CP II; Membrane Simulation

Brandon Parfimczyk & Kien Do

30-03-2020

Contents

1 Manual

1. Compiling: Run 'make all' in the project root folder). This compiles all the tests and the main program into the "./executables/" folder.
2. for main program: change directories to ../executables and run ' ./membrane-sim' (for more information and help use -h)
3. if all *.dat have been created, calculate observables and get graphs for numerical analysis: ../data/gen_e_violation.sh and ../data/gen_pos_mom
4. generate *.png for visualizations ../data/extract_e_violation_p.sh and ../data/extract_pos_mom
5. to get the animation of the membrane in *.gif go to ../data/gnuplot and gnuplot » load 'arr_3d_gif.plt'

2 Planning

The planning in this project was the following:

03.03 - Assemblance, read the notes and implement the algorithm for the integrators, discuss the skeleton of the main program

07.03. - restructuring of codes and generalization for potentials in integrator algorithms

10.03. - implement the initial conditions

14.03. - make the plots and animations, discuss how to obtain the continuous cases from the discretizations

22.03. - discuss results

30.03. - finalize everything in report.

3 Physical Transcription

In this project we want to determine the time evolution of a membrane in two dimensions, modeled by the Hamiltonian

$$\hat{H} = \sum_{n_1,n_2=0}^{N} a^2 \left\{ \frac{1}{2}\pi^2(\mathbf{n}) - \frac{1}{2} h \hat{\nabla}^2 h(\mathbf{n}) + V(h(\mathbf{n})) \right\}. \tag{1}$$

h stands for the displacement of the membrane and the potential will be:

$$V(h(\mathbf{n})) = \frac{\lambda}{4!} h^4 \tag{2}$$

Just like in previous projects we are using a discretization of space and time ($\mu \in (0,1)$):

$$T = \tau * \text{RUNS} \tag{3}$$

$$n_\mu \in \frac{1}{N-1}(0,1,...,N-1). \tag{4}$$

For this project we don't implement Periodic Boundary conditions anymore, insted we fix our boundaries to

$$h(\mathbf{n}) = 0, \tag{5}$$

for $\exists \mu : n_\mu = 0$.
We can derive the equations of motion with the Hamilton formalism and acquire

$$\ddot{h}_t = \hat{\nabla}^2 h(\mathbf{n}) - V'(h(\mathbf{n})) \tag{6}$$

We will solve this by using two symplectic integrators: (**a**) Leapfrog (2nd order) and (**b**) Yoshida (4th order) integrator. The results will be displayed and analyzed in physical and numerical terms. The project will be concluded by finding suitable values for the quantities linear lattice size N and integration timestep τ to have a meaningful appoximation to the continuous system.

4 Code Recycling

We can use some code of the previous project. This includes the codes for generating indices from coordinates and vice versa and parts of code for the Laplacian, since this already uses the nearest neighbours. The recycled functions are:

In Module [geometry.c]:

1. `ind2coord()`
2. `coord2ind()`
3. `nn_neighbour()` (with modifications for the non PBC)

4. `nn_create()`

In Module [functions.c]:

1. `laplace_arr()` (with modifications for the non PBC)

This will now ignore the points which are border points of our system. The array is created by the function `is_border_create()` which saves the information of `is_border(int index)` for all indices. We save routines in the Laplacian by doing this.

We do not have to implement the non PBC explicitly in the integrators when the inital conditions for all border points in the pi array are zero. The border points will be updated but it is just adding zeroes. When the boundaries are not equal 0 then the simulation will have a constant drift into the direction of the momentum at the borders and will not be physical anymore. Care if changing the boundary conditions. You will have to modify the integrators to ignore the boundary points at all. The following code snippet could be useful for this:

```
1 for (int ind = 0; ind < ipow(N, D); ind++) {
2         if (is_border_arr[ind]) {
3                 out[ind] = h[ind];
4                 continue;
5                 }
6         }
```

5 Modules and some relevant functions

Module [`[functions.c]`] contains

- `is_border()` checks wether our index is a border point.
- `laplace_arr()` to calculate the laplace acting on the displacement in Eq. (1)
- `dpolynom()` was `laplace_arr()` to calculate the laplace acting on the displacement in Eq. (1).firstly being implemented for a generic power potential. To generalize the implementation of a more generic potential, see [`potentials.c`].

Module [`initial_cond.c`] contains

- `init_h_pyramid_a()` to set up the initial condition for our differential equation (6). It is an approximate pyramid.
  Achieved by putting identical triangles next to each other on top of the base, a $(N-1)x(N-1)$ square along axis 1. The object looks like the roof of a typical dream house of

a four headed family now.
Then identify (on the base) the set of points (two triangles) where we have to do the same as in the previous step, but now along axis 0.

- `init_h_pyramid_b()` is another initial condition. However, the edges from the top of the pyramid don't connect to the corners of the base square.
- `init_pi_zero()` sets all our initial π_0 to 0.

Module [`integrators.c`] contains

- `integ_set_integrator()` which basically points at the address of one of the available integrators below.
- `leapfrog()`
- `integ_yoshida()`

observables.c contains

- `hamilton()` calculates the energy of the system at the integeration time t given we know the displacement h_t and π_t

potential.c contains

- `pot_set_func()` which basically points at the address of one of the available potentials below for the energy calculation or their respective derivative for the usage of the integrators.

5.1 Note: Challenge encountered in implementing a generic potential

Different from the project qunatum mechanical particle in a potential where we had a potential which was not dependent on the state of the particle (here: displacement of the membrane at position **n**) we had to think how to implement a generic potential, Brandon came up with the idea to use pointers. We implemented the potentials (the computer reserves an address for them) needed beforehand and in need we just use a function `pot_set_func()` which points at the adress in question. This way we didn't need to hardcode a specific potential in the algorithm of the integrator.

6 Main program: membrane-sim.c

By defining the preprocessor token `'MAINPROGRAM'` in all the main programs we can get the global variables defined inside main source file and extern in all modules by `global.h`. This enables us to change global variables at runtime. We use this for the command line handling. All the possible options are:

```
Usage: membrane-sim [options]...

The following options are available:
    -N <integer> number of lattice points in one dimension, positive integer
    -D <integer> number of dimensions, positive integer
    -i <integer> choice of integrator:
                        0 yoshida integrator
                        1 leapfrog integrator
    -p <integer> choice of potential:
                        0 no potential
                        1 higgs potential
    -w <double> set param0 for use in potential
    -t <float> time step tau for a single integration step
    -T <float> ending time t_end
    -e <integer> output observables and field arrays only every <integer>th
                    time integration step
    -f <file> write output field to "<file>" and observables to
                    "<file>_obs"
                    if the specified filename has an filetype extension then
                    the resulting observable file will share the same extension
                    e.g. simulated_system.dat -> simulated_system_obs.dat
                            membrane_sys -> membrane_sys_obs
                    if this option is not specified the program will print the
                    field array data to stdout and omit printing observables
    -h show this help text
```

In the program we can choose the initial coordinates to place the top of the pyramid and the height of the pyramid.
Using `pot_set_func()` and `integ_set_integrator()` we can choose the integrator and potential to work with. `integ_integrator()` then performs the integration until T has been reached. The program also calculates the energy and the energy difference of the system at the point of time 0 and t and also the displacement and its velocity at the middle of the field and

saves it in a seperate file.

7 Testing

7.1 NN-Testing

This had to be modificated to work with the non PBC in this project.

7.2 Testing Leapfrog And Yoshida Integrator

To test wether both integrators do their job like intended we used them on a very simple system of differential equation of a free falling object:

$$\dot{v} = -g \tag{7}$$

$$\dot{x} = v. \tag{8}$$

So we chose N = D = 1, since we have a single object instead of a field. In the algorithm of an integrator the calculation of the acceleration is crucial for example in [integrators.c] `leapfrog()`:
We have to set `pot_deriv_func(h[n]) = h_F[n] + g`. Since $N = 1$ and the boundary conditions of `laplace_arr()` ($h_F[0] = h[0]$) we implement

```
1 double pot_deriv_newt_grav(double h)
2 {
3 return h + g_newt;
4 }
```

The test is succesful, if the difference with the analytical solution of the ODE [7] is below machine precision:

$$x(t) = -\frac{g}{2}t^2 + v_0 t + x_0. \tag{9}$$

7.3 Testing Approximation Of Continous Pyramid

The test is summing all displacement values together and compares it with the analytical volume of an exact pyramid:

$$V = \frac{1}{3} A_{sq} * h. \tag{10}$$

In theory we just need to do $N \to \infty$ and the difference should approach 0. Finite ressources can't achieve this.
But if there is a bijective relationship of a geometrical object to a volume formula, all we have

to do is see wether with scaling N, the difference is monotoneous decreasing.
In doubt we plotted the initial condition, as one can see if one runs the script ../data/gnuplot/arr_3d_init.plt.

7.4 Testing Linearity Of Laplace Function

test_functions: `[laplace_arr()` basically just applies first the laplace operator on two summands and adds them together and second applies laplace on the sum of both summands and comapres both. The difference is being divided by the Volume of the lattice to avoid scaling errors.

8 Result

8.1 Physical analysis

Eq. [1] which decribes the energy of the system. Calculating the time derivative analytically of this function gives 0. That means that energy should be conserved. Looking at the observable `*_obs.dat` we see that the average error stays constant (up to a certain digit after the comma), which hints that we did something right. Increasing the integration time step τ should decrease the energy difference even more. We agreed on that τ is small enough if the energy difference is in order of machine precision (10^{-15}).
In the case of having no potential and choosing the initial conditions to be in a middle of the pyramid we actually see a standing wave emerging as the mebrane simply just moves up and down, what was to be expected.
In the case of using the Higgs potential the animation shows a non trivial time evolution of the membrane. As the pyramid like initial condition flattens (non uniformly) and the wave starts to spread out, it doesn't recover the pyramid back as the waves hit the boundaries and come back and interferes. One sees that there are more than just one mode occuring (one can see it more clearly, when viewing it from the side as in some point of time the field is partially positively displaced and partially negatively displaced.)

8.2 Numerical analysis

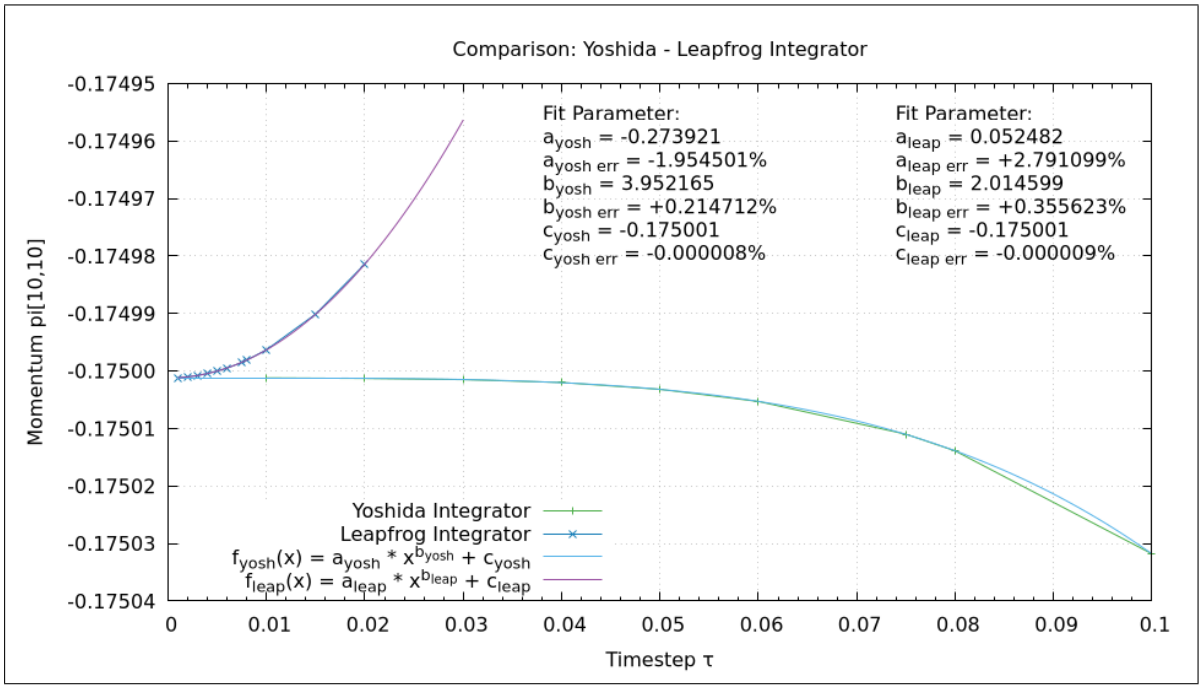

Figure 1: Momentum at T = 12 versus integrator time step for both integrators.

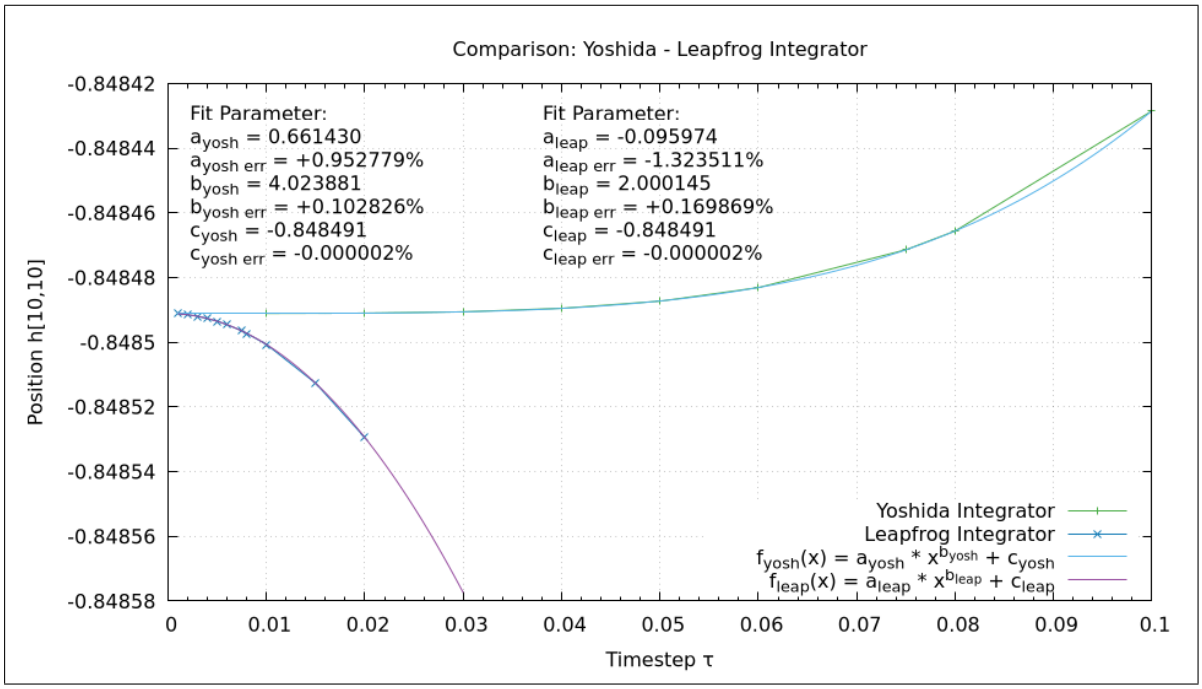

Figure 2: Position at T = 12 versus integrator time step for both integrators.

In fig. [1] we can see that the fit parameter give us the right results. (b = 2 und b = 4) we get the right orders for τ for the respective integrators back. It also tells us that if we want to calculate an observable with one magnitude higher precision, we need to choose $\tau \to \tau \cdot 10^{-\frac{1}{\text{order of integrator}}}$. From this figure we can not read off a good value for τ. However, we see the behaviour that both integrators approach the value -0.175 for the momentum as τ goes to 0. A similar observation can be made in fig. [2]

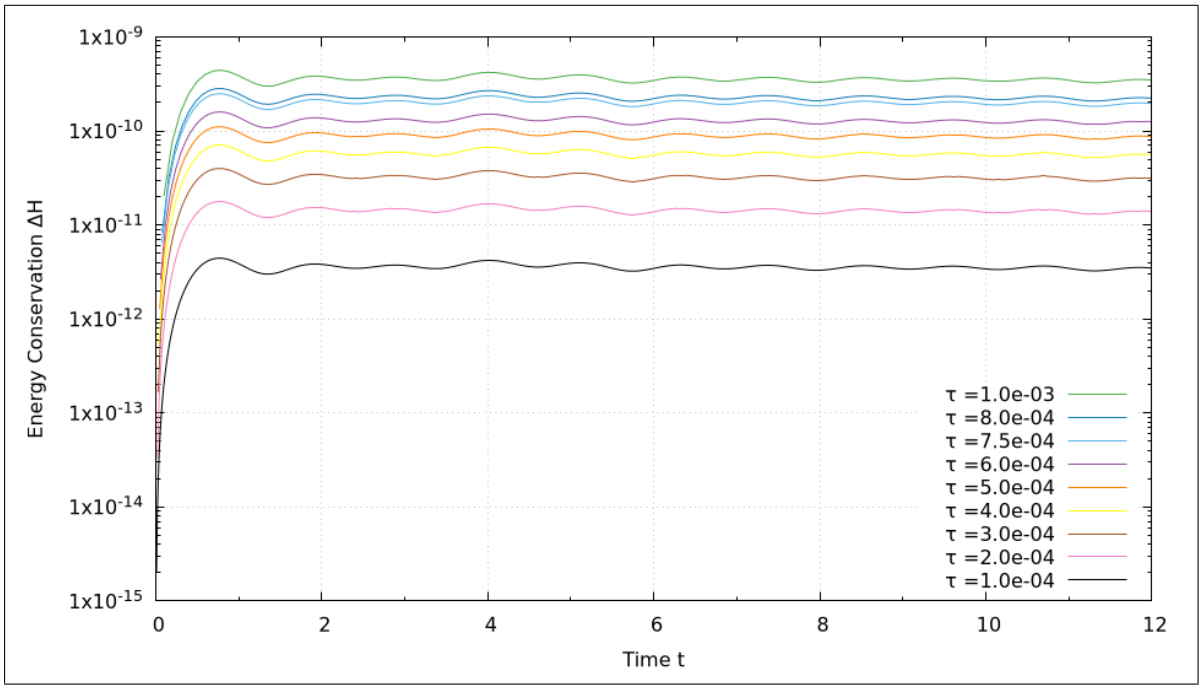

Figure 3: energy difference from t=0 and at time t versus the time with the leapfrog integrator

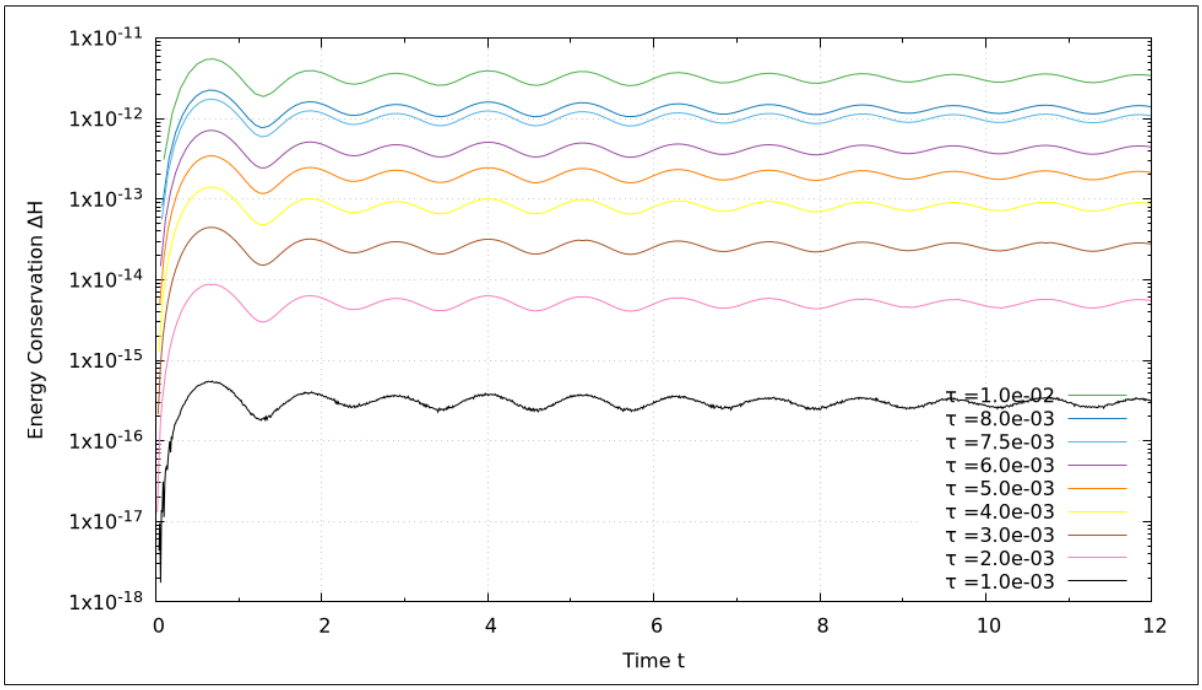

Figure 4: energy difference from t=0 and at time t versus the time with the yoshida integrator

In fig. [3] and [4] we calculated the energy difference of the field at time 0 and time t for a field with linear size $N = 21$ and several values for τ. We see that the yoshida integrator needs only a value of $\tau = 1e-3$ such that we get a good approximation. The Leapfrog integrators needs a τ magnitudes lower than that to achieve this.

In fig. [5] we tried to find out how to choose N and τ such that we obtain a good approximation to the continuous case. It turns out that the higher N is the smaller the energy difference. Such that for a given N once we found a suitable τ, we can choose any smaller τ and we would still recover the continuous case. Vice versa with given τ and the choice for N, as with increasing N the observable values start to converge to certain values. This can be seen in fig. [6] more clearly.

We decided on $N = 401$ and $\tau = 5{,}0 \times 10^{-3}$. The Question is to which decimal place do you want to have the observables precise. In fig [7] you can see the initial condition for those values. Inside the `./data/figs/` folder is a animation for this case.

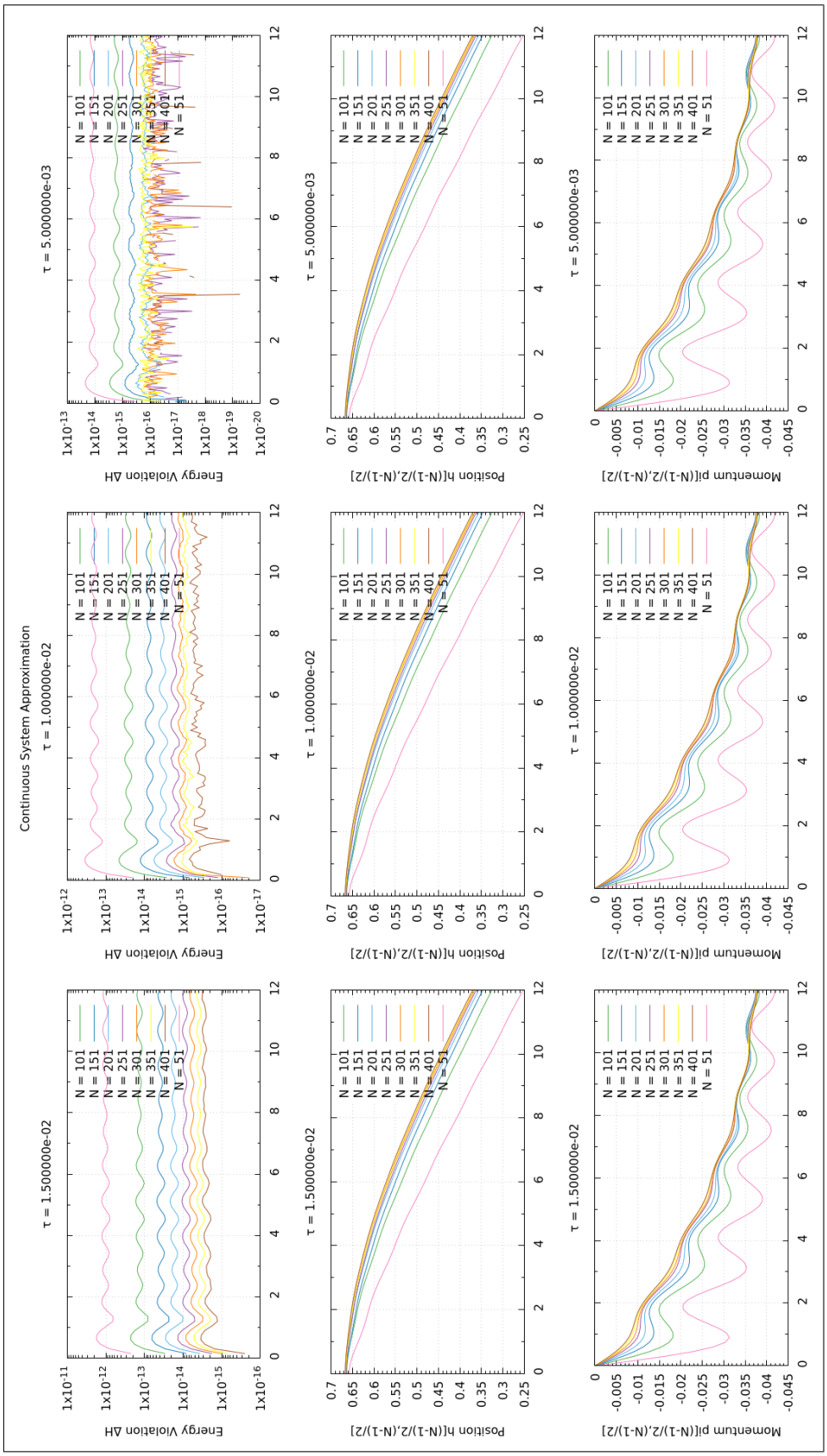

Figure 5: for different values of tau each graph shows the observables for multiple N's with yoshida integrator

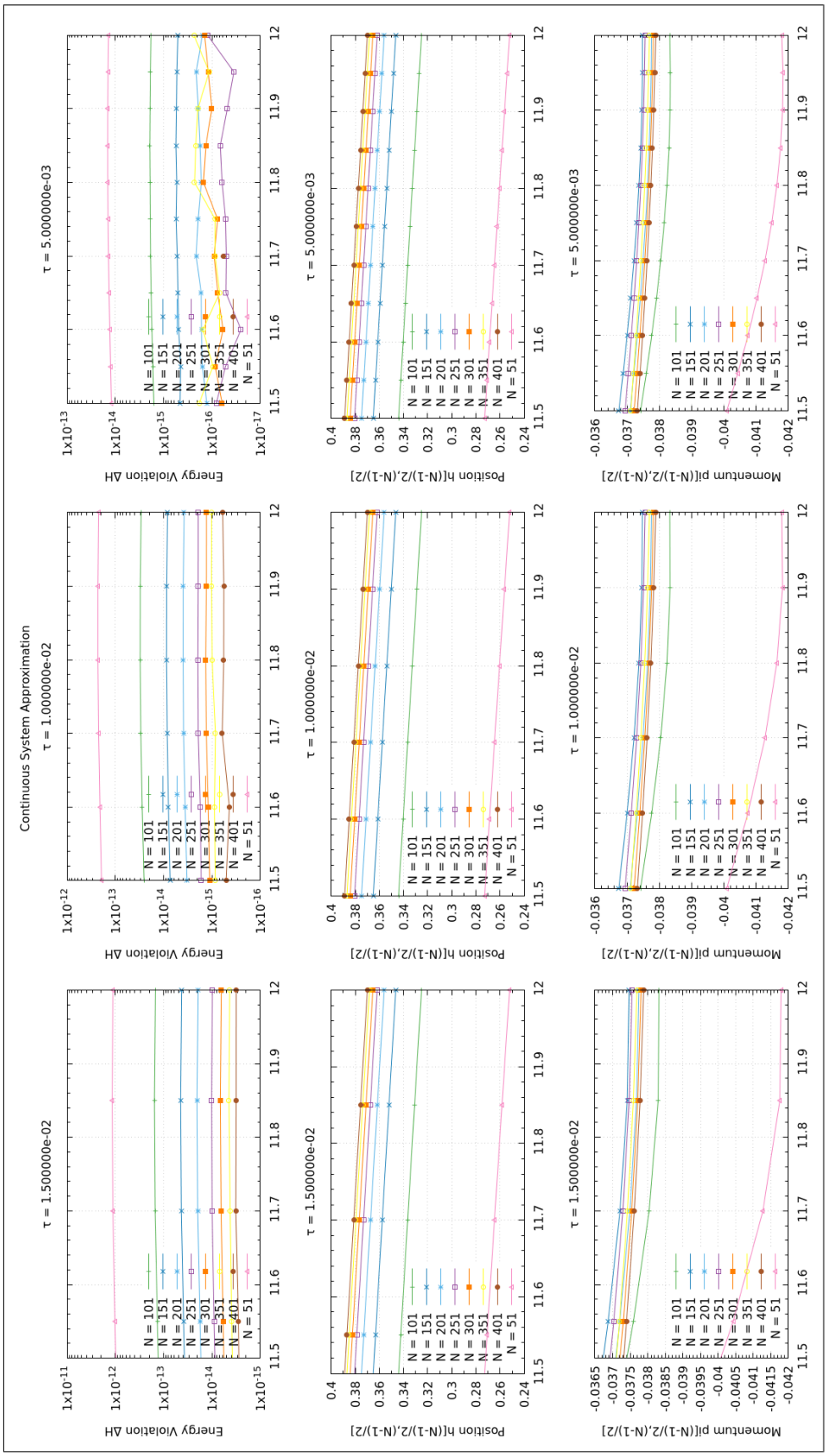

Figure 6: for different values of tau each graph shows the observables for multiple N's with yoshida integrator - zoomed in

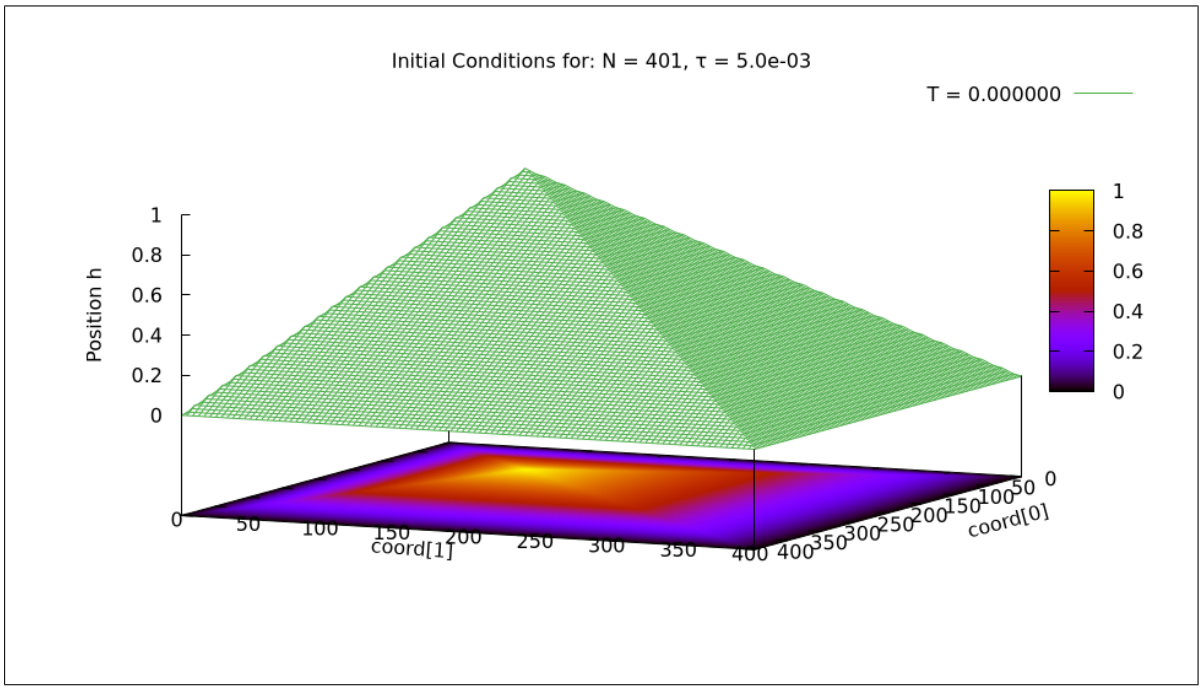

Figure 7: Initial condition of the field. We pull the membrane and this gives approximatively a pyramid

9 Extension to more than two dimensions

Except of the animation one might try to find similar analysis for higher dimensions. The whole program is written with a variable dimension. Only the test for the `is_border()` function and the pyramid starting condition are only working for 2-D.