sizes of time step are used in this technique and the time step restriction is much more relaxed in implicit methods compared with explicit methods. For the advection equation to make a fair comparison specially for the computational cost, we used the explicit $RK4$ method as a standard method, while the Multiscale timestepping technique is implemented with the implicit Crank-Nicolson method. Note that the eigenvalues of the matrix A are purely imaginary. The $RK4$ method is explicit but unlike FE, the stability region includes part of the imaginary axis as well as some part of the right half plane. When $h \to 0$, λh therefore lies in the stability region (λ is the eigenvalue). We also used the Crank-Nicolson method as standard method, because of the interest for the next Section for the diffusion equation and for future considerations.

Here we have solved the advection equation (3.1) with the advection coefficient $a = 1$. Results for different types of wavelets are given below, showing the order of accuracy of the MT-CN technique.

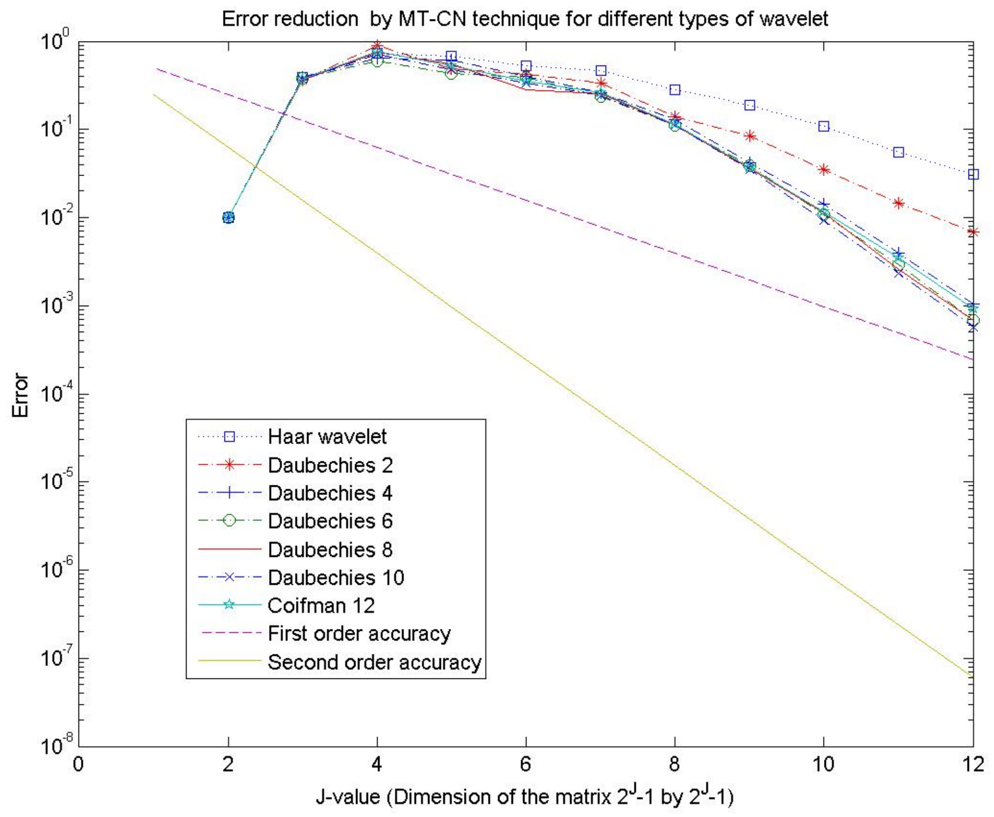


Figure 4.18. Order of accuracy of MT-CN technique for different types of wavelets when solving the linear advection equation (3.1).

From the Figure 4.18 we can see that the MT-CN technique shows first order accuracy in error reduction when the discretization matrix and the initial vector are transformed by lower order wavelets. On the other hand if we look the curves for $J \geq 8$ and for higher order wavelets it shows almost the second order of accuracy. Moreover in this experiment the Coifman wavelet 12 and the Daubechies wavelets 4 with four vanishing moments in both of two different types of wavelet give very similar result in error reduction.

Computational cost We will here compare the computational cost of the MT-CN technique with a standard RK4 method. Upon transforming the discretization into a wavelet basis, the cost of using standard RK4 may change and we also want to estimate this. In this experiment we have therefore measured the computational cost for all of the three cases: cost of standard RK4 method for the original discretization problem; cost of standard RK4 for the wavelet basis problem and cost by the MT-CN technique.

Moreover we have seen in Chapter 3 that the number of non-zero elements of the wavelet basis matrix depends on the order of the wavelets. It means computational cost is related to the order of the wavelets. Here we consider two different cases: Haar wavelets and Daubechies 4 wavelets. The experimental computational costs are given in Figure 4.19-20.

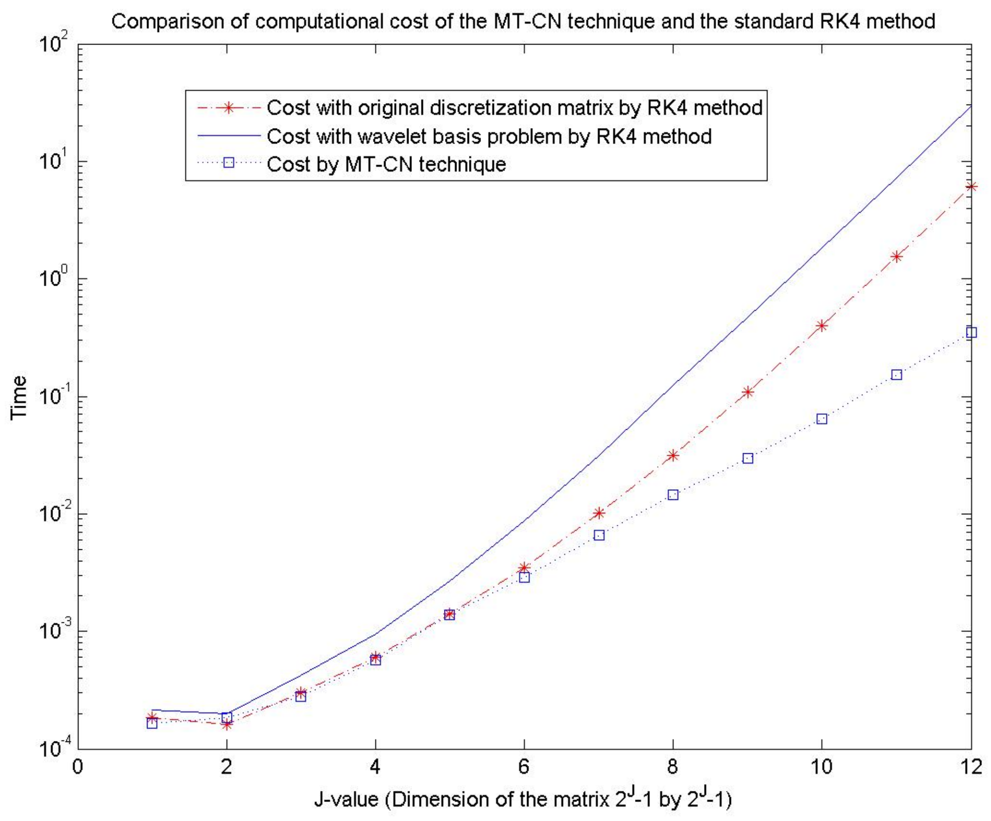


Figure 4.19. Comparison of time evaluation of the MT-CN technique and the RK4 method when solving the advection equation (3.1). The problem is transformed by the Haar wavelets.

In Figure 4.19 we can see that though for smaller problems the $MT-CN$ technique shows the same numerical cost with the standard $RK4$ method for the original discretization problem but with increasing the size ($J > 6$) of the problem some cost reduction is observed. Finally if we look at the end of the curves we can see that the cost by MT-CN technique is significantly lower than the standard method.

Moreover if we compare the Figure 4.20 with the Figure 4.19 we can see that the MT-CN technique is not as fast as the Figure 4.19 and it is because of the wavelet order. As higher order wavelet transformation matrix contained more non-zero elements and the cost is related with the number of non-zero elements of the matrix.

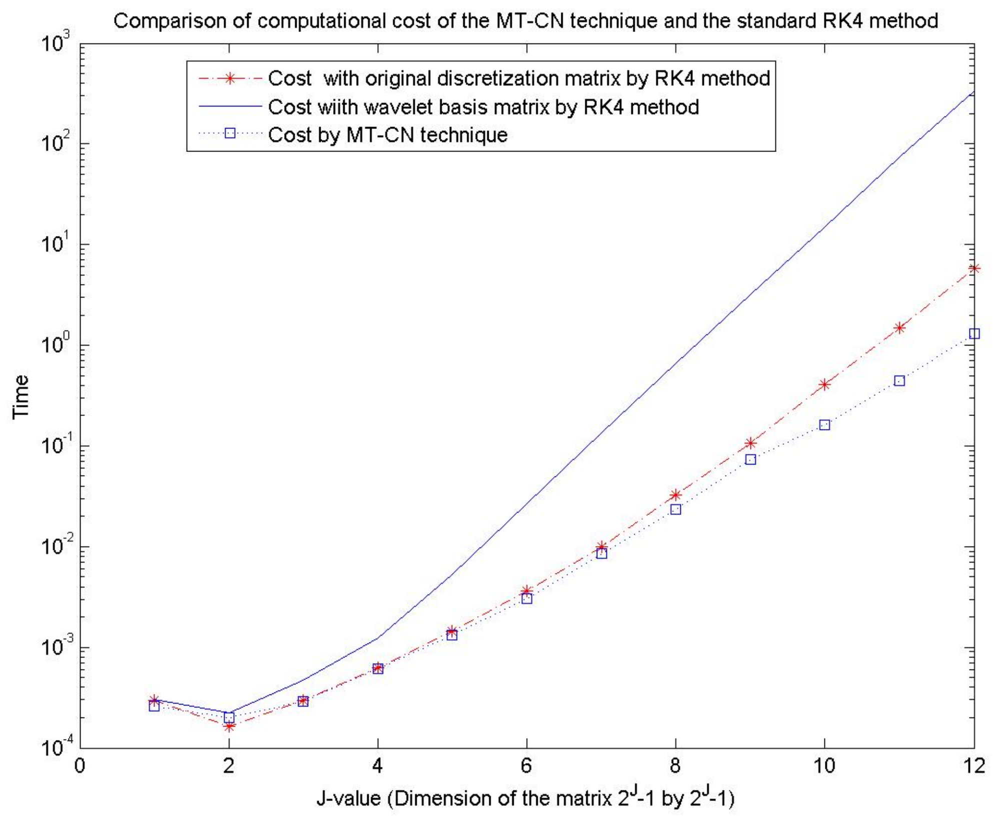


Figure 4.20. Comparison of time evaluation of the MT-CN technique and the RK4 method when solving the advection equation (3.1). The problem is transformed by the Daubechies 4 wavelets.

Now we observe the comparison of the computational cost of the MT-CN technique and the standard CN method. We have already mentioned that this comparison for solving the advection equation is bit messy as the solution of the advection equation is preferably done by explicit methods which are less costly compared to implicit methods. However the experimental result is given below in Figure 4.21.

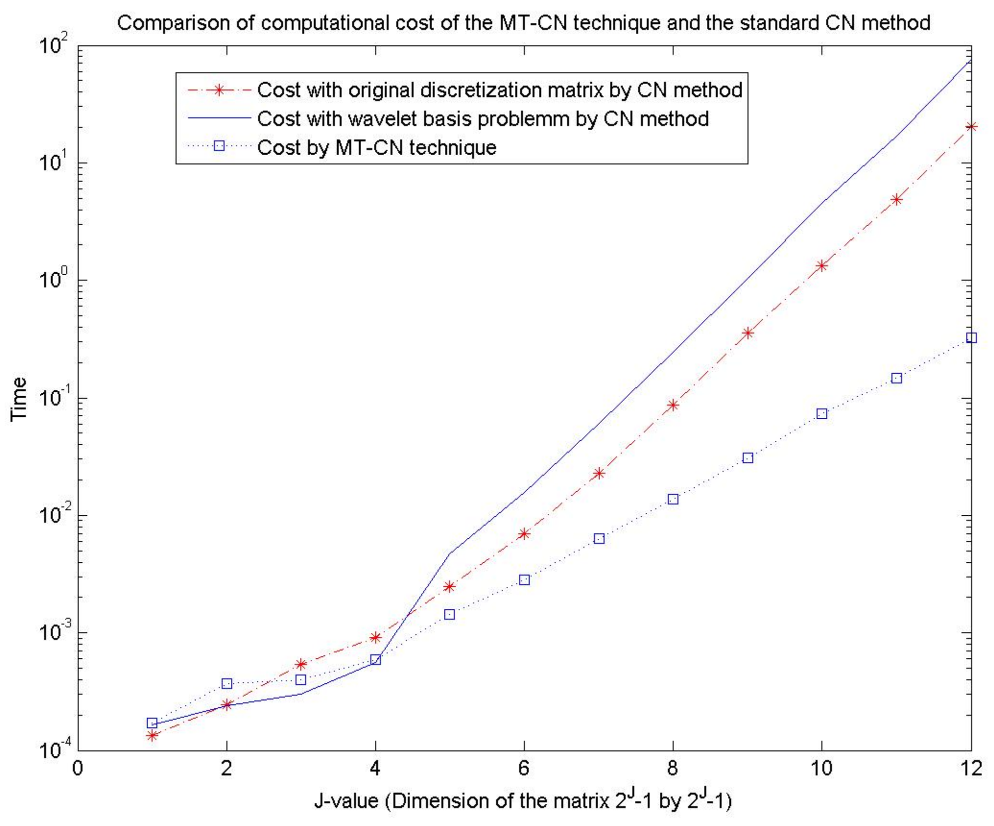


Figure 4.21. Comparison of computational cost for the MT-CN technique and the standard CN method when solving the linear advection equation (3.1). The problem is transformed by the Haar wavelets.

From the Figure 4.21 we can see that for $J > 4$ the cost line of the $MT - CN$ technique departs from other two curves and the distance increases with increasing the problem size.

From all of the three figures showing numerical cost evaluation for the advection equation we can see that the computational cost of the standard method in solving the wavelet basis problem is much higher than the other two cases. It is obvious it should be slower than the original discretization problem because of the sparsity. We also see that the speed is superior for the Multiscale timestepping technique. Now we look at how much performance is gained by our MT-CN technique. In the same way as in the structured case we have calculated the cost ratio. It is calculated as “computational cost by CN method for original discretization matrix" divided by “ computational cost by MT-CN technique". The result is given in Figure 4.22

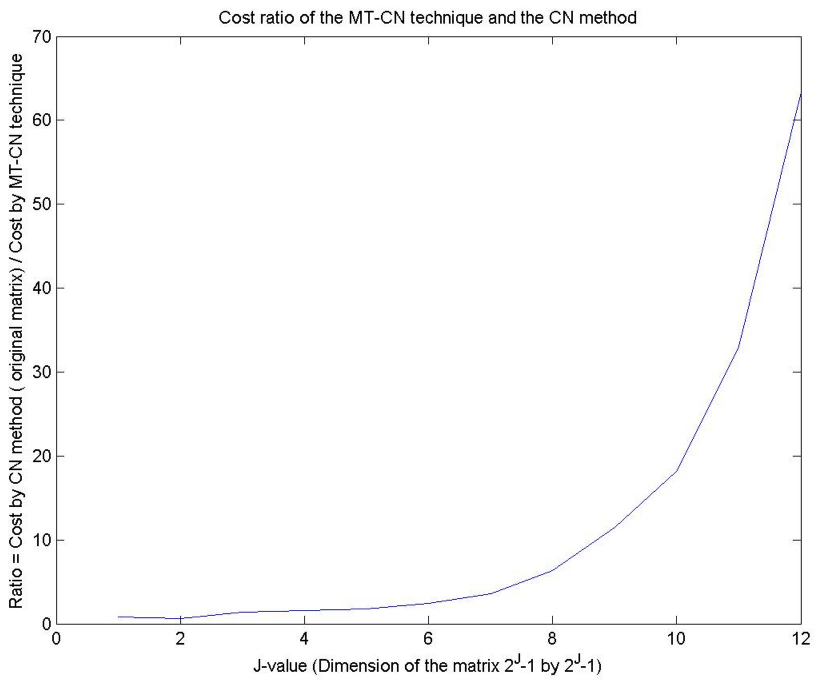


Figure 4.22. Performance achievement by MT-CN technique compared with the standard CN method when solving the linear advection equation (3.1). For the MT-CN technique the problem is transformed by the Haar wavelets.

4.2.2 Advection diffusion equation

In this Section we have implemented our MT technique for the linear advection diffusion equation (3.6). For this problem we have chosen the diffusion coefficient $\epsilon = 0.1$ and as for the linear advection equation we also considered the (almost) periodic initial data $u = e^{-100(x-0.5)^2}$ ($x \in [0,1]$), size of space step $\triangle x = \frac{1}{N}$, time step size $\triangle t = 2 * T_{end}/N$, $T_{end} = 0.5$. Here $N = 2^J - 1$ represent the problem size.

As the diffusion equation is typically solved by implicit methods, here we used the CN method as the standard method and the MT technique is also implemented with the standard CN method. After these considerations, we made numerical experiments in the same way as for the linear advection equation. The results for error reduction for different types of wavelets are given above in Figure 4.23 .

In Figure 4.23 we can see the same characteristics of the solution in error reduction as in the case of the linear advection equation for higher order wavelets. The decay rate of error by the MT-CN technique shows second order accuracy for wavelets with 4 vanishing moments and higher. But for this equation the solution is divergent when we use lower order wavelets. The reason is that for the diffusion equation the elements are larger compared to the advection equation ($P = 2$ in (3.11)). The lower order Haar

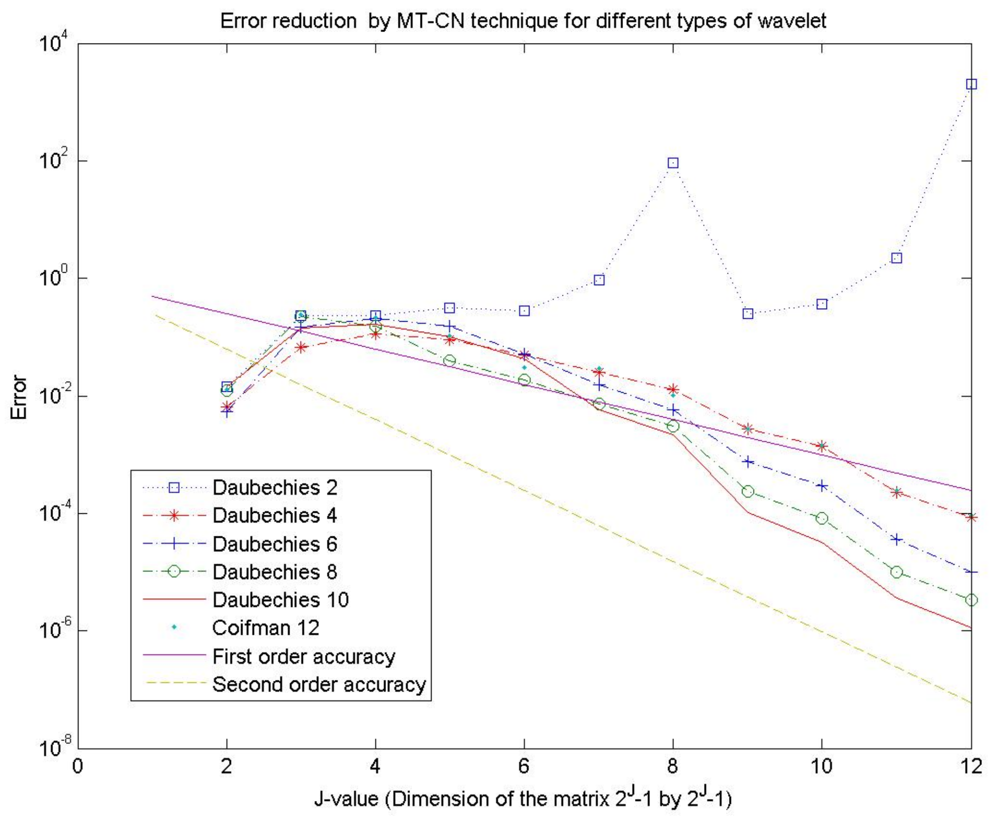


Figure 4.23. Order of accuracy of MT-CN technique for different types of wavelet when solving the linear advection diffusion equation (3.6).

wavelet and Daubechies 2 wavelet transformation do not give sufficient decay rate to the matrix elements (m is too small in (3.11)). But this problem is overcome by higher order wavelet transformations.

Computational cost Figure 4.24 indicates the same properties as in Figure 4.21 for the advection equation. After a certain size of the problem the cost of the $MT - CN$ technique crosses the other cost curves and the distance to them increases for larger problem. Similarly to the cost ratio Figure 4.25 we can also see that for this problem a significant numerical cost reduction is obtained by the MT-CN technique.

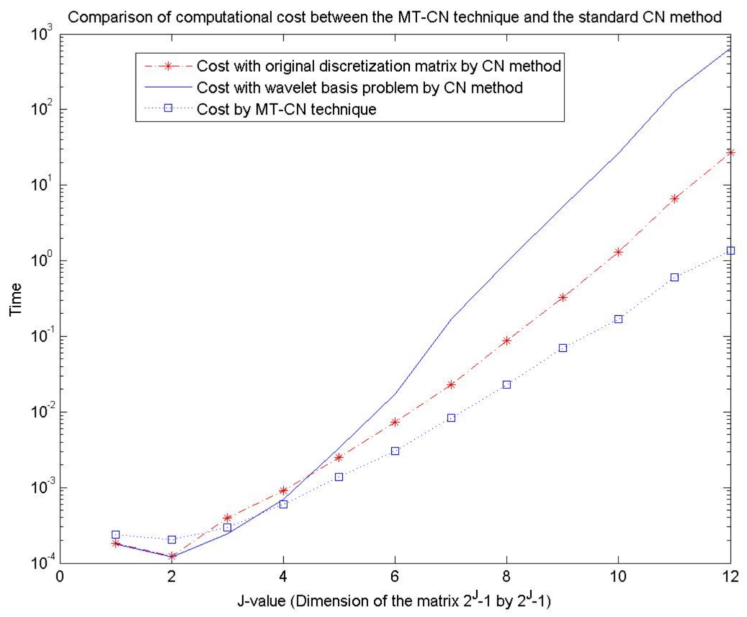


Figure 4.24. Comparison of computational cost MT-CN technique and the CN method when solving the advection diffusion equation (3.6). The problem is transformed by the Daubechies 4 wavelets.

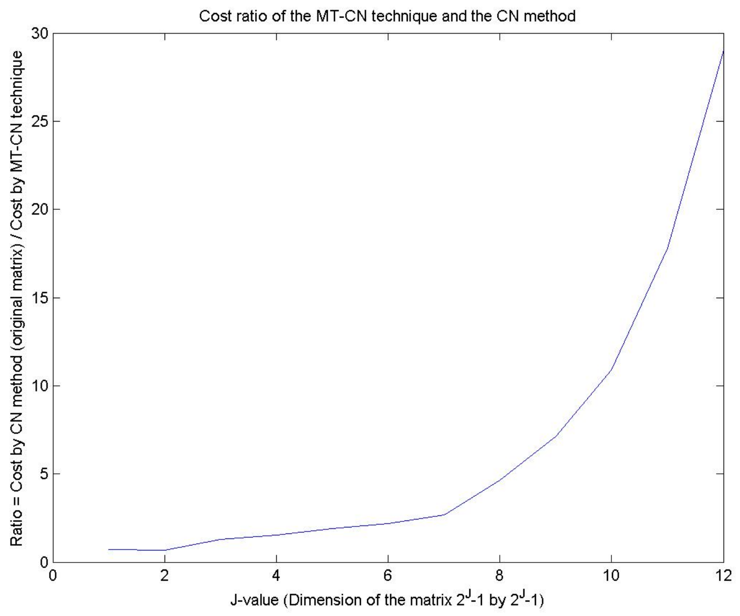


Figure 4.25. Performance achievement by MT-CN technique compared with CN method when solving the advection diffusion equation (3.6). For the MT-CN technique the problem is transformed by the Daubechies 4 wavelets.

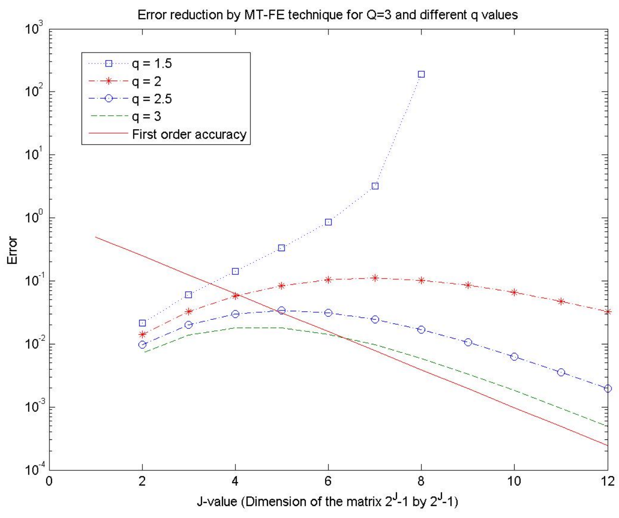
4.3 Non-linear ODEs

In this section we worked with the non-linear structured problem (2.14). The non-linear term $\mathbf{z}\mathbf{z}^T b$ is constructed by the solution vector $\mathbf{z}$ and the unit vector b. The structure of the matrix and the initial vector is same as the structure of the matrix and the solution vector in the blocked version problem in Section 4.1.

Moreover for all of the experiments in this section we implemented our Multiscale timestepping technique only with the FE method by considering $\triangle t = 2^{-J+1}$ and run the simulation until $T_{end} = 1$. After these considerations, the numerical results of error reduction are given in Figure 4.26.

From Figure 4.26 we can see compared to the structured problem experimented in Section 4.1, the $MT-FE$ method for the non-linear structured problem needs bit higher values of (q, Q). In Matlab in each time step we call the function $V * \mathbf{z} + \mathbf{z} * (\mathbf{z}^T * b)$. If we compare with the structured problem in Section 4.1, the extra $\mathbf{z} * (\mathbf{z}^T * b)$ value is added to the solution in each time step. For the standard FE method this value is added to the corresponding elements of the solution vector in each time step, whereas for the $MT-FE$ technique specially for the lower part elements (j large) of the solution vector the value is added very few times.

Moreover the Figure shows that compared to the matrix generator number the vector generator number dominate the solution. If we look the formulation procedures of the non-linear structured problem we can see that to construct the non-linear part we are using the initial vector two times. So it is reasonable that the vector generator number should have more impact in the solution compared to the solution of the structured linear problem.



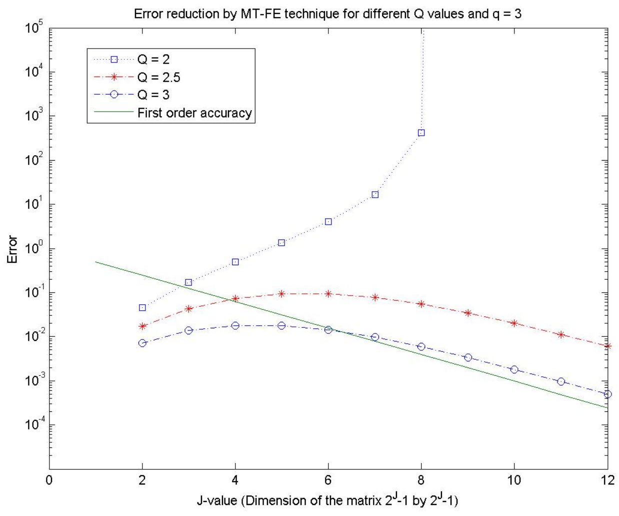


Figure 4.26. Error reduction of MT technique with standard Forward Euler method for different set of (Q, q) values.

Chapter 5

Conclusion

In this thesis we have worked with time upscaling, using a Multiscale timestepping technique. In the solution of PDE and ODE problems, we have studied accuracy of our technique and its computational cost. We have begun our work with some structured ODE problems that follow some decay properties. Then we have moved to well known $PDEs$. After discretization of the $PDEs$ we transformed the matrix and the solution vector into a wavelet basis, which ensured the same properties as the structured $ODEs$ problem. Then we have implemented our Multiscale timesteping technique to the modified problems.

Experimented results of this thesis showed that a significant computational cost can be cut by using Multiscale timestepping technique to $ODEs$ and $PDEs$. We have also seen that this technique often requires implicit numerical methods compared to the explicit time stepping methods.

With lot of advantages shown for the Multiscale timestepping technique, there are also some drawbacks of our work. First of all, when we solved PDE problems by transforming with higher order wavelets this technique is bit slower compared to the problems that transformed with lower order wavelets. Secondly for the non-linear problems our Multiscale timestepping technique did not show the same order of accuracy in error reduction for the same value of the matrix generator number and vector generator number (q, Q) considered in the linear problems. But we hope that these problems will be overcome or at least it will show better performances when this technique will be implemented for larger problems in supercomputers.

Bibliography

[1] Engquist and O. Runborg. *Wavelet-based numerical homogenization with applications.* In T.J. Barth, T.F. Chan and R. Haimes, editors, Multiscale and Multiresolution Methods: Theory and Applications, pages 1-148, 2001.

[2] Johan Walden. *Orthogonal Compactly Supported Wavelets for PDEs.* PhD thesis in Numerical Analysis, Uppsala University, Sweden, 1996.

[3] Olof Runborg. *Fast interface tracking via a multiresolution representation of curves and surface*, Commun. Math. Sci. Vol. 7 No.2, 2009.

[4] Michael B. Giles. *Multilevel Monte–Carlo path simulation*,Oper. Res., 56(3), 607–617, 2008.

[5] B.V. Rathish Kumar and Mani Mehra. *Wavelet based preconditioners for sparse linear systems.* Appl. Math. Comput. 171 (2005) 203–224: 2005.

[6] T. F. Chan, W. P. Tang, and W. L. Wan. *Wavelet Sparse Approximate Inverse Preconditioners*, September 1996.

[7] B. Engquist, S. Osher ,S. Zhong. *Fast wavelet based algorithm for linear evolution equations*, SIAM J. Sci. Comput., 15(4), 755–775, 1994.

[8] L Demanet, L Ying. *Wave atoms and time Upscaling of wave equations* ,2007, Numer. Math. 113-1 (2009) 1-71.

[9] C.C Stolk. *A fast method for linear waves based on geometrical optics*,SIAM J. Numerical Analysis 47(2),1168-1194, 2009.

[10] B. Lastdrager, B. Koren and J. Verwer. *The sparse-grid combination technique applied to time-dependent advection problems*, Appl. Numer. Math., 38(4), 377–401, 2001.

[11] M. Griebel and D. Oeltz. *A sparse grid space-time discretization scheme for parabolic problems*, Computing, 81(1):1-34, 2007.

[12] Jelena Popovic. *Fast Adaptive Numerical Methods for High Frequency Wave and Interface Tracking*, PhD thesis in Numerical Analysis, KTH University, Sweden, TRITA-NA 2012:13, 2012.

[13] L. Ying , E. Candes. *Fast geodesics computation with the phase flow method*, J. Comput. Physics 220(1): 6-18 (2006).

[14] I. Daubechies. *Orthonormal Bases of Compactly Supported Wavelets*,Comm. Pure & Appl. Math., 41 (7), pp. 909-996, 1988.

[15] G. Beylkin, R. Coifman, and V. Rokhlin. *Fast wavelet transforms and numerical algorithms I*, Comm. Pure Appl. Math. Vol. 44, pp. 141-183, 1991.

TRITA-MAT-E 2014:40
ISRN-KTH/MAT/E—14/40-SE