Project 3

Edward Glöckner

Beräkningsvetenskap 2 1TD395

May 30, 2022

UPPSALA UNIVERSITET

Contents

1 Introduction

In this project we study the wave equation: (1) which describes the propagation of waves.

$$u_{tt} = c^2\nabla^2 u \tag{1}$$

We start by analysing the one dimensional case, i.e. wave propagation on a string. By obtaining discrete spatial finite difference approximations of the wave equation, the equation transforms into an initial-value problem (IVP). This IVP can be solved fairly easiliy with the use of numerical approximation methods, more specifically central difference of order 2 (CD2), and runge kutta of order 4 (RK4).

These numerical approximation methods are an incredibly useful tool in an engineers toolbox, since differential equations occur very frequently in engineering purposes. Many differential equations cannot be solved analytically, and one must then turn to these numerical methods.

In this project we focus on the Dirhclet and Neumann boundary conditions. The physical meaning of Dirichlet boundary condition in the one dimensional case, is that the ends of the strings are fixed. Neumann's boundary condition means that the string must come in "straight" to the edge. The physical meaning of this boundary condition applied on a one dimensional string, can be interpreted as the string having no mass. After analysing numerical stability and convergence rate of CD2 and RK4, we then solve the two dimensional case. To do this we discretise the wave equation with the help of CD2.

2 Initial-boundary value problems in 1D

We start with the one dimensional (1D) case, and restrict the computational domain to -1 $\leq x \leq$ 1. Here $u = u(x,t)$. Since the system is on second order form (second derivative in time) two initial conditions are needed: $u = f(x)$, $u_t = p(x)$, -1 $\leq x \leq$ 1, where $f(x)$ and $p(x)$ are initial data. In the present study $p(x) = 0$, i.e., the system starts from rest, which is most often the case in applications. In Figure 1 we plot an initial Gaussian profile $f(x) = exp((5x)^2)$, that will be later used in the computations.

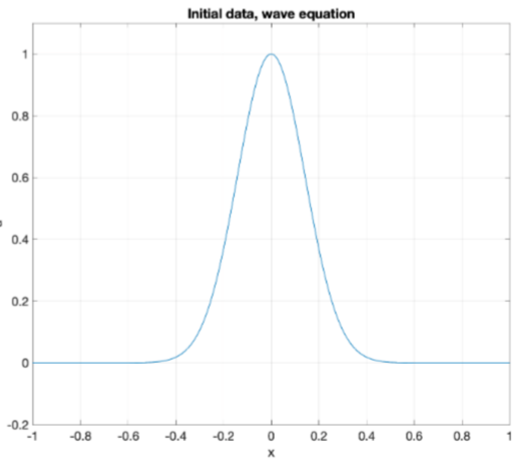


Figure 1: Initial data u(x,0) = f(x) used in the computations

A well-posed IBVP requires one BC at the left and right boundaries (two in total), denoted $L^{(l)}u = g^{(l)}(t)$ and $L^{(r)}u = g^{(r)}(t)$, respectively. There are many different types of well-posed BC for the 1D wave equation. In the present study we will focus on two different well-posed BC:

$$\begin{cases} L^{(l,r)}u = u = g^{(l,r)}(t), & \text{Dirichlet BC} \\ L^{(l,r)}u = u_x = g^{(l,r)}(t), & \text{Neumann BC} \end{cases} \tag{2}$$

where $g^{(l,r)}(t)$ are boundary data (in this study we set $g^{(l,r)}(t) = 0$). The full problem given by,

$$\begin{cases} u_{tt} = c^2 u_{xx}, & -1 \leq x \leq 1, t \geq 0 \\ L^{(l)}u = 0, & x = -1, t \geq 0 \\ L^{(r)}u = 0, & x = 1, t \geq 0 \\ u = f(x), u_t = 0, & -1 \leq x \leq 1, t = 0 \end{cases} \tag{3}$$

is referred to as an IBVP. One example of a well-posed IBVP is to specify homogeneous Neumann BC ($u_x = 0$) at both boundaries. It can be shown that the eigenvalues (here denoted $-\lambda^2$) to the spatial operator, which corresponds to $c^2 \frac{\partial^2}{\partial x^2}$ combined with Neumann BC is given by $-\lambda^2 = -(\frac{cn\pi}{2})^2$, n = 0,1,... (i.e., non-positive and real). This means that the eigenvalues to (3) are given by $\pm\sqrt{-\lambda^2} = \pm\frac{icn\pi}{2}$, n = 0,1,.... Here i denotes the imaginary unit. Hence the eigenvalues to (9) reside on the imaginary axis in the complex plane, including zero (which is a double root).

Another example of a well-posed IBVP is with homogeneous Dirichlet BC (u = 0) at both boundaries. It can be shown that the eigenvalues to the spatial operator now is given by $-\lambda^2 = -(\frac{cn\pi}{L})^2$, n = 1,2,.... This means that the eigenvalues to (3) now are given by $\pm\frac{icn\pi}{2}$, n = 1,2,.... Notice that the zero double root is not included here.

3 Analytic solutions

The following Gaussian profiles,

$$\theta^{(1)}(x,t) = e^{(-(\frac{x-ct}{r_*})^2)}, \theta^{(2)}(x,t) = -e^{(-(\frac{x+ct}{r_*})^2)} \tag{4}$$

are introduced where r_* defines the width of the Gaussian.

Let $r_* = 0.2$, where the domain is restricted to -1 $\leq x \leq$ 1. If initial data are set to:

$$u(x,0) = \theta^{(1)}(x,0), u_t(x,0) = 0 \tag{5}$$

analytic solutions to (3) using either Neumann or Dirichlet BC are given by,

$$\begin{cases} u^{(1)}(x,t) = +\frac{1}{2}\theta^{(1)}(x+2,t) - \frac{1}{2}\theta^{(2)}(x-2,t), & \text{Neumann BC} \\ u^{(2)}(x,t) = -\frac{1}{2}\theta^{(1)}(x+2,t) + \frac{1}{2}\theta^{(2)}(x-2,t), & \text{Dirichlet BC} \end{cases} . \tag{6}$$

after the Gaussian pulses have been reflected at the boundaries, i.e., approximately when $ct \in [1.75, 2.25]$. (When doing a numerical convergence study, avoid using end time excactly T = 2.)

4 The resulting IVP

The domain (-1 $\leq x \leq$ 1) is discretized using the following m equidistant grid points (with grid-spacing h):

$$x_j = -1 + (j-1)h, j = 1, 2, ..., m, h = \frac{2}{m-1} \tag{7}$$

The approximate solution at grid point x_j is denoted v_j, and the discrete solution vector $v = [v_1, v_2, ..., v_m]^T$ (where superscript T denotes transpose, i.e., v is a column vector of length m).

Stable fourth order accurate (semi-discrete) spatial finite difference approximations of (3) are constructed from the given MATLAB function FSBP4. To use the function you need to prescribe the number of gridpoints m, and grid-spacing h. The call [BD, BN]=FSBP4(101) returns the discretisation matrices (excluding c^2) for Neumann BC (here denoted BN) and Dirichlet BC (here denoted BD) using m = 101 grid-points. (If $c^2 \neq 1$, simply multiply the matrices by c^2).

A semi-discrete finite difference approximation of (3) results in the following IVP (with m unknowns),

$$\begin{cases} v_{tt} = c^2 B v, & t \geq 0 \\ v = f, v_t = 0, & t = 0 \end{cases} \tag{8}$$

Here B is a *mxm* discretisation matrix, returned by the MATLAB function FSBP4 (i.e., B denotes either BN or BD). Denote the eigenvalues to c^2B by -λ_j^2, j = 1,2,...,m. The eigenvalues to (8) is givenby $\pm i\lambda_j$, j = 1,2,...,m. This coupling between the eigenvalues to c^2B and (8) is more clearly seen by first rewriting the IVP to first order form. Let w = vt, then (8) can be rewritten

$$\begin{cases} \begin{bmatrix} v \\ w \end{bmatrix}_t = \begin{bmatrix} 0 & 1 \\ c^2B & 0 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} & t \geq 0 \\ \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix} & t = 0 \end{cases} . \tag{9}$$

where I is the *mxm* unit matrix (eye(m) in Matlab). Hence, the eigen-values to (13) equals the eigenvalues to the 2m x 2m matrix A given by

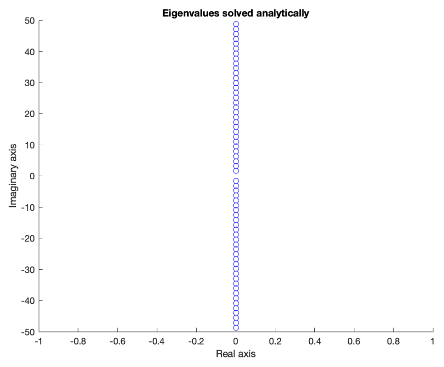
$$A = \begin{bmatrix} 0 & I \\ c^2B & 0 \end{bmatrix} \tag{10}$$

5 Assignment 1

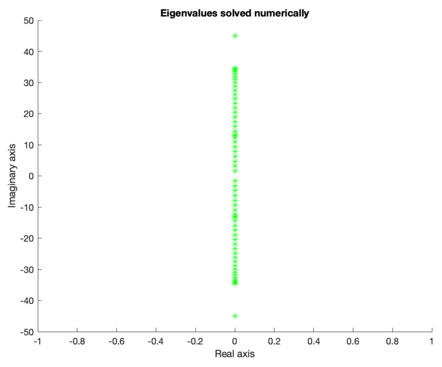
Plot the eigenvalues to (8), in the complex plane, for the case with Dirichlet BC, when c = 1 and m = 31. (The eigenvalues corresponds to the eigenvalues to the matrix A given by (10).) Compare the result against the analytic eigenvalues given by $\lambda = \pm\frac{icn\pi}{2}$ (for n =1,2,...,31). Present a plot of the eigenvalues (both analytic and numerical). Verify for example that the smallest (in magnitude) discrete eigenvalues agree with the corresponding continuous eigenvalues ($\pm\frac{ic\pi}{2}$).

5.1 Solution Assignment 1

We denote the eigenvalues to c^2B by λ_j^2, then the eigenvalues to (8) are easily calculated by finding $\pm i\lambda_j$. The analytical solution to the eigenvalues are given by: $\lambda = \pm\frac{icn\pi}{2}$, and figure 2 illustrates the results.



(a)



(b)

Figure 2: Plot of the eigenvalues solved (a) analytically (b) numerically

Table 1 shows the error between the numerical, and the analytical solution of the eigenvalues, for the first five eigenvalues. We can see that the smallest eigenvalues agree relatively well with the analytical eigenvalues.

Error of the eigenvalues	
Positive imaginary part	Negative imaginary part
5.57e-07	5.57e-07
2.09e-05	2.09e-05
0.000199	0.000199
0.001086	0.001086
0.004408	0.004408

Table 1: Error of the numerically solved eigenvalues

6 Assignment 2

The two largest (imaginary) eigenvalues in (8) are given by $\pm i\frac{c}{h}\gamma$ for some constant $\gamma > 0$ that depends on the type of BC. This imply that numerical stability when employing RK4 is given by $k \leq \frac{2.8}{c\gamma}h$. The constant γ can be estimated by computing the eigenvalues to (8) with c = 1 using for example m = 31 grid points. Present estimates of γ for Neumann and Dirichlet BC. With this information, motivate why Euler forward (RK1) or Heun (RK2) are unstable methods for this IVP.

6.1 Solution Assignment 2

Since the eigenvalues are purely imaginary we can write:

$$\lambda_{max} = \frac{c}{h}\gamma \Leftrightarrow \gamma = \frac{2\lambda_{max}}{(1-m)c} \tag{11}$$

With this information we can program a MATLAB script which solves the equation, and outputs:

```
Biggest gamma for Dirichlet using c = 1 and m = 31 is: 3.0009
Biggest gamma for Neumann using c = 1 and m = 31 is: 2.3053 »
```

Figure 3 illustrates the stability domain for some runge-kutta methods.

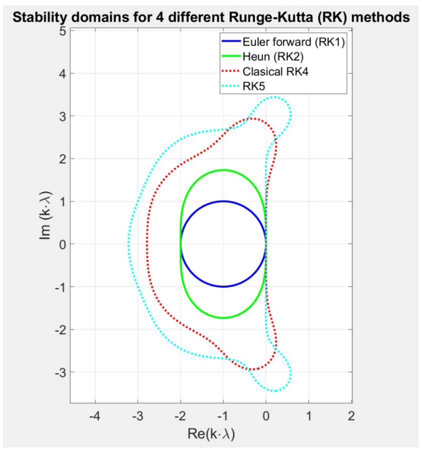


Figure 3: Stability domain for some numerical methods

We can see from figure 3 that Heuns method (RK2) is unstable for purely imaginary eigenvalues, and therefor this method does not work for our IVP. The stability domain for Euler forward (RK1) is given by: $Q(k\lambda) = |1 + k\lambda| \leq 1$, and we can see that this method is not stable for an ODE with at least 1 purely imaginary eigenvalue. Since our ODE only have imaginary eigenvalues, this method also does not work for our IVP.

7 Time discretization of the resulting IVP

Let $t_n = (n-1)k$, n = 1,2,... denote the discrete time-levels, where k is the time-step, and introduce the notation $v^n = v(t_n)$. Now it is time to solve (8) using RK4 and CD2. The RK4 method is explicit and requires that the ODE system (8) is first rewritten onto first order (in time) form (14). The symplectic CDp methods solves the problem directly on second order form.

8 Assignment 3

Write a MATLAB program that solves (8) for both types of BC, using RK4 for time-integration. It should be easy to change number of grid-points (m) time-step (k), end time (T), initial data, and type of BC. Verify by numerical simulations that $k_* = \frac{2.8}{c\gamma}$ is the largest possible time-step for numerical stability (use result from Assignment 2), by performing long-time simulations (until T = 10000). Plot the solution at t = 0.2, 0.5, 0.7, 1.8 using m = 101 and k = 0.01 for Dirichlet and Neumann BC. Initialise the solutions using (5). Hint: It can be instructive to simulate the solution. To generate a movie the function VideoWriter is useful.

8.1 Solution Assignment 3

In order to solve (8) we first need to rewrite it into first order form. We use the variable substitution $v_t = w \Leftrightarrow v_{tt} = w_t$, which will give us the following equation system.

$$\begin{cases} v_t = w, & t \geq 0 \\ w_t = c^2 B v, & t \geq 0 \\ v = f, w = 0, & t = 0 \end{cases} \tag{12}$$

Now with the use of linear algebra we can simply this equation system into

$$\begin{bmatrix} v \\ w \end{bmatrix}_t = \begin{bmatrix} 0 & I \\ c^2 B & 0 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{13}$$

where I is the unit m x m - matrix. Our initial values can be written as:

$$\begin{bmatrix} v \\ w \end{bmatrix}_{t=0} = \begin{bmatrix} f \\ 0 \end{bmatrix} \tag{14}$$

Using the Gaussian profiles we can write our initial values as:

$$\begin{bmatrix} f \\ 0 \end{bmatrix} = \begin{bmatrix} \theta^{(1)}(x,0) \\ 0 \end{bmatrix} = \begin{bmatrix} e^{-(\frac{x}{r_*})^2} \\ 0 \end{bmatrix} \tag{15}$$

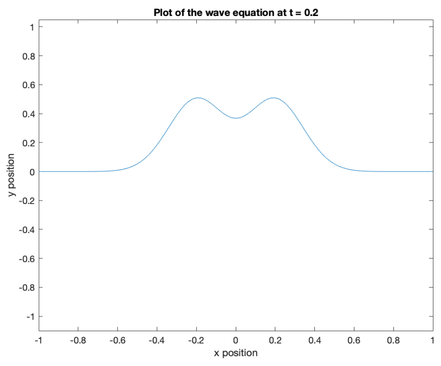
where r_* is 0.2. We denote:

$$u = \begin{bmatrix} v \\ w \end{bmatrix}, A = \begin{bmatrix} 0 & I \\ c^2 B & 0 \end{bmatrix}$$

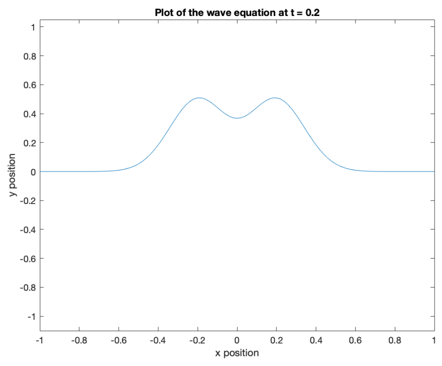
which give us the following form:

$$u_t = Au \tag{16}$$

This is then solved in MATLAB, and in figure 4, 5, 6 and 7 you can see some solutions using $m = 101$ and $c = 1$. The function "RK4.m" is used to solve the equation, and functions "animation.m" and "plotsnapshot.m" are used to illustrate the results.

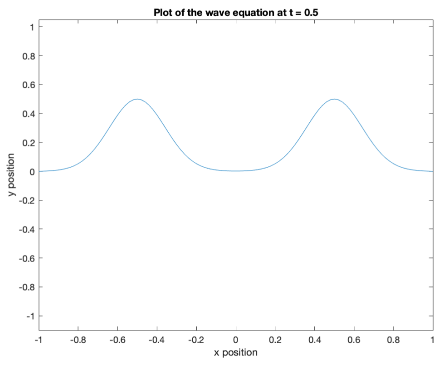


(a)

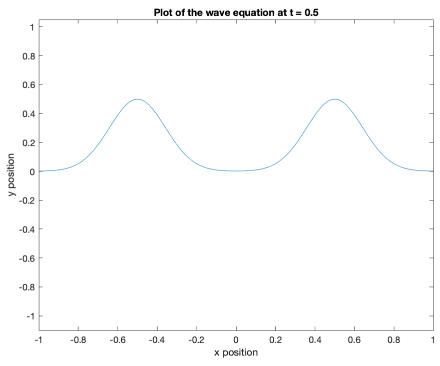


(b)

Figure 4: Plot of the wave equation at t=0.2 s using boundary condition (a) Dirichlet (b) Neumann

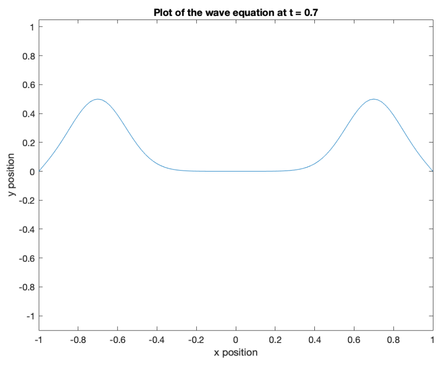


(a)

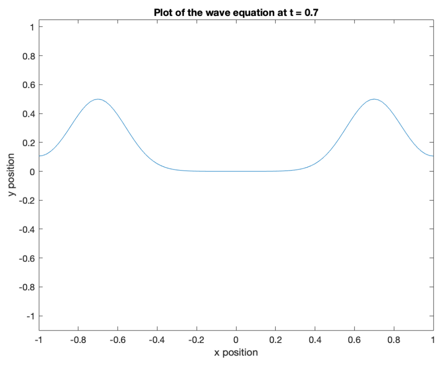


(b)

Figure 5: Plot of the wave equation at t=0.5 s using boundary condition (a) Dirichlet (b) Neumann

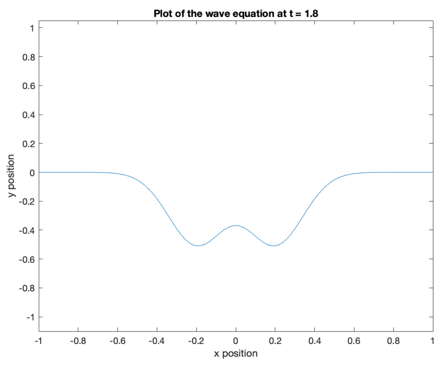


(a)

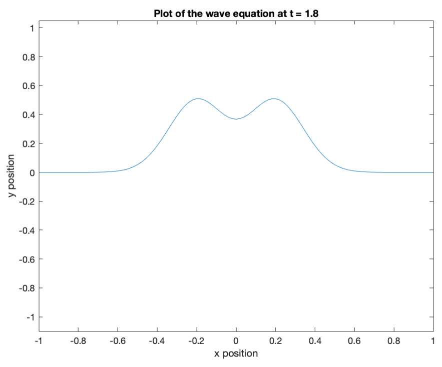


(b)

Figure 6: Plot of the wave equation at t=0.7 s using boundary condition (a) Dirichlet (b) Neumann



(a)

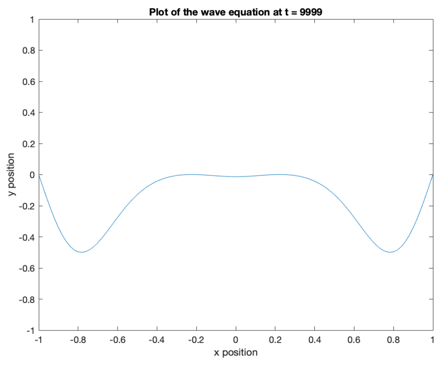


(b)

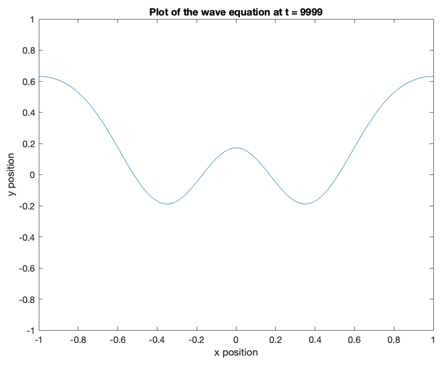
Figure 7: Plot of the wave equation at t=1.8 s using boundary condition (a) Dirichlet (b) Neumann

Note that for the plots using the Dirichlet boundary condition, the string is fixed at the ends. For the Neumann boundary condition the derivative is always zero at the ends, which mean that the string comes in "straight" to the ends.

Now we verify that the largest possible time-step for numerical stability is given by $k_* = \frac{2.8}{c\gamma}h$. From assignment 2 we get that $\gamma_D = 3.0009$ and $\gamma_N = 2.3053$. Using $c = 1$, and $m = 101$ we get k_* for Dirichlet boundary condition is 0.0187, and k_* for Neumann boundary condition is 0.0243. Figure 8 illustrates the wave equation at $t = 9999$ using k_*.



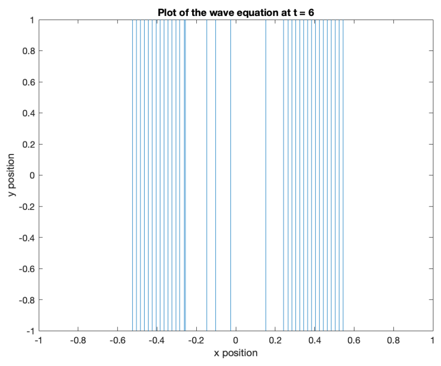
(a)



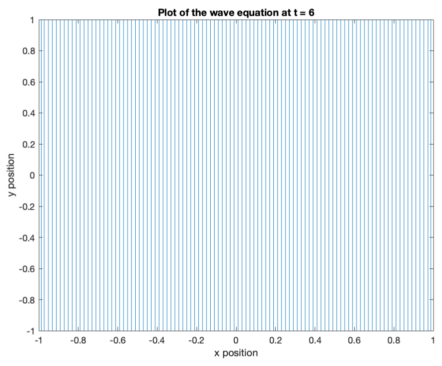
(b)

Figure 8: Plot of the wave equation at t=9999 s at k_* using boundary condition (a) Dirichlet (b) Neumann

We can see that these simulations are stable. Now we use $k_* + 0.001$ as our time step and try to plot the wave equation. Figure 9 illustrates the results.



(a)



(b)

Figure 9: Plot of the wave equation at t=6 s at $k_* + 0.001$ using boundary condition (a) Dirichlet (b) Neumann

We can see that these equations are unstable already at $t = 6$. The reason i can't show a plot at a later time is that the solution diverges to infinity so fast. Hence we have shown that $k_* = \frac{2.8}{c\gamma}h$ is indeed true.

9 Assignment 4

Do a convergence study for the cases with Dirichlet and Neumann BC, based on the analytic solutions given by (6). Initialise the solution using (5). Use for example m = 51, m = 101, m = 201 and m = 301 grid-points. Use k = 0.1 h for the time step to make sure the solution is stable. Measure the error at T = 1.8. Present a table with the discrete l_2 norm of the error and convergence rate q. Do you obtain the expected convergence behaviour?

The convergence rate is calculated as

$$q = \frac{log_{10}(\frac{||u-v^{(m_1)}||_h}{||u-v^{(m_2)}||_h})}{log_{10}(\frac{m_2}{m_1})} \tag{17}$$

where u is the analytic solution, and $v^{(m1)}$ the corresponding numerical solution with m_1 unknowns and

$$||u - v^{(m_1)}||_h \equiv \frac{1}{m_1}\sqrt{\sum_{j=1}^{m_1}(u(x_j) - v_j)^2} \tag{18}$$

is the discrete l_2 norm of the error. In (17) we measure the spatial convergence (related to the SBP operator in this case) and this means that we should use a discrete norm since the length of the solution vector scales linearly with the number of grid-points m. In Matlab we can use $sqrt(h) * norm(u - v)$ to compute the discrete l_2 norm of the error, where v is the numerical solution vector (at a certain time-level) and u the corresponding analytic solution (also a vector of length m). (Note that $h = \frac{2}{m-1}$, which is why $h \approx \frac{1}{m}$).

9.1 Solution Assignment 4

To calculate the convergence rate we calculate the l2-norm using two different grid points, and then by using equation (17). Table 2 shows the errors at T=1.8. To find these errors we simply sum all errors at the discrete x-values, at the final time point.

Error study at time step T = 1.8		
Grid point m_1	Error Dirichlet	Error Neumann
51	0.0037	1.8125e-06
101	2.4971e-07	2.2010e-07
201	3.3814e-07	3.3386e-07
301	4.7003e-07	4.6815e-07
501	7.4110e-07	7.3952e-07

Table 2: Table of the grid points and errors for Dirichlet and Neumann boundary condition, at T = 1.8

The discrete l2 norm of the error and the convergence rate for Dirichlet boundary condition is shown in table 3 and for Neumann boundary condition is shown in table 4.

Convergence study for RK4, with final time-step: T = 1.8				
Grid point m_1	Grid point m_2	l2 norm for m_1	l2 norm for m_2	Convergence rate
51	101	0.00152672	2.8763e-05	5.8126
101	201	2.8763e-05	1.7619e-06	4.0580
201	301	1.7618e-06	3.5534e-07	3.9649
301	501	3.5537e-07	4.7501e-08	3.9496

Table 3: Table of the grid points, l2 norms, and convergence rate for Dirichlet boundary condition

Convergence study for RK4, with final time-step: T = 1.8				
Grid point m_1	Grid point m_2	l2 norm for m_1	l2 norm for m_2	Convergence rate
51	101	0.00045226	3.4390e-05	3.7704
101	201	3.4396e-05	2.1004e-06	4.0626
201	301	2.1003e-06	4.0409e-07	4.0819
301	501	4.0404e-07	5.1473e-08	4.0460

Table 4: Table of the grid points, l2 norms, and convergence rate for Neumann boundary condition

We can see that as the grid points increases the convergence rate converges to 4, which is what we expect since runge-kutta 4 is fourth order accurate.

10 Assignment 5

Show that a second order symplectic (CD2) approximation of (8) including the initial conditions can be written,

$$\begin{cases} \frac{v^{n+1}-2v^n+v^{n-1}}{k^2} = c^2Bv^n, & n = 1,2,3,... \\ v^1 = f, v^2 = (I + \frac{k^2}{2}c^2B)f \end{cases} \tag{19}$$

Present also the corresponding (fourth order accurate) CD4 approximation. This is similar to the analysis in Problem 4. (Simply replace b with c^2B and y_n with v_n.)

10.1 Solution Assignment 5

We can write our system as a finite difference approximation.

$$\begin{cases} v_{tt} \approx \frac{v^{n+1}-2v^n+v^{n-1}}{k^2} = c^2Bv^n, & t > 0 \\ v^1 = f, v_t = \frac{y^2-y^1}{k} = 0, & t = 0 \end{cases} \tag{20}$$

Equation (20) is already on CD2 form, except that our initial condition is still only first order accurate. To increase the accuracy to the second order we can use a Taylor expansion of the initial condition. A taylor expansion of $v(t_n + k)$ can be written as:

$$v(t_n + k) = v(t_n) + kv_t(t_n) + \frac{k^2}{2}v_{tt}(t_n) + \frac{k^3}{6}v_{ttt}(t_n) + \frac{k^4}{24}v_{(4)}(t_n) + ... \tag{21}$$

Putting equation (21) into (20) with the following initial conditions the result becomes second order accurate as in equation (22).

$$\frac{v^2 - v^1}{k} = \frac{v(0+k) - v(0)}{k} = \frac{k}{2} v_{tt}(0) + O(k^2) \tag{22}$$

Using $v_{tt}(0) = c^2 B v(0)$ we get:

$$\frac{v^2 - v^1}{k} - \frac{k}{2} c^2 B v^1 = 0 \Leftrightarrow v^2 = \frac{k^2}{2} c^2 B v^1 + v^1 \tag{23}$$

Which we can rewrite into:

$$v^2 = (I + \frac{k^2}{2} c^2 B) f \tag{24}$$

Thus resulting in (19).

To expand this into CD4, we have to taylor expand $v(t_n + k)$ and $v(t_n - k)$ around t_n and put it into equation (25):

$$\frac{v(t_n + k) - 2v(t_n) + v(t_n - k)}{k^2} - v_{tt}(t_n) = N(k) \tag{25}$$

Taylor expansion of $v(t_n - k)$ can be written as in equation (26).

$$v(t_n - k) = v(t_n) - k v_t(t_n) + \frac{k^2}{2} v_{tt}(t_n) - \frac{k^3}{6} v_{ttt}(t_n) + \frac{k^4}{24} v_{(4)}(t_n) + \ldots \tag{26}$$

Now put the Taylor expansions from equations (21) and (26) into equation (25) and the accuracy becomes as follows.

$$\begin{aligned} N(k) &= \frac{k^2}{12} v_{tttt}(t_n) + O(k^4) \\ &= \frac{k^2}{12} c^4 B^2 v(t_n) + O(k^4) \end{aligned} \tag{27}$$

Put equation (27) in equation (25) which result in the next equation, CD4, and is fourth-order accurate.

$$\begin{aligned} \frac{v^{n+1} - 2v^n + v^{n-1}}{k^2} &= c^2 B v^n + \frac{k^2}{12} c^4 B^2 v^n \\ &= c^2 B (1 + \frac{k^2}{12} c^2 B) v^n \end{aligned}$$

We also have to improve the accuracy of the initial condition to the fourth-order which we can do with more Taylor expansion. Equation (28) is the initial condition of the first derivative of fourth order accuracy.

$$\frac{v^2 - v^1}{k} - \frac{k}{2}c^2 B v^1 - \frac{k^3}{24}c^4 B^2(v^1) = 0 \tag{28}$$

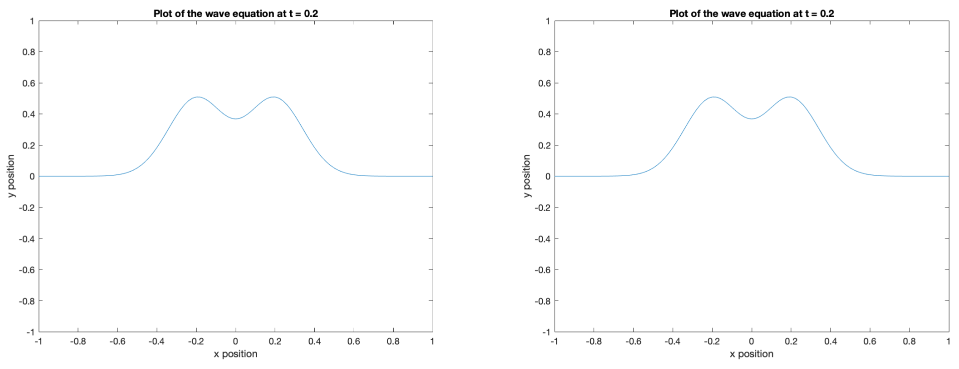
11 Assignment 6

Write a MATLAB program that solves (8) using CD2. Hence, the BC is either Neumann or Dirichlet. It should be easy to change number of grid-points (m) time-step (k), end time (T), initial data, and type of BC. It can be shown that $k_* = \frac{\sigma}{c}h$ for some constant $\sigma > 0$ that depend on the order of the type of BC.

Try to predict the time-step limit k_* (largest possible time-step for stability) by pinpointing σ. This is done by numerical long-time simulations (to verify stability). This limit is not the same as for RK4 (referred to as γ). Present σ for CD2 for Neumann and Dirichlet BC. Initialize the solutions using (5).

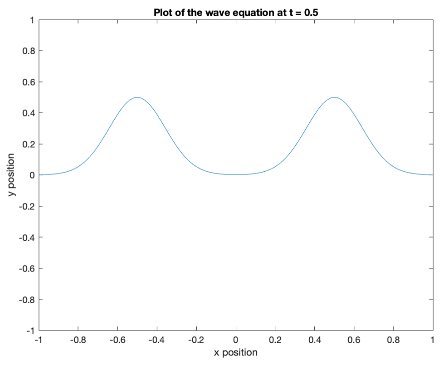
11.1 Solution Assignment 6

We use equation (19) and solve this in MATLAB, and in figure 10, 11, 12 and 15 you can see some solutions using $m = 101$ and $c = 1$.

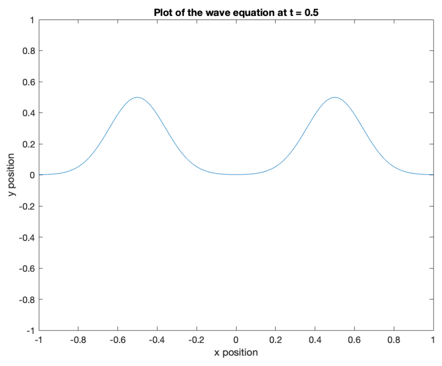


(a) (b)

Figure 10: Plot of the wave equation at t=0.2 s using boundary condition (a) Dirichlet (b) Neumann

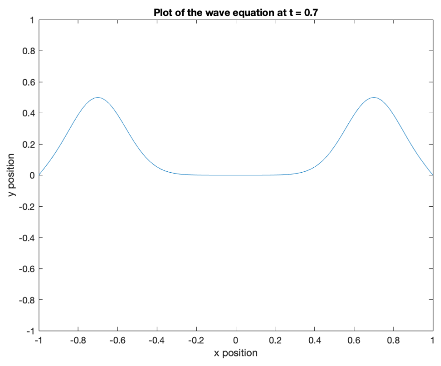


(a)

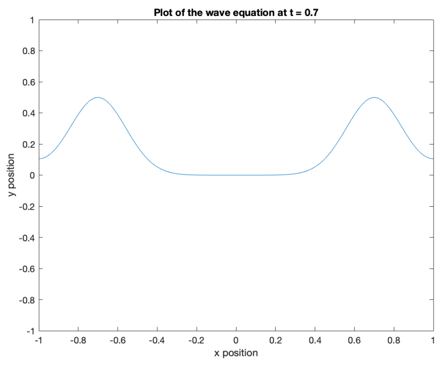


(b)

Figure 11: Plot of the wave equation at t=0.5 s using boundary condition (a) Dirichlet (b) Neumann

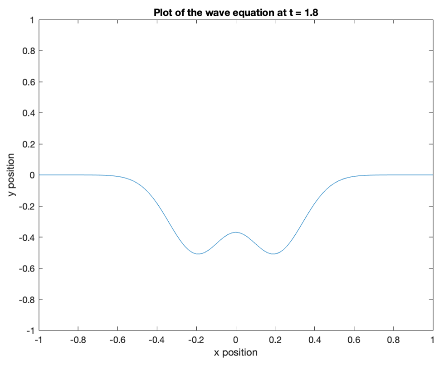


(a)

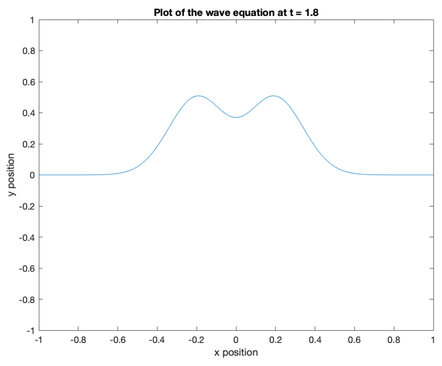


(b)

Figure 12: Plot of the wave equation at t=0.7 s using boundary condition (a) Dirichlet (b) Neumann



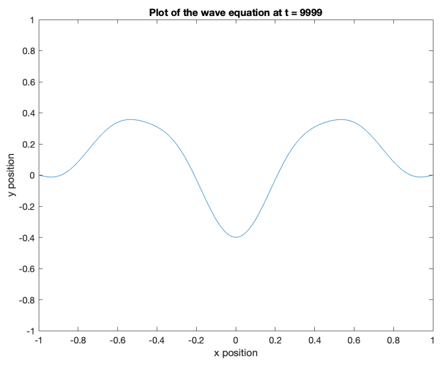
(a)



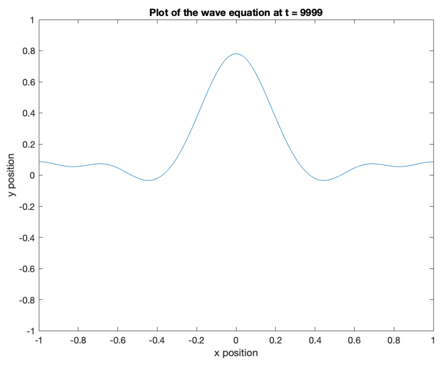
(b)

Figure 13: Plot of the wave equation at t=1.8 s using boundary condition (a) Dirichlet (b) Neumann

By performing numerical long time simulation I was able to pinpoint $k_* = 0.01332$ for Dirichlet boundary condition and $k_* = 0.01732$ for Neumann boundary condition . Figure 14 illustrates the wave equations at $t = 9999$ using k_*.



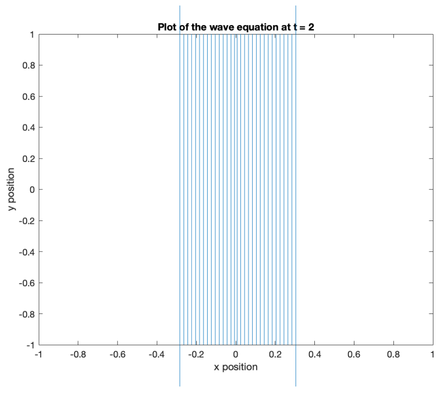
(a)



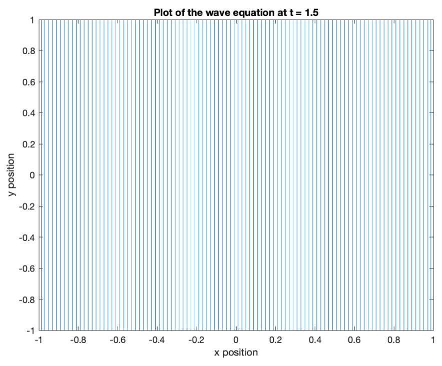
(b)

Figure 14: Plot of the wave equation at t=9999 s at k_* using boundary condition (a) Dirichlet (b) Neumann

We can see that these simulations are stable. Now we use $k_* + 0.001$ as our time step, and try to plot the wave equations.



(a)



(b)

Figure 15: Plot of the wave equation at $k_* + 0.001$ using boundary condition (a) Dirichlet (b) Neumann

We can easily see that these equations are unstable already at t = x. The reason i can't show a plot of a later time is that the solution diverges to infinity so fast.

With this information we calculate σ for Dirichlet boundary condition to be 0.8660, and 0.6660 for Neumann boundary condition.

12 Assignment 7

Verify the CD2 implementation by performing a convergence study for the cases with Dirichlet and Neumann BC, based on the analytic solutions given by (6). Initialize the solution using (5). Use for example m = 51, m = 101, m = 201 and m = 301 grid-points. Measure the error at T = 1.8. Present a table with the discrete l_2 norm of the error and convergence rate q. Do you obtain the expected convergence behaviour? Discuss pros and cons when comparing RK4 and CD2 for this particular problem. Which of the 3 methods (RK4, CD2 or CD4) do you believe is most efficient for this type of problems (wave equations)?

12.1 Solution Assignment 7

To calculate the convergence rate we calculate the l2-norm using two different grid points as in assignment 4. Table 5 shows the errors at T=1.8. To find these errors we simply sum all errors at the discrete x-values, at the final time point.

Error study at time step T = 1.8		
Grid point m_1	Error Dirichlet	Error Neumann
51	0.0039	1.8413e-06
101	2.5563e-07	2.2220e-07
201	3.4078e-07	3.3433e-07
301	4.7174e-07	4.6835e-07
501	7.4200e-07	7.3825e-07

Table 5: Table of the grid points and errors for Dirichlet and Neumann boundary condition, at T = 1.8

The discrete l2 norm of the error and the convergence rate for Dirichlet boundary condition is shown in table 6, and for Neumann boundary condition is shown in table 7.

Convergence study for CD2, with final time-step: T = 1.8				
Grid point m_1	Grid point m_2	l2 norm for m_1	l2 norm for m_2	Convergence rate
51	101	0.00147387	2.2398e-05	6.1318
101	201	2.2327e-05	7.2841e-06	1.6276
201	301	7.2841e-06	3.5508e-06	1.7793
301	501	3.5508e-06	1.3416e-06	1.9103

Table 6: Table of the grid points, l2 norms, and convergence rate for Dirichlet boundary condition

Convergence study for CD2, with final time-step: T = 1.8				
Grid point m_1	Grid point m_2	l2 norm for m_1	l2 norm for m_2	Convergence rate
51	101	0.00033388	2.0426e-05	4.0889
101	201	2.0426e-05	7.0323e-06	1.5494
201	301	7.0323e-06	3.5130e-06	1.7187
301	501	3.5130e-06	1.3385e-06	1.8938

Table 7: Table of the grid points, l2 norms, and convergence rate for Dirichlet boundary condition

We can see that the convergence rate using 51 grid points is not very accurate, but when the grid points increases the convergence rate converges to 2, which is what we expect since CD2 is second order accurate.

One obvious reason to use RK4 and CD4 above CD2, is that they are fourth order accurate. When it comes to RK4, it is however relatively expensive when it comes to computational operations, compared to CD methods, simply because RK4 require more function evaluations, and therefor have a bigger time complexity. CD4 offers the same accuracy as RK4, but with the same time complexity as CD2. Thus CD4 is as accurate as RK4 but require 4 times less computational operations.

We can see that CD2 require a smaller time step for numerical stability for both Dirichlet and Neumann boundary conditions compared to RK4. One other very important note is that CD4 is symplectic which is great for Hamiltonian system such as the one studied in this project, whilst RK4 is not, making it better for gerneal ODE:s. RK4 is moreover an almost universal method, except for Hamiltonian ODE:s over great times, as it has a great stability area and can handle imaginary eigenvalues.

Since CD4 is as accurate as RK4, offers symplectic behaviour for our system and require far less computational operations, I would argue that CD4 is the best method for this specific situation (wave equations).

13 The two dimensional problem

A 2D extension of (9) with Neumann BC onto the square domain restricted by $1 \leq x, y \leq 1$ is given by

$$\begin{cases} u_{tt} = c^2(u_{xx} + u_{yy}), & -1 \leq x, y \leq 1, t \geq 0 \\ u_{x,y} = 0, & x, y \in \delta\Omega \\ u = f(x, y), u_t(0) = 0, & -1 \leq x, y \leq 1, t = 0 \end{cases} \tag{29}$$

where $\delta\Omega$ represent the boundary of the computational domain (here defined by the square). By $u_{x,y} = 0$, we simply mean that the normal derivative is set to zero at each of the four boundaries ($u_x = 0$ on left and right boundary and $u_y = 0$ on top and bottom boundary). An equivalent 2D extension of (9) with Dirichlet BC is given by,

$$\begin{cases} u_{tt} = c^2(u_{xx} + u_{yy}), & -1 \leq x, y \leq 1, t \geq 0 \\ u = 0, & x, y \in \delta\Omega u = f(x, y), u_t(0) = 0, \\ -1 \leq x, y \leq 1, t = 0 \end{cases} \tag{30}$$

To simplify notation (without any restriction) we assume the same dimension (m) in both the x- and y-direction. The square domain is discretized with an m ×m-point equidistant grid defined as

$$x_j = -1 + (j-1)h, j = 1, 2, ...m, h = \frac{2}{m-1} \tag{31}$$

$$x_i = -1 + (i-1)h, i = 1, 2, ...m, h = \frac{2}{m-1} \tag{32}$$

The numerical approximation at grid point (x_j, y_i) is denoted $V_{j,i}$. The discrete solution can be represented as a m ×m matrix.

$$V = \begin{bmatrix} V_{1,1} & \dots & V_{1,m} \\ \vdots & \ddots & \vdots \\ V_{m,1} & \dots & V_{m,m} \end{bmatrix} \tag{33}$$

To define the grid for the 2D domain we can utilize the Matlab function meshgrid (the function call in Matlab: [X,Y]=meshgrid(x,y)).

Let $c^2 B$ denote the discretisation matrix for the 1D problem, presented in (13). The semi-discrete approximation of the 2D problem is given by,

$$\begin{cases} V_{tt} = c^2(BV + VB^T), & t \geq 0 \\ V = F, V_t = 0, & t = 0 \end{cases} \tag{34}$$

where

$$F = \begin{bmatrix} F_{1,1} & \dots & F_{1,m} \\ \vdots & \ddots & \vdots \\ F_{m,1} & \dots & F_{m,m} \end{bmatrix} \tag{35}$$

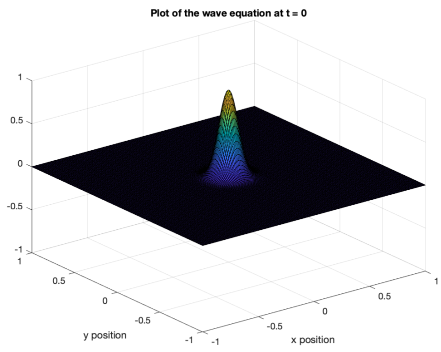
is the initial data (a *mxm* - matrix). Here BT denotes the transpose of B. Here B V is a matrix-matrix multiplication.

14 Assignment 8

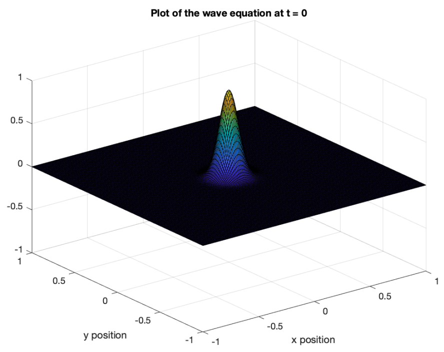
Discretise (34) with either RK4 or CD2. Write a Matlab program that simulates the wave propagation using $u(x, y, 0) = f(x, y) = e^{-100(x^2+y^2)}$ and $u_t(x, y, 0) = 0$ as initial data. Solve the problem with both Dirichlet and Neumann BC (and a mix if you prefer), and set c = 1. Present plots of the solution at t = 0, t = 1, t = 2 and t = 10. (Use m = 201.) In Figure 3 the initial Gaussian and numerical solution at t = 0.5, t = 1.2 and t = 3.0 is presented with 201 ×201 grid points (and at t = 2, first page), using Neumann BC.

14.1 Solution Assignment 8

Equation (34) is discretisied using central difference of second order, and figure 16, 17, 18 and 19 shows the wave equation at some time points, for both Dirichlet and Neumann boundary condition using $m = 201$ and $c = 1$.

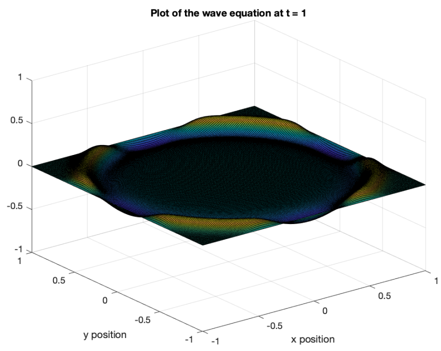


(a)

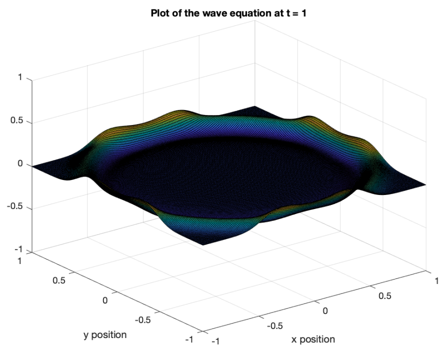


(b)

Figure 16: Plot of the wave equation at t=0 s using boundary condition (a) Dirichlet (b) Neumann



(a)



(b)

Figure 17: Plot of the wave equation at t=1 s using boundary condition (a) Dirichlet (b) Neumann

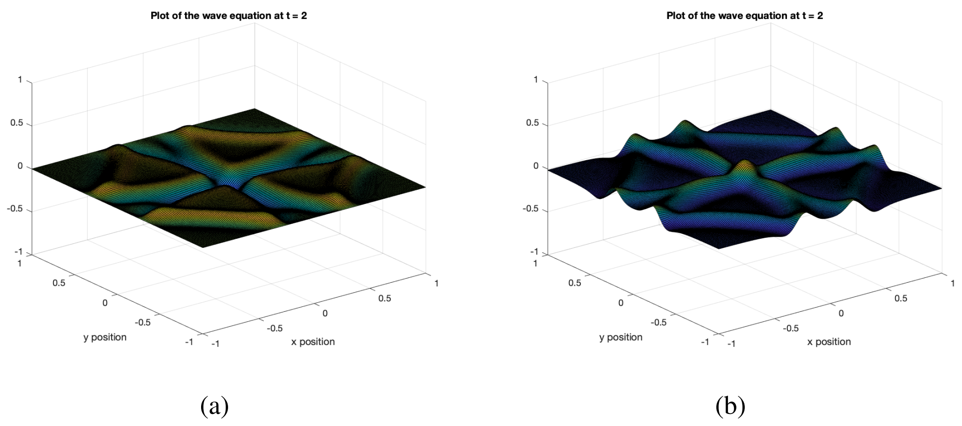


Figure 18: Plot of the wave equation at t=2 s using boundary condition (a) Dirichlet (b) Neumann

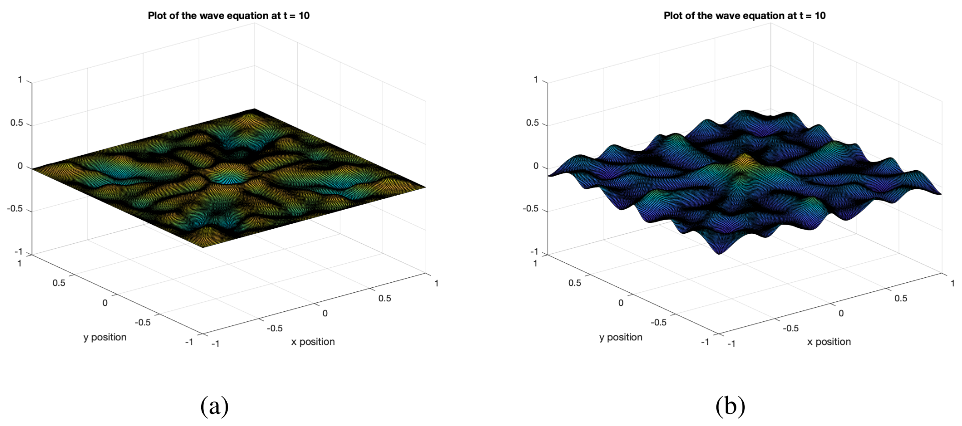


Figure 19: Plot of the wave equation at t=10 s using boundary condition (a) Dirichlet (b) Neumann

15 Conclusion

In conclusion, we have used two different numerical methods; CD2 and RK4, to solve the wave equation in both the one dimensional case, and the two dimensional case, for Dirhclet and Neumann boundary condition. We have verified that the accuracy of the models are indeed correct, and discussed pros and cons of using central difference methods versus runge kutta methods when it comes to wave equations.

A Appendix

A.1 Assignment 1

```
clear variables; close all; clc;

% Define variables
c = 1;
m = 31;
[BD,BN] = FSBP4(m);

% Numerical calculation
num_eig1 = 1i*abs(sqrt(eig(c^2*BD)))';
num_eig2 = -1i*abs(sqrt(eig(c^2*BD)))';

% Analytical calculation

lambda_pos = @(n) 1i*c*n*pi/2;
lambda_neg = @(n) -1i*c*n*pi/2;
n = 1:1:m;

ana_eig1 = lambda_pos(n);
ana_eig2 = lambda_neg(n);

% Calculate errors

error_pos = abs(fliplr(num_eig1)-ana_eig1);
error_neg = abs(fliplr(num_eig2)-ana_eig2);

% Visualize the results
figure(1)
hold on

plot(num_eig1, "g*")
plot(num_eig2, "g*")
xlabel("Real axis")
ylabel("Imaginary axis")
title("Eigenvalues solved numerically")
hold off

figure(2)
hold on
plot(ana_eig1, "bo")
plot(ana_eig2, "bo")
xlabel("Real axis")
ylabel("Imaginary axis")
title("Eigenvalues solved analytically")
```

```
hold off
```

A.2 Assignment 2

```
clear variables; close all; clc;

% Define variables
c = 1;
m = 31;
[BD,BN] = FSBP4(m);
h = 2/(m-1);

% Numerical calculation
num_eig1 = 1i*abs(sqrt(eig(c^2*BD))); % Dirichlet
num_eig2 = 1i*abs(sqrt(eig(c^2*BN))); % Neumann

% Find the biggest eigenvalue
eig_max1 = max(abs(num_eig1));
eig_max2 = max(abs(num_eig2));

gamma1 = h*eig_max1/c;
gamma2 = h*eig_max2/c;

disp("Biggest gamma for Dirichlet using c = " + c + " and m ...
    = " + m + " is: "+ gamma1)
disp("Biggest gamma for Neumann using c = " + c + " and m = ...
    " + m + " is: "+ gamma2)
```

A.3 Assignment 3

A.3.1 Assignment3.m

```
clear varibales; close all; clc;

% Set variables

m = 101;        % Grid points
T = 10000;      % End time
dx = 2/(m-1);   % Step
dt = 0.01;      % Time step
BC = "N";       % Boundary condition, "N" for Neumann and "D" ...
    for Dirichlet
c = 1;          % Wave speed

gamma_D = 3.0009;
gamma_N = 2.3053;
```

```

k_star_D = dx*2.8/(c*gamma_D);
k_star_N = dx*2.8/(c*gamma_N);

v = RK4(m, T, dx, dt, BC, c);

%animation(T,dt,dx,v,0,10)

%plot_snapshot(dx,dt,T,1.8,v)
```

A.3.2 RK4.m

```
function v = RK4(m,T,dx,dt,BC,c)
    % m - grid points
    % T - end time
    % dx - step
    % dt - time step
    % BC either "D" for Dirichlet or "N" for Neumann
    % c - wave speed

    [BD, BN] = FSBP4(m); % Discretisation matrix

    if BC == "D"
        A = [zeros(m,m), eye(m); c^2*BD, zeros(m,m)];
    elseif BC == "N"
        A = [zeros(m,m), eye(m); c^2*BN, zeros(m,m)];
    end

    x = -1:dx:1;   % x vals
    t = 0:dt:T;    % t vals

    % Initiate the solution matrixes
    % Number of rows are number of time values. Number of ...
        columns are number of x
    % values. Think of 1 row as a snapshot of the string

    v = zeros(length(t),length(x));
    w = zeros(length(t),length(x));

    % Apply initial condition

    v(1,:) = exp(-(x./0.2).^2);
    w(1,:) = 0;

    % Implementing RK4
    for i = 1:length(t)-1

        u = [v(i,:),w(i,:)].';

        w1 = func(A, u);
        w2 = func(A, u + dt*0.5*w1);
        w3 = func(A, u + dt*0.5*w2);
        w4 = func(A, u + dt*w3);

        u = u + dt*(w1 + 2*w2 + 2*w3 + w4)/6;

        v(i+1,:) = u(1:m).';
        w(i+1,:) = u(m+1:2*m).';
    end
end
```

A.3.3 func.m

```
function f = func(A,u)
    f = A*u;
end
```

A.3.4 animation.m

```
function animation(T,dt,dx,v,time1,time2)
    % T - end time
    % dt - time step
    % dx - step
    % v - matrix of wave equation
    % time1 - time point the animation will start
    % time2 - time point the animation will end

    t = 0:dt:T;    % t vals
    x = -1:dx:1;   % x vals

    % Find the index of the time points
    index1 = find(t>=time1);
    index1 = index1(1);

    index2 = find(t>=time2);
    index2 = index2(1);

    vidObj = VideoWriter('video.avi');
    open(vidObj);

    for k = index1:index2
        plot(x,v(k,:))
        axis([-1 1 -1.1 1.1]) % Set x-axis and y-axis
        xlabel('x position')
        ylabel('y position')
        title('Animation of the wave equation')
        frame = getframe(gcf);
        writeVideo(vidObj,frame);
    end

    close(vidObj);
end
```

A.3.5 plotsnapshot.m

```
function plot_snapshot(dx,dt,T,time,v)
    % dx - step
    % dt - time step
    % T - end time
    % time - time point the equation will be plotted
    % v - matrix of wave equation

    x = -1:dx:1;  % x vals
    t = 0:dt:T;   % t vals

    % Find index of the time point
    index = find(t>=time);
    index = index(1);

    plot(x,v(index,:));
    axis([-1 1 -1 1]) % Set x-axis and y-axis
    xlabel("x position");
    ylabel("y position");
    title_prompt = ["Plot of the wave equation at t = " + time];
    title(title_prompt);
end
```

A.4 Assignment 4

```
clear variables; close all; clc;

% Given variables
m2 = [101,201,301,501]; % Grid points
m1 = [51,101,201,301];
T = 1.8; % End time
c = 1;    % Wave speed

% Analytical solution

uN = @(x,t) ...
    +0.5.*exp(-((x+2-c*t)./0.2).^2)+0.5.*exp(-((x-2+c*t)./0.2).^2);
```

```
uD = @(x,t) ...
    -0.5.*exp(-((x+2-c*t)./0.2).^2)-0.5.*exp(-((x-2+c*t)./0.2).^2);

qD = zeros(1, length(m1)); % Convergence rate for dirichlet ...
    boundary condition
qN = zeros(1, length(m1)); % Convergence rate for neumann ...
    boundary condition

error_m1_N = zeros(1,length(m1));
error_m2_N = zeros(1,length(m1));
error_m1_D = zeros(1,length(m1));
error_m2_D = zeros(1,length(m1));

norm_1_N = zeros(1,length(m1));
norm_2_N = zeros(1,length(m1));
norm_1_D = zeros(1,length(m1));
norm_2_D = zeros(1,length(m1));

% This for loop calculates the errors, l2 norm, and ...
    convergence rate
% for dirichlet and neumann boundary condition
for k = 1:length(m1)

    % Calculate x step
    dx_1 = 2/(m1(k)-1);
    dx_2 = 2/(m2(k)-1);

    % Calculate x values
    x1 = -1:dx_1:1;
    x2 = -1:dx_2:1;

    % Calculate t step
    dt_1 = 0.1*dx_1;
    dt_2 = 0.1*dx_2;

    % Calculate t values
    t1 = 0:dt_1:T;
    t2 = 0:dt_2:T;

    % Calculate the numerical solutions
    vD1 = RK4(m1(k), T, dx_1, dt_1, "D", c);
    vD2 = RK4(m2(k), T, dx_2, dt_2, "D", c);
    vN1 = RK4(m1(k), T, dx_1, dt_1, "N", c);
    vN2 = RK4(m2(k), T, dx_2, dt_2, "N", c);

    % Take the last row
    vD1 = vD1(length(t1), :);
    vD2 = vD2(length(t2), :);
    vN1 = vN1(length(t1), :);
    vN2 = vN2(length(t2), :);

    % Calculate the analytical solutions
    D = @(x) uD(x,T);
    N = @(x) uN(x,T);
    analytical_D_1 = D(x1);
    analytical_D_2 = D(x2);
    analytical_N_1 = N(x1);
    analytical_N_2 = N(x2);

    % Calculates the errors
    error_m1_N = abs(sum(analytical_N_1-vN1));
    error_m2_N = abs(sum(analytical_N_2-vN2));
    error_m1_D = abs(sum(analytical_D_1-vD1));
    error_m2_D = abs(sum(analytical_D_2-vD2));


    % Calculate the l2 norm
    l2_D_1 = sqrt(1/m1(k)) * norm(analytical_D_1-vD1);
    l2_D_2 = sqrt(1/m2(k)) * norm(analytical_D_2-vD2);
    l2_N_1 = sqrt(1/m1(k)) * norm(analytical_N_1-vN1);
    l2_N_2 = sqrt(1/m2(k)) * norm(analytical_N_2-vN2);

    % Save the norms
    norm_1_N(k) = l2_N_1;
    norm_2_N(k) = l2_N_2;
    norm_1_D(k) = l2_D_1;
    norm_2_D(k) = l2_D_2;

    % Calculate the convergence rates
    qD(k) = log10(l2_D_1/l2_D_2)/log10(m2(k)/m1(k));
    qN(k) = log10(l2_N_1/l2_N_2)/log10(m2(k)/m1(k));
end
```

A.5 Assignment 6

A.5.1 Assignment6.m

```
clear variables; close all; clc;

% Set variables

m = 101;        % Grid points
T = 10000;         % End time
dt = 0.01;      % Time step
dx = 2/(m-1);   % Step
BC = "N";       % Boundary condition
c = 1;          % Wave speed
```

```matlab



v = CD2(m,T,dx,dt,BC,c);

%animation(T,dt,dx,v,0,10);
%plot_snapshot(dx,dt,T,6,v)
```

A.5.2 CD2.m

```matlab
function v = CD2(m,T,dx,dt,BC,c)

    [BD, BN] = FSBP4(m); % Discretisation matrix

    if BC == "D"
        B = BD;
    elseif BC == "N"
        B = BN;
    end
    x = -1:dx:1;  % x vals
    t = 0:dt:T;   % t vals

    % Initiate the solution matrix
    v = zeros(length(t),length(x));

    % Apply initial condition
    f = exp(-(x./0.2).^2)';
    v(1,:) = f;
    matrix = eye(m)+dt^(2)*c^(2)*0.5*B;
    v(2,:) = matrix*f;

    for i = 2:length(t)-1
        v(i+1,:) = dt^(2)*c^(2)*B*v(i,:)' + 2*v(i,:)' - ...
            v(i-1,:)';
    end

end
```

A.6 Assignment 7

A.6.1 Assignment7.m

```matlab
clear variables; close all; clc;

% Given variables
m1 = [101,201,301,501]; % Grid points
m2 = [51,101,201,301];
```

```
T = 1.8; % End time
c = 1;   % Wave speed

% Analytical solution

uN = @(x,t) ...
    +0.5.*exp(-((x+2-c*t)./0.2).^2)+0.5.*exp(-((x-2+c*t)./0.2).^2);
uD = @(x,t) ...
    -0.5.*exp(-((x+2-c*t)./0.2).^2)-0.5.*exp(-((x-2+c*t)./0.2).^2);

qD = zeros(1, length(m1)); % Convergence rate for dirichlet ...
    boundary condition
qN = zeros(1, length(m1)); % Convergence rate for neumann ...
    boundary condition

error_m1_N = zeros(1,length(m1));
error_m2_N = zeros(1,length(m1));
error_m1_D = zeros(1,length(m1));
error_m2_D = zeros(1,length(m1));

norm_1_N = zeros(1,length(m1));
norm_2_N = zeros(1,length(m1));
norm_1_D = zeros(1,length(m1));
norm_2_D = zeros(1,length(m1));

% This for loop calculates the errors, l2 norm, and ...
    convergence rate
% for dirichlet and neumann boundary condition
for k = 1:length(m1)

    % Calculate x step
    dx_1 = 2/(m1(k)-1);
    dx_2 = 2/(m2(k)-1);

    % Calculate x values
    x1 = -1:dx_1:1;
    x2 = -1:dx_2:1;

    % Calculate t step
    dt_1 = 0.1*dx_1;
    dt_2 = 0.1*dx_2;

    % Calculate t values
    t1 = 0:dt_1:T;
    t2 = 0:dt_2:T;

    % Calculate the numerical solutions
    vD1 = CD2(m1(k), T, dx_1, dt_1, "D", c);
    vD2 = CD2(m2(k), T, dx_2, dt_2, "D", c);
    vN1 = CD2(m1(k), T, dx_1, dt_1, "N", c);
    vN2 = CD2(m2(k), T, dx_2, dt_2, "N", c);

    % Take the last row
    vD1 = vD1(length(t1), :);
    vD2 = vD2(length(t2), :);
    vN1 = vN1(length(t1), :);
    vN2 = vN2(length(t2), :);

    % Calculate the analytical solutions
    D = @(x) uD(x,T);
    N = @(x) uN(x,T);
    analytical_D_1 = D(x1);
    analytical_D_2 = D(x2);
    analytical_N_1 = N(x1);
    analytical_N_2 = N(x2);

    % Calculates the errors
    error_m1_N = abs(sum(analytical_N_1-vN1));
    error_m2_N = abs(sum(analytical_N_2-vN2));
    error_m1_D = abs(sum(analytical_D_1-vD1));
    error_m2_D = abs(sum(analytical_D_2-vD2));

    % Calculate the l2 norm
    l2_D_1 = sqrt(1/m1(k)) * norm(analytical_D_1-vD1);
    l2_D_2 = sqrt(1/m2(k)) * norm(analytical_D_2-vD2);
    l2_N_1 = sqrt(1/m1(k)) * norm(analytical_N_1-vN1);
    l2_N_2 = sqrt(1/m2(k)) * norm(analytical_N_2-vN2);

    % Save the norms
    norm_1_N(k) = l2_N_1;
    norm_2_N(k) = l2_N_2;
    norm_1_D(k) = l2_D_1;
    norm_2_D(k) = l2_D_2;

    % Calculate the convergence rates
    qD(k) = log10(l2_D_1/l2_D_2)/log10(m2(k)/m1(k));
    qN(k) = log10(l2_N_1/l2_N_2)/log10(m2(k)/m1(k));
end
```

A.7 Assignment 8

A.7.1 Assignment8.m

```
clear varibles; close all; clc;

% Set variables

```

```
m = 201;       % Grid points
dx = 2/(m-1); % x step
dy = 2/(m-1); % y step
c = 1;         % Tave speed
T = 10;        % Tnd time
dt = 0.001;    % Time step
t = 0:dt:T;    % Time values
BC = "N";      % Boundary condition


V = CD2_2D(m,T,dx,dy,dt,BC,c);


%animation_3D(T,dt,dx,dy,V,0,10)

%plot_snapshot_3D(dx,dy,dt,T,1.8,V)
```

A.7.2 CD22D.m

```
function V = CD2_2D(m,T,dx,dy,dt,BC,c)

    [BD, BN] = FSBP4(m); % Discretisation matrix
    if BC == "D"
        B = sparse(BD);
    elseif BC == "N"
        B = sparse(BN);
    end

    x = -1:dx:1;
    y = -1:dy:1;
    t = 0:dt:T;

    % Initiate the discrete solution matrix

    V = zeros(length(x), length(x), length(t));

    % Apply initial conditions

    f = @(x,y) exp(-100*(x.^(2)+y.^(2)));

    for b = 1:length(x)
        V(b,:,1) = sparse(f(x(b),y));
    end
    for i = 1:length(x)
        V(i,:,2) = sparse(dt^(2)*c^(2)*0.5*(B * f(x(i),y)' + ...
            transpose(f(x(i), y)*B'))' + f(x(i), y));
    end

    % Implement CD2 algorithm

    for k = 2:length(t)-1
        V(:,:,k+1) = sparse(2*V(:,:,k) - V(:,:,k-1) + ...
            c^(2)*dt^(2)*(B*V(:,:,k) + V(:,:,k)*transpose(B)));
    end

end
```

A.7.3 animation3D.m

```
function animation_3D(T,dt,dx,dy,v,time1,time2)
    % T - end time
    % dt - time step
    % dx - step
    % v - matrix of wave equation
    % time1 - time point the animation will start
    % time2 - time point the animation will end

    t = 0:dt:T;    % t vals
    x = -1:dx:1;   % x vals
    y = -1:dy:1;

    index1 = find(t>=time1);
    index1 = index1(1);

    index2 = find(t>=time2);
    index2 = index2(1);

    vidObj = VideoWriter('video.avi');
    open(vidObj);

    for k = index1:index2
        [X,Y] = meshgrid(x,y);
        surf(X,Y,v(:,:,k));

        axis([-1 1 -1 1 -1 1]) % Set x-axis and y-axis
        frame = getframe(gcf);
        writeVideo(vidObj,frame);
    end

    close(vidObj);
end
```

A.7.4 plotsnapshot3d

```
function plot_snapshot_3D(dx,dy,dt,T,time,V)
```

```
    % dx - step
    % dt - time step
    % T - end time
    % time - time point the equation will be plotted
    % v - matrix of wave equation

    x = -1:dx:1;  % x vals
    y = -1:dy:1;
    t = 0:dt:T;   % t vals

    index = find(t>=time);
    index = index(1);

    [X,Y] = meshgrid(x,y);
    surf(X,Y,V(:,:,index));
    axis([-1 1 -1 1 -1 1]) % Set x-axis and y-axis
    xlabel("x position");
    ylabel("y position");
    title_prompt = ["Plot of the wave equation at t = " + time];
    title(title_prompt);
end
```

A.8 FSBP4.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% 4th order SBP Finite differece Op.          %%%
%%% Derived by Ken Mattsson                     %%%
%%%                                             %%%
%%% H            (Norm)                         %%%
%%% D1           (first derivative)             %%%
%%% D2           (second derivative)            %%%
%%%                                             %%%
%%% AD is the discretisation with Dirichlet BC  %%%
%%% AN is the discretisation with Neumann BC    %%%
%%% or Robin:                                   %%%
%%%                                             %%%
%%% b*u+u_x=0, x=x_l                            %%%
%%% b*u-u_x=0, x=x_r                            %%%
%%%                                             %%%
%%% here b <= 0 (b=0 is Neumann)                %%%
%%%                                             %%%
%%% m is the number opf grid-points, L is the   %%%
%%% width of the domain. Gris-space h=L/(m-1)   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [AD, AN]=FSBP4(m,b)

x_l=-1; x_r=1;

```

```
h=(x_r-x_l)/(m-1); % Grid space

if nargin == 1
    b=0;
end

%

H=diag(ones(m,1),0);
H(1:4,1:4)=diag([17/48 59/48 43/48 49/48]);
H(m-3:m,m-3:m)=fliplr(flipud(diag([17/48 59/48 43/48 49/48])));
H=H*h;
HI=inv(H);


Q=(-1/12*diag(ones(m-2,1),2)+8/12*diag(ones(m-1,1),1)-8/12*diag(ones(m-1,1),-1)+1/1
Q_U = [0 0.59e2 / 0.96e2 -0.1e1 / 0.12e2 -0.1e1 / 0.32e2; ...
    -0.59e2 / 0.96e2 0 0.59e2 / 0.96e2 0; 0.1e1 / 0.12e2 ...
    -0.59e2 / 0.96e2 0 0.59e2 / 0.96e2; 0.1e1 / 0.32e2 0 ...
    -0.59e2 / 0.96e2 0;];
Q(1:4,1:4)=Q_U;
Q(m-3:m,m-3:m)=flipud( fliplr(-Q_U(1:4,1:4) ) );

e_1=zeros(m,1);e_1(1)=1;
e_m=zeros(m,1);e_m(m)=1;

D1=HI*(Q-1/2*e_1*e_1'+1/2*e_m*e_m') ;

M_U=[0.9e1 / 0.8e1 -0.59e2 / 0.48e2 0.1e1 / 0.12e2 0.1e1 / ...
    0.48e2; -0.59e2 / 0.48e2 0.59e2 / 0.24e2 -0.59e2 / 0.48e2 ...
    0; 0.1e1 / 0.12e2 -0.59e2 / 0.48e2 0.55e2 / 0.24e2 ...
    -0.59e2 / 0.48e2; 0.1e1 / 0.48e2 0 -0.59e2 / 0.48e2 ...
    0.59e2 / 0.24e2;];
M=-(-1/12*diag(ones(m-2,1),2)+16/12*diag(ones(m-1,1),1)+16/12*diag(ones(m-1,1),-1)

M(1:4,1:4)=M_U;

M(m-3:m,m-3:m)=flipud( fliplr( M_U ) );
M=M/h;

S_U=[-0.11e2 / 0.6e1 3 -0.3e1 / 0.2e1 0.1e1 / 0.3e1;]/h;
S_1=zeros(1,m);
S_1(1:4)=S_U;
S_m=zeros(1,m);
S_m(m-3:m)=fliplr(-S_U);

D2=HI*(-M-e_1*S_1+e_m*S_m);


```

```
alpha=.2508560249;      % How much we can borrow
penalty=1.2;              % must be >= 1, higher leads to ...
    stiffness
tau=-penalty/(h*alpha); % Needed for Dirichlet BC

AN=-HI*M+b*(e_1*e_1'+e_m*e_m');
AD=D2+HI*(-e_1*S_1+e_m*S_m)'+tau*HI*(e_1*e_1'+e_m*e_m');
```