Version: November 9, 2017

Practical Course Reader for Control of distributed systems

made at the

University of California at Berkeley

BY **Arnaud de LA FORTELLE**

visiting professor from MINES ParisTech

---

for the course CE-C291F — EE-C291 — ME-C236

With the help of:

Bradley Cage	student
Hongbei Chen	student
Yue Hu	student
Carlin Liao	student
Xin Peng	student
Robert Ruigrok	student
Lin Yang	student
Qingan Zhao	student
Ruitong Zhu	student

Contents

Chapter 1

Introduction

1.1 Objectives

This practical course reader aims at giving good advice to students realizing a project. Starting with the theoretical knowledge contained in the reader of Prof. Alexandre Bayen, students have to realize a project based on a distributed parameter system that they should:

1. describe
2. model
3. explain

The first part is a descriptive (i.e. qualitative) description of the system to be studied, its interest and the goal of the study. The second part is an elaboration, under simplifications, of a quantitative model (meaning here some kind of Partial Differential Equation and some control or optimization). The third part is an analysis of this model, either by solving the equations, by simulation or by a combination of both, using the tools learned during the course. It should lead to a better understanding of how the system works and how it can be optimized or controlled to fulfill the goals set in the description.

Therefore this practical course reader shows examples of modeling and various analysis methods that can support the creativity of the students realizing a project.

1.2 System description

A system is, according to Cambridge dictionary "a set of connected things or devices that operate together". There is no recipe to describe a system.

However, it is often useful, instead of having an *analytical* description (i.e. tear the system into pieces), to have a *synthetic* one (i.e. seeking for a global view). Without any generalization, again, one can look after the following 4 views of a system:

1. objective
2. environment
3. actions
4. transformations

One can very arguably contest these words, but remember they are only guidelines. In most systems, since they are man-made, there is a goal. This *objective* is important to describe since it contains the meaning: what is it for? Note this is quite often the opposite of the analytical view: how does it work? However, in our case, this goal contains the information about the *objective function* for optimization or for control. The *environment* dictates what the system has to do: a system is never isolated and its goal is often closely related to its environment. At least, the *actions* a system perform are partially internal and partially external, so that taking into account the environment allow understanding of disturbances but also the objective. Since a system is usually complex, it may evolve (i.e. experience *transformations*) according to actions and in line with the objective. One can also think of transformation as internal states of a finite automaton.

In any case, a description is intended at telling the whole story to other people and this is why it is highly recommended to use only words, no formula, but drawings can well replace a thousand words.

1.3 System modeling

While a system description is intended to be purely qualitative, modeling aims at estimating some quantities. Very clearly, the choice of the quantities is decisive. Important quantities can be derived from the system description, such as the objective function (what to maximize or minimize?), the transformations (internal states) or environment variables.

There are many ways to measure quantities, e.g. numbers of vehicles in a microscopic view may be equivalent to measuring densities of vehicle at a macroscopic scale. Unfortunately there is no algorithm to produce significant model and this is a knowledge that needs practice. However, in our present case, namely partial differential equations modeling, there is very often a

kind of scaling that is necessary in order to produce continuous-space and continuous time variables. Since this reader aims at being practical, the reader is referred to Chapter 2 for modeling examples.

One can (too?) quickly summarize some usual quantities of interest:

- macroscopic quantities (temperature, density, deformation...);
- variables (state space, time...) and parameters including the control;
- optimization criteria;
- invariant quantities (mass, energy, momentum...) and constraints;
- initial and boundary conditions.

1.4 System explanation

What is called here explanation has been also called "solution" during the course. It consists mainly in studying the model under various views and to produce new knowledge. This usually implies simulations, analytical analysis, optimization and visualization; all previous methods can be compared or combined in order to give more insight of the model.

1.4.1 Analysis

Analysis is important for the concepts it carries. We have learned a few methods to exactly solve or to transform the PDEs:

1. Dimensional analysis allow to build interesting quantities that should remain constant;
2. Product-form solutions leads to decomposition of solutions (often in the type of Fourrier transform) with knowledge of the modes and their frequencies (spatial and temporal).
3. the method of characteristics is a powerful methods to build solutions, even discontinuous like shockwaves in LWR equations. The characteristic curves are interesting features.

In the present course, students should perform some initial analytical analysis, even if it does not lead to a complete solution, in order to exhibits interesting quantities as mentioned above. These quantities should be used as guidelines as far as possible for further study.

1.4.2 Simulation

Simulation is interesting for all the data it brings. However there are many ways to simulate a system. At least we can mention:

- Microscopic simulation — often agent-based — where a lot of microscopic entities evolves with their dynamics; This method is relevant when the PDE is a macroscopic view of the system, i.e. when it is derived from a scaling;
- Finite differences schemes, that is often a direct implementation of the dynamics underlying the PDE (differences apply on a discretization of space and time);
- Finite elements is usually associated to computation of eigenvalues and eigenvectors, i.e. to product-form solutions and the computation of modes; in this case, discretization is made through a projection of the functions onto a finite (functional) basis.

In this course, students have to produce simulations. The goal is to exploit the data to visualize the system evolution (see also Section 1.4.4) and get a better understanding. One of the main problem of simulation is to have a clear idea of the quantities to be displayed, be it statistics, metrics or functions. This is why a good prior analysis (see also Section 1.4.1) is a good idea. A must in simulation is to compare quantities computed from the output of the simulation with similar quantities computed from theoretical analysis: e.g. number of vehicles per km in micro-simulation with densities in LWR equations.

1.4.3 Optimization or control

The goal of this course is to learn how a system can be controlled or optimized. This requires to have a good description of the system in order to have clear ideas of what is to be optimized or controlled. Usually, the design of the control is free and this is where students can be creative.

There is a very common distinction between 2 close concepts: planning and control. In most real systems, the planning phase is responsible for building suitable a path, according to the dynamics of the system and to the environment; then, a control loop ensures the systems follows this path by using actuators (the controls) and taking care of disturbances (in sensors and in actuators). Planning is open-loop, usually with a simple model (deterministic...) but with a complex cost function that expresses the desirable

behavior. Control is closed-loop, designed to be robust, taking into account noises and disturbances, with a simple cost function: follow the plan.

Techniques are very diverse and really depend on the system and the kind of control. Examples are given in Chapter 4.

1.4.4 Visualization

The visualization aspect of a study may be the most important yet the most challenging. Indeed, a drawing is extremely instructive, provided it is well explained. But the beauty can be misleading and the students have to take great care of the quantities they illustrate: What should be seen in the drawing? What does it tell us? Does it compare well with existing knowledge or not?

Therefore it is advised to start quickly (very early with simulation data) to try to see something and to continuously improve the visualization aspect. It is important for explaining the results as well as for enhancing the students' own understanding (and debugging the codes, very often).

Chapter 2

Modeling

2.1 Objectives

The modeling of a system starts with a good description (see Section 1.2). Very often, the system can is made of numerous elements (e.g. cars, particles...) or can be decomposed into small chunks (string, heated body...). In this course we want to obtain PDEs: this means that the dynamics is local. Small elements influence only their neighbors through quantities that are very important: tension and curvature for a string, density and flow for traffic... In order to obtain PDEs, it is often necessary to assume such quantities are slowly varying and that one can *scale* the system. Moreover, one should keep only a very limited number of quantities and simplifying assumptions are key to that objective (neglect gravity, torsion, viscosity...). Finally, an equation (even after scaling) is always obtain as limit of another equation: it is important to explain which principles (physical or other) lead to these equations, be it a conservation equation or dynamical equations.

Now, a model is not complete with only a PDE: initial and boundary conditions must be explicit, the control parameter (or the space of choice) must be given as well as the optimization criteria. The following examples illustrate this method.

2.2 Derivation of partial differential equations

This section shows how one can describe some simple systems and derive from physical principles the PDEs modeling their behavior.

2.2.1 2D wave equation

Prepared by subgroup 1 (Bradley Cage and Lin Yang)

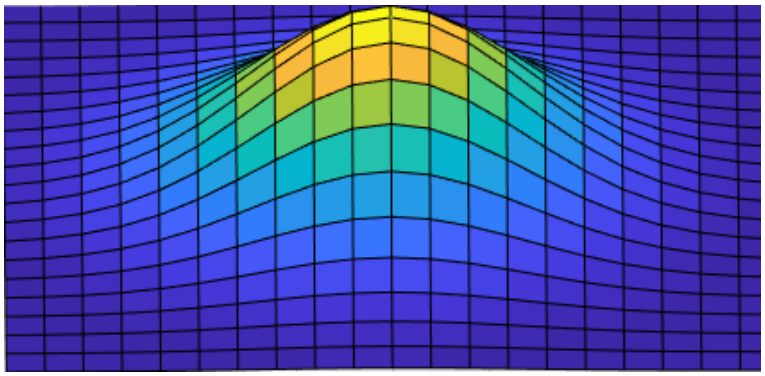

Fig. 2.1: The membrane system

Description Our system is comprised of a flexible membrane stretched to some shape, with all of its edges fixed in place. The desired goal is to understand the vertical position of the various points on the membrane over time. The membrane in this system has vertical deflections which are small compared to its overall size, and deflections happen only in the vertical direction.

This 2D system is a continuation of the 1D wave equation, and is a natural precursor to the 3D wave case.

Model Assumptions:

- Membrane has uniform planar density ρ
- The tension per unit length, F_t, caused by stretching the membrane is the same at all points and in all directions and does not change during the motion
- Vertical position is given by some function $u(x, y, t)$

We begin from basic principles.

$$\Sigma F = m\vec{a}$$

Taking some small section of the membrane dx by dy, we can replace mass and acceleration and since we know density and that $\vec{a}$ is the second derivative of position with respect to time, thus enabling us to rewrite the equation.

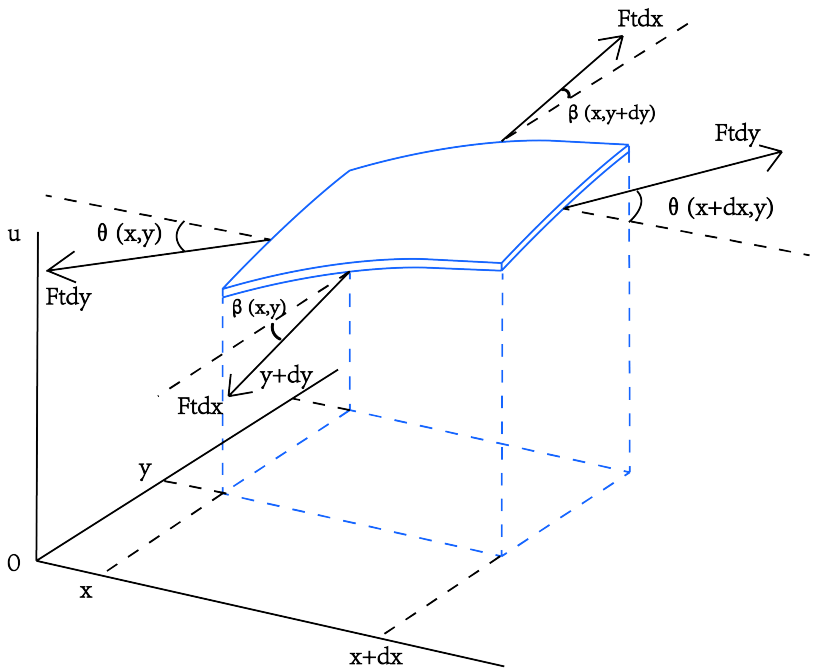

Fig. 2.2: The force analysis of a small section of the membrane system

$$\Sigma F = \rho dx dy \frac{\partial^2 u}{\partial t^2} \tag{1}$$

Performing a force balance on the section of membrane in the x and y directions gives us tensions at each on each side, then resolved to their vertical components. Remember that since tension is constant per unit length, we must multiply the force acting on each side by the length of that side. Thus, the force acting on this balance lets us rewrite ΣF (that is we only consider vertical forces, forces acting in the $x-u$ and $y-u$ planes):

$$\Sigma F = F_x + F_y$$

$$F_x = F_t dy \Big[\sin\big(\theta(x+dx,y,t)\big) - \sin\big((\theta(x,y,t)\big) \Big]$$

$$F_y = F_t dx \Big[\sin\big(\beta(x,y+dy,t)\big) - \sin\big((\beta(x,y,t)\big) \Big]$$

We can confidently use the small angle approximation sin in the x direction

$$\sin(\theta) \approx \tan(\theta) = \frac{\partial u}{\partial x} = u_x$$

and likewise in the y direction

$$\sin(\beta) \approx \tan(\beta) = \frac{\partial u}{\partial y} = u_y$$

to get our equations into the form

$$F_x = F_t dy\Big[u_x(x+dx,y,t) - u_x(x,y,t)\Big]$$

$$F_y = F_t dx\Big[u_y(x,y+dy,t) - u_y(x,y,t)\Big]$$

From there we can sum these forces and plug them back in to equation 1

$$\rho dxdy\frac{\partial^2 u}{\partial t^2} = F_t\Big[dy\Big[u_x(x+dx,y,t)-u_x(x,y,t)\Big]+dx\Big[u_y(x,y+dy,t)-u_y(x,y,t)\Big]\Big]$$

We then divide by dx and dy and take the limit as $dx, dy \to 0$:

$$\rho\frac{\partial^2 u}{\partial t^2} = \lim_{dx,dy\to 0} F_t\left[\frac{u_x(x+dx,y,t) - u_x(x,y,t)}{dx}+\frac{u_y(x,y+dy,t) - u_y(x,y,t)}{dy}\right]$$

We recognize that we now have derivatives in the form of difference quotients, and can take the partial derivative of each one (since u is a function of multiple variables)

$$\rho\frac{\partial^2 u}{\partial t^2} = F_t\left[\frac{\partial}{\partial x}u_x + \frac{\partial}{\partial y}u_y\right] = F_t\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right] \tag{2}$$

Dividing over the uniform tension, we reach our final form.

$$\frac{\rho}{F_t}\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \tag{3}$$

We can adhere to standard conventions and write our final 2D wave equation as

$$a^2\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \qquad a = \sqrt{\frac{\rho}{F_t}} \tag{4}$$

Equation 4 is also commonly written using the Laplace operator:

$$a^2\frac{\partial^2 u}{\partial t^2} = \nabla^2 u \tag{5}$$

2.2.2 2D heat diffusion

Prepared by subgroup 2 (Qingan Zhao and Ruitong Zhu)

Description The aim of this part is to describe and model a PDE that describes temperature dynamics in a two-dimensional body via heat conduction. Basically, heat conduction is the exchange of heat from regions of higher temperatures into regions with lower temperatures, which varies in the transfer rate for different materials.

Consider a thin flat body with a constant thickness h and uniform density ρ'. Assume that the faces of the thin body are in perfect insulation, which means there is no heat flow travel in the out-of-plane direction of the body. Hence, heat can only flow in the direction within the plane of the body, which turns into a two-dimensional problem. Then a two-dimensional coordinate system is established such that each point of the body can be described with a coordinate (x, y). Then the (2D-uniform) density of the body is $\rho = \rho' h$. Denote the temperature function of each point by T so that the temperature of the body at position (x, y) and time t are described as $T(x, y, t)$, as shown in Figure 2.3. The goal is to derive $T(x, y, t)$ when there is no internal heat source.

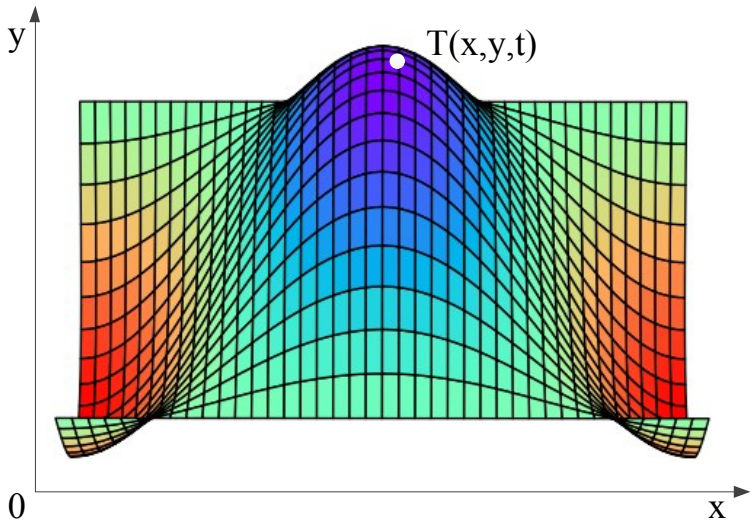

Fig. 2.3: System description in 2 dimensions

Model Consider a small rectangular element of the body with vertices (x, y), $(x + \mathrm{d}x, y)$, $(x, y + \mathrm{d}y)$, and $(x + \mathrm{d}x, y + \mathrm{d}y)$. The heat flows are shown

in Figure 2.4.

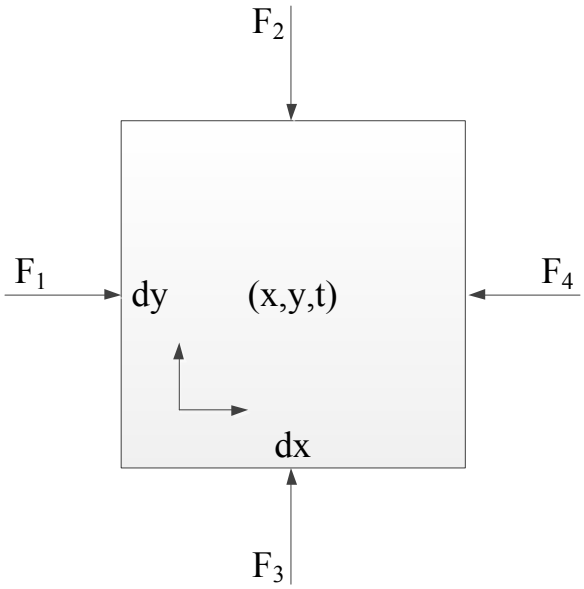

Fig. 2.4: Heat flows in a small rectangular element of the body

The heat amount Q (i.e the thermal energy) of the rectangular element at time t is:

$$Q(x, y, t) = CmT(x, y, t) \tag{6}$$

where C is called *heat capacity*, which is a supposed to be constant (assuming the material is uniform and temperature do not vary too much); $m = \rho A$ is the mass of the rectangular element where A its surface.

The rate of thermal energy change with respect to time is therefore:

$$\frac{\partial Q}{\partial t} = C\rho \mathrm{d}x\mathrm{d}y\frac{\partial T}{\partial t} \tag{7}$$

As shown in Figure 2.4, the incoming flow is $F_1 + F_2 + F_3 + F_4$. Denote the heat flux $\vec{q}$ in horizontal and vertical directions by q_x and q_y, then we have:

$$F_1 = q_x(x, y, t)\mathrm{d}y \tag{8}$$
$$F_2 = -q_y(x, y + \mathrm{d}y, t)\mathrm{d}x \tag{9}$$
$$F_3 = q_y(x, y, t)\mathrm{d}x \tag{10}$$
$$F_4 = -q_x(x + \mathrm{d}x, y, t)\mathrm{d}y \tag{11}$$

Now, we know that according to energy conservation, the thermal energy variation of any small element (as in Equation (7)) is equal to the total

incoming heat flow. By putting the partial flows as in Equations (8)-(11), this conservation principle yields:

$$C\rho \mathrm{d}x\mathrm{d}y\frac{\partial T}{\partial t} = \mathrm{d}y[q_x(x,y,t) - q_x(x+\mathrm{d}x,y,t)] + \mathrm{d}xh[q_y(x,y,t) - q_y(x,y+\mathrm{d}y,t)] \tag{12}$$

Now, another physical principle, *Fourier's Law*, states that the heat flow is (negatively) proportional to the gradient of temperature:

$$\vec{q} = -k\nabla T \tag{13}$$

where k is known as the thermal conductivity of the material (also considered as a constant). Then q_x and q_y are expressed as:

$$\begin{aligned} q_x &= -k\frac{\partial T}{\partial x} \\ q_y &= -k\frac{\partial T}{\partial y} \end{aligned} \tag{14}$$

Hence, Equation (12) can be written as:

$$\frac{\partial Q}{\partial t} = k\mathrm{d}yh\left(\frac{\partial T(x+\mathrm{d}x,y,t)}{\partial x} - \frac{\partial T(x,y,t)}{\partial x}\right) + k\mathrm{d}xh\left(\frac{\partial T(x,y+\mathrm{d}y,t)}{\partial y} - \frac{\partial T(x,y,t)}{\partial y}\right) \tag{15}$$

Combine Equation (7) and (15):

$$\frac{\partial T(x,y,t)}{\partial t} = \frac{k}{c\rho}\left(\frac{\partial^2 T(x,y,t)}{\partial x^2} + \frac{\partial^2 T(x,y,t)}{\partial y^2}\right) \tag{16}$$

Denote $k/c\rho$ by a^2, and the two-dimensional heat equation can be drawn:

$$\frac{\partial T}{\partial t} = a^2\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) \tag{17}$$

To solve the PDE, we also need to specify initial conditions and boundary conditions. For initial conditions, assume that in domain Ω:

$$T(x,y,0) = \phi(x,y) \tag{18}$$

As for boundary conditions, it could be devided into two different situations according to the domain Ω. If $\Omega = \mathbb{R}^2$, we don't have to worry about the boundary, which means the equation can be solved soly based on the initial condition defined on $\mathbb{R}^2$. But if there is a boudary $\partial\Omega$, in this case, the boundary conditions can be described as:

$$T(x,y,t)|_{(x,y)\in\partial\Omega} = g(x,y,t) \tag{19}$$

2.3 Manipulation of partial differential equations

2.3.1 Weak formulation

Prepared by subgroup 3 (???)

Objective

Demonstration

$u = 0 \Delta u = \partial^2 \frac{u}{x^2} + \partial^2 \frac{u}{y^2} = 0 \iint_D (\frac{\partial u}{\partial x} - \frac{\partial u}{\partial y}) dxdy = \oint_L Pdx + QdyP = -\frac{\partial u}{\partial y}, Q = \frac{\partial u}{\partial x} \oint_L Pdx +$

Chapter 3

Simulation

3.1 Objectives

The objective is to obtain numerical data (often the successive states of the system), to visualize these data, to build meaningful statistics and finally to link the results with theoretical considerations. Best would be to compare simulations with analytical results, but this may prove infeasible and then more analysis is required to compare theory and simulations.

3.2 Simulation examples

3.2.1 Forward Euler simulation of the 1D heat equation

Prepared by subgroup 1 (Lin Yang and Bradley Cage)

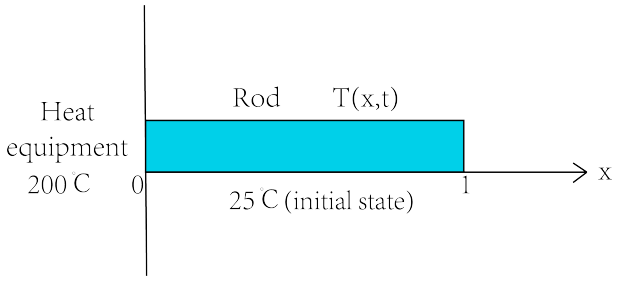

Fig. 3.1: The heated rod system, with the temperature at any point given by $u(x,t)$

Model presentation Our system is comprised of a rod fixed to a heated wall kept at constant temperature. The length of the rod is arbitrarily taken as 1m. The initial temperature of the rod is in equilibrium with the ambient temperature at 25°C and the temperature of the wall is 200°C. We represent temperature at a point on the rod at some time with the function $u(x,t)$. Note that the rod is thin with respect to its length, thus is modeled as a 1D system, that is, temperature is solely a function of rod position and time. The goal of this simulation is to show the variations in temperature of various points on the rod over time. //

We take our initial state as:

- $T(0,t) = 200°\text{C}$
- $T(x,t) = 25°\text{C},\ x \neq 0$

The partial differential equation of the 1D heat propagation:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} \tag{1}$$

We explicitly state that $u(\dot{})$ is a function of x and t

$$\frac{\partial}{\partial t} u(x,t) = \alpha \frac{\partial^2}{\partial x^2} u(x,t) \tag{2}$$

We use a finite difference approximation to get compute the derivatives in space and time

$$\frac{u_i^{N+1} - u_i^N}{\Delta t} = \alpha \frac{u_{i+1}^N - 2u_i^N + u_{i-1}^N}{\Delta x^2} \tag{3}$$

Rearranging we reach the final form we need for our Forward Euler approximation. Note that the quantity $\alpha \frac{\Delta t}{\Delta x^2}$ is Fourier's number, with α being the thermal diffusivity of a material. In the following simulations, we arbitrarily choose Aluminium with $\alpha = 9.7e-5$.

$$u_i^{N+1} = u_i^N + \alpha \frac{\Delta t}{\Delta x^2} \left[u_{i+1}^N - 2u_i^N + u_{i-1}^N \right] \tag{4}$$

Implementation We provide an implementation in Matlab.

Code Listing 3.1: Forward Euler and Plotting in MATLAB

```
% Values arbitrarily chosen. It's a useful exercise to vary these
T_final = 300;
N_t = 2000;
X_final = 1;
```

```
N_x = 100;

% Calculate time and x steps based on sampling size and # of samples
T = linspace(0, T_final, N_t+1);
X = linspace(0, X_final, N_x+1);
dt = T(2) - T(1);  % Calculate delta t
dx = X(2) - X(1);  % Calculate delta x

alpha = 9.7e-5; % Thermal diffusivity of Aluminium in m^2/s
Fo = (alpha*dt)/(dx^2); % Fouriers number = diffusive transport rate/st

% Define your initial condition here. This could be some function IC(x)
%  however for simplicity's sake we take a rod with a uniform temperatu
%  and in contact with a hot plate at one end
wall_temp = 200;
init_temp = 25;

% Initialize the N state and the N-1 state
u_old = zeros(1, N_x+1);
u_old(:) = init_temp;
u_old(1) = wall_temp;
u_cur = u_old;
u_plot = u_old;

for t = 1:N_t
    for i = 2:N_x
        % Forward Euler solution to heat equation
        u_cur(i) = Fo*(u_old(i+1) - 2*u_old(i) + u_old(i-1)) + u_old(i);
    end
    u_cur(1) = wall_temp; % Set the left boundary to be our high of 200
    u_old(:) = u_cur; % We move to the next time step, reset N-1 state
    u_plot = [u_plot; u_cur];
end

[X_plot,T_plot] = meshgrid(X,T); % Create 2D meshgrid to create surface

surf(X_plot,T_plot,u_plot,'EdgeColor','none') % Create surface plot
c = colorbar; % Attach colour bar and create scale
c.Label.String = 'Temperature [C]';

xlim([0 1]) % Add axis limits
```

```
xlabel('Distance along rod [m]'); % Add descriptive axis labels
ylabel('Time [s]');
zlabel('Temperature [C]')
```

Results We can create plots of the rods temperature as a function of time and position. This gives us insight into how the system evolves as we maintain our constant temperature.

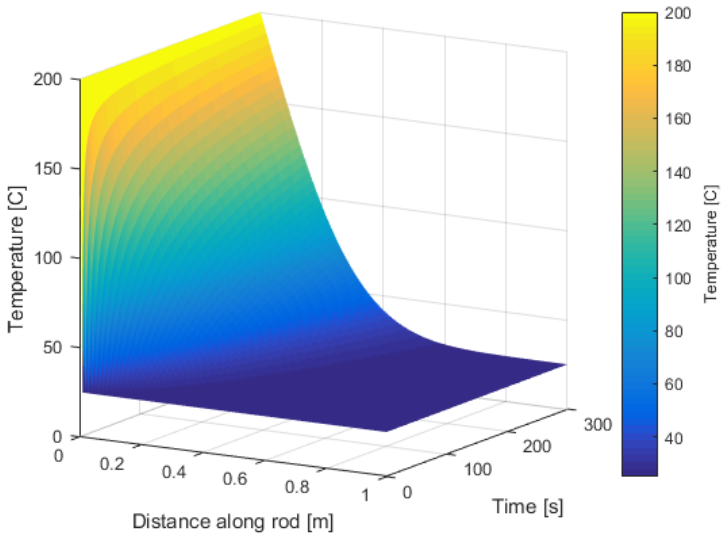

Fig. 3.2: Temperature variation for the whole rod over 300 seconds

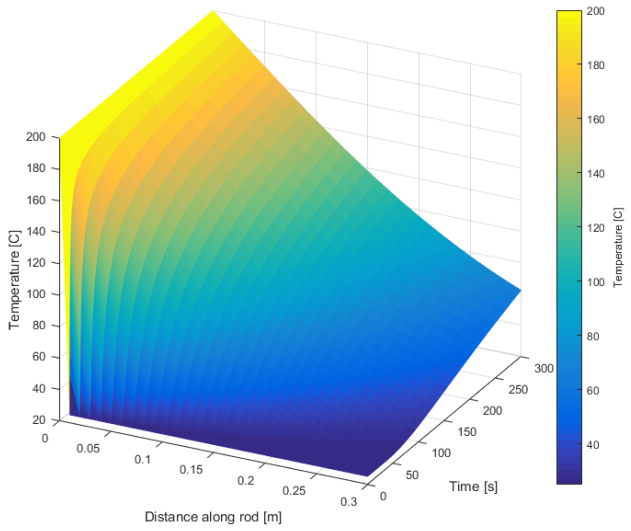

Fig. 3.3: Temperature variation for the rod segment with x varies from 0 to 0.3m over 300 seconds

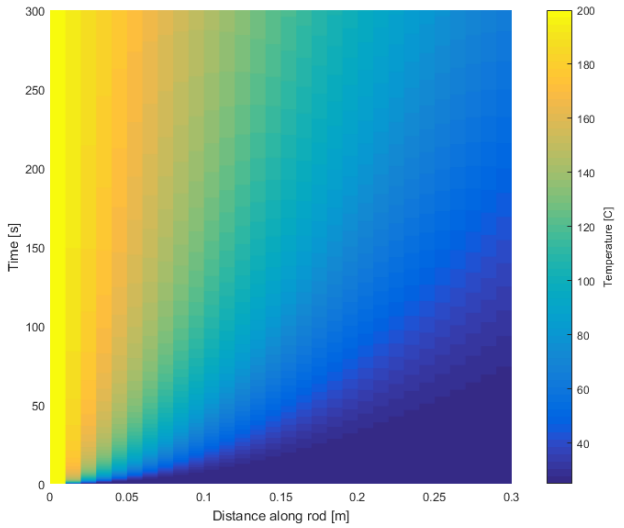

Fig. 3.4: Top view of the temperature variation for the whole rod over 300 seconds

Interpretation The results we obtain are consistent with our model and the physical characteristics. We can see that from the visualization the temperature falls off as $\frac{1}{x^2}$, which is consitent with the theoretical definition of the Fourier number as described above. We can see pathlines of the heat front evolve quadratically over time, especially when looking at the system from above.

Conclusion Here we have learned that in examining systems such as the heated rod, we can safely analyze it as a 1D system. We were able to discretize the partial differential equations and apply a forward Euler simluation to yield phyiscally relevant and meaningful simulations. Through the Matlab visualization, we gain a better understanding of how heat is moving through the rod and the system. From the resources and analysis provided, it is trivial to create other iinitial/boundary conditions and repeat the simulations to garner a deeper insight into the heat behaviour. This is left as an exercise to the reader.

3.2.2 1D vibrating string simulation

Prepared by subgroup 2 (Qingan Zhao and Ruitong Zhu)

Model presentation What is the model we want to simulate? What do we want to observe? Which is the state space and the dynamics?

Implementation Explain the structure of the code. Do not put necessarily all the code (not more than 100 lines) since some routines (functions) can hide efficiently some unnecessary complexity. Provide a code that run (and explicit librairies and dependencies). Ensure your file name is aligned with this part.

Results Explain the quantities you are studying (i.e. metrics and statistics). Provide good visualization.

Interpretation Relate these quantities to the model and to theoretical knowledge of the course.

Conclusion What have we learned? Is everything aligned (theory and practice)? What was difficult? Provide perspectives.

This code aims to simulate the vibration string (i.e., wave equation) in one dimension. The partial differential equation (PDE) of this problem is given as follow:

$$\frac{\partial^2 h}{\partial t^2} = a^2 \left(\frac{\partial^2 h}{\partial x^2} \right) \tag{5}$$

where h is the wave function of each point so that the displacement of the string at position x and time t are described as $h(x,t)$; a is the wave speed which equals to $(E/\rho)^{0.5}$.

The simluation is based on finite differences method (FDM). Assume the wave speed a equals to 1; the length of the string (constrained at both ends) is 2; the maxium time for this simulation is 4; stepsize dx and dt are both equal to 0.01. The initial condition of shape and speed are described as follows:

$$h(x,0) = sin(\pi x) \tag{6}$$

$$\left.\frac{\partial h}{\partial t}\right|_{(x,0)} = 0 \tag{7}$$

The boundary conditions are described as follows:

$$h(0,t) = h(2,t) = 0 \tag{8}$$

Here is the code:

```
import numpy as np
import math
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import pyplot as plt
from matplotlib import rc
plt.rc('text', usetex=True)
plt.rc('font', family='serif')

## parameter
a = 1  ## a coefficient of stiffness
L = 2  ## The string is constrained at x=0 and x=L.
T = 4  ## maxium time for this simulation.
dx = 0.01  ## time step
dt = 0.01  ## distance step
N = int(L / dx);
M = int(T / dt);
r = (a * dt / dx) ** 2  ## a parameter
```

```
## initial shape of the string
def initial(x):
tmp = math.sin(math.pi * x)
return tmp

## initial speed of the string
def diff(x):
tmp = 0 * x
return tmp

## Define an array and a blank matrix for later use.
x = [0]
h = np.zeros((M + 1, N + 1))

## t=0
for i in range(N):
x.append(x[i] + dx)  ## x axis
h[0, i + 1] = initial(x[i + 1])  ## displacement of the
                                 string

## t=dt
for i in range(N - 1):
h[1, i + 1] = h[0, i + 1] + r * (h[0, i] + h[0, i + 2] - 2
                                 * h[0, i + 1]) / 2 + dx * diff
                                 (x[i + 1])
## displacement of the string

## t=n*dt n>1
for j in range(1, M):
for i in range(N - 1):
```

```
    h[j + 1, i + 1] = (h[j, i + 2] + h[j, i] - 2 * h[j, i + 1])
                                    * r - h[j - 1, i + 1] + 2 * h
                                   [j, i + 1]
    ## displacement of the string

    t = [0]
    for j in range(M):
    t.append(t[j] + dt)  ## t axis
```

Plot the 3D graph (x, t, h):

```
    ## Plot the 3D figure.
    fig = plt.figure()
    ax = Axes3D(fig)
    X, T= np.meshgrid(x,t)
    ax.plot_surface(X, T, h, cmap='rainbow')
    plt.title(u'Vibrating String',fontsize=14)
    ax.set_xlabel('X')
    ax.set_ylabel('T')
    ax.set_zlabel('h')
    plt.show()
```

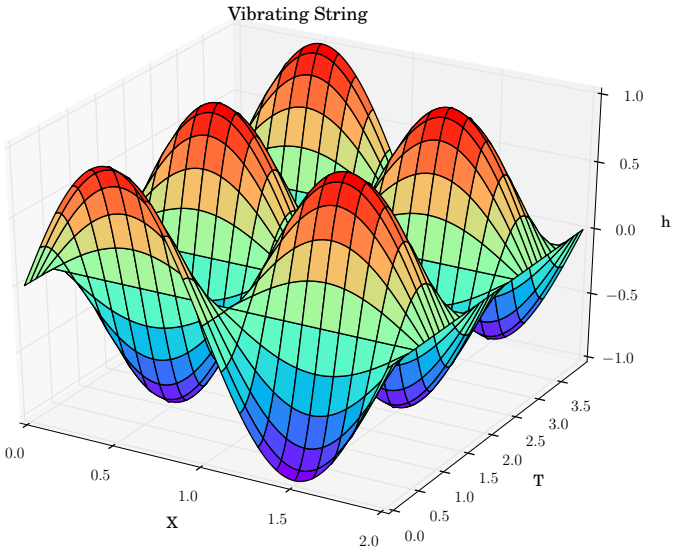

Fig. 3.5: 3D graph of the simulation

Plot the shape of the string when $t = \{0, 0.5, 1, 1.5, 2\}$:

```
for i in range(5):
plt.scatter(x,h[i*50,],label="t="+str(i*0.5))
plt.title(u'Shape of the String',fontsize=14)
plt.xlabel(u'x',fontsize=14)
plt.ylabel(u'h',fontsize=14)
plt.legend(loc='upper left')
plt.show()
```

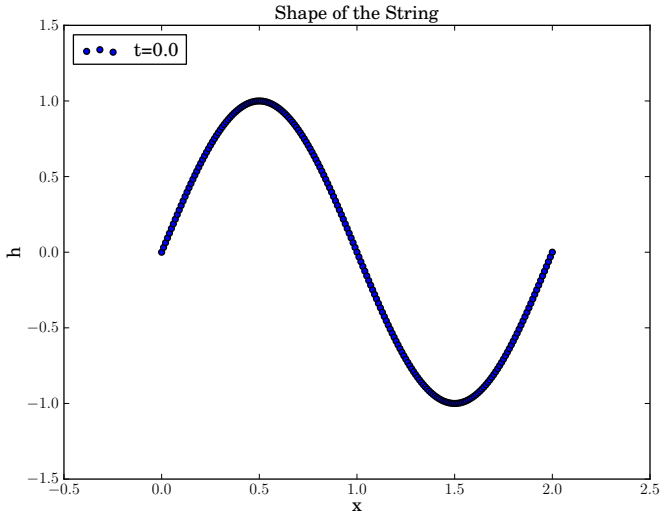

Fig. 3.6: Shape of the string when t=0

3.2.3 1D traffic simulation

Prepared by subgroup 3 (Yue Hu and Carlin Yao)

Model presentation What is the model we want to simulate? What do we want to observe? Which is the state space and the dynamics?

Implementation Explain the structure of the code. Do not put necessarily all the code (not more than 100 lines) since some routines (functions) can hide efficiently some unnecessary complexity. Provide a code that run (and explicit librairies and dependencies). Ensure your file name is aligned with this part.

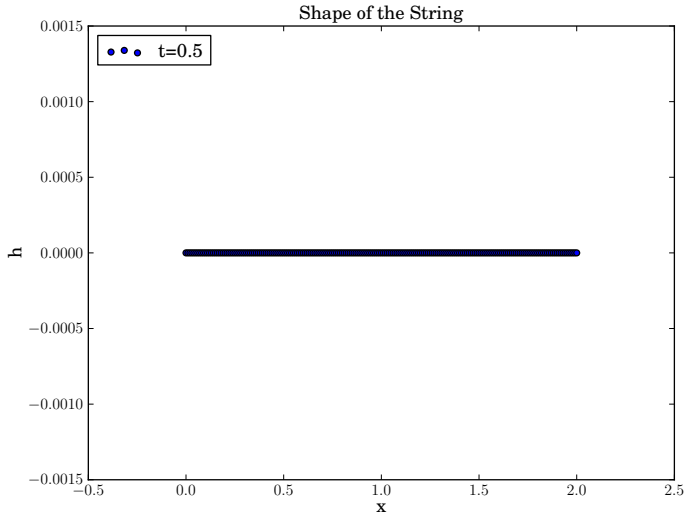

Fig. 3.7: Shape of the string when t=0.5

Results Explain the quantities you are studying (i.e. metrics and statistics). Provide good visualization.

Interpretation Relate these quantities to the model and to theoretical knowledge of the course.

Conclusion What have we learned? Is everything aligned (theory and practice)? What was difficult? Provide perspectives.

3.2.4 Random walk in 2D simulation

Prepared by subgroup 4 (Yue Hu, Carlin Liao and Robert Ruigrok)

Model presentation In this example we simulate the random walk of a particle in a 2D space. A random walk is a mathematical object, known as a stochastic or random process, that describes a path that consists of a succession of random steps. In order to simulate this process, we let a particle move over a discretized grid where its motion is drawn from a set of possible directions. In this simulation we are interested in finding expected distribution of particles after a certain number of time steps as well as the position where they hit the boundaries of the spatial grid.

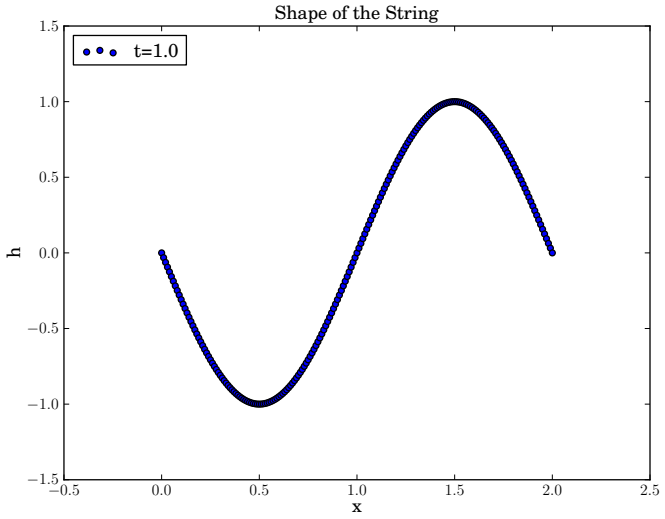

Fig. 3.8: Shape of the string when t=1

A particle starts at a specified initial position, from where it begins moving through the grid. In our code, we used a coordinate system to represent the location of a particle as provided in figure 3.2.4. The outer border of the grid is enclosed by a "wall". When the particle hits this wall, its motion stops and the location where it makes contact is registered.

The dynamics of a particle are relatively straightforward and can be described by equation 9. A particle has 5 options for its motion: moving up, right, down, left or no motion (options are depicted in figure 3.2.4). Every motion has a certain probability p to occur. These probabilities can be given as input and must add up to 1.

$$X_{k+1}(x,y) = X_k(x,y) + \begin{cases} (0,1) & \text{with probability } p \uparrow \\ (1,0) & \text{with probability } p \rightarrow \\ (0,-1) & \text{with probability } p \downarrow \\ (-1,0) & \text{with probability } p \leftarrow \\ (0,0) & \text{with probability } p \bullet \end{cases} \tag{9}$$

Implementation At the top of the file, it is possible to set the grid size, starting position of particles, # of particles, simulation horizon and motion direction probabilities. The script will also generate snapshots of the particle distribution at different moments in time during the simulation. By selecting

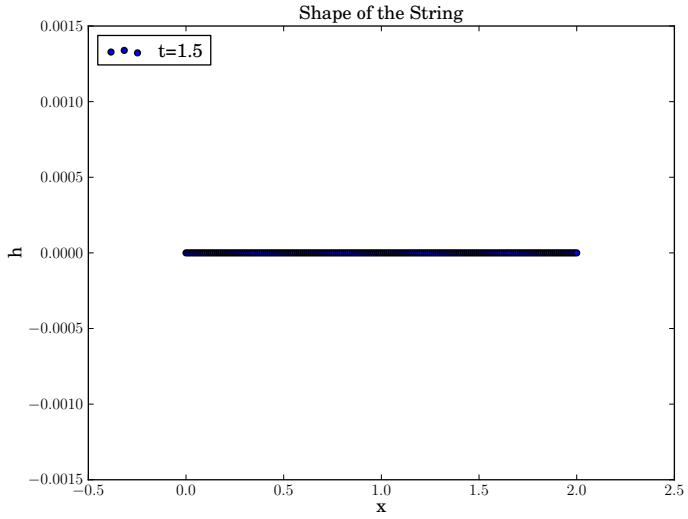

Fig. 3.9: Shape of the string when t=1.5

the number of subplots, you can determine how many instances you would like to see. This will provide insights into how the particle distribution develops over time.

Particles are simulated one at a time. The simulation runs until the time horizon T is reached or the particle hits the wall. Their position is saved in a 3-dimensional array (time, x-position, y-position) at specified moments in time only; this way the amount of required memory is kept to a minimum.

Information about where particles hit the wall is included in the same arrays. This data "circumvents" the $n \times n$ data about the particle distribution within the grid. As a result, when we plot the full array we can see the distribution of particles in the grid at their respective location, and the distribution of particles at the wall directly "behind" the wall.

```
# This file will model/simulate the random motion

import numpy as np
import matplotlib.pyplot as plt
from random import *
import pylab

############### START INPUT ##################
```

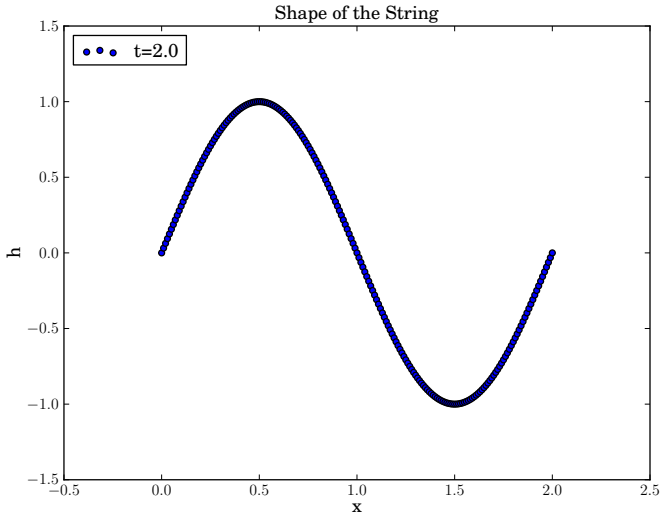

Fig. 3.10: Shape of the string when t=2

```
GridSizeSquare = 20 # n, will create an nxn grid
#define starting position as ratio or grid size
Pos_init = np.ceil(np.array([0.4,0.4])*GridSizeSquare)
#T is the total amount of time steps, scale with square of
                                grid size. 0.3 is nice
T = 0.3*np.power(GridSizeSquare, 2)
n_particles = 10000 # #particles. 10000+ recommended
# define the # of subplots for intermediate time snap shots
subplot_row = 2
subplot_column = 2
#defines the drift probabilities: ([up,right,down,left,0])
```

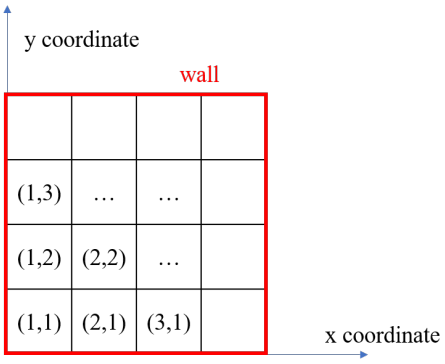

Fig. 3.11: Coordinate representation in spatial grid

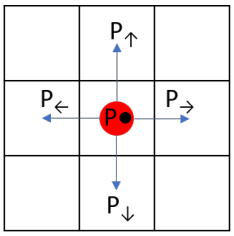

Fig. 3.12: Potential motion per time step

```
motion_prob = np.array([0.2,0.2,0.2,0.2,0.2])

############## END INPUT #################

#only look at square grids, but this could be changed:
x_grid = GridSizeSquare
y_grid = GridSizeSquare
n_subplot = subplot_row*subplot_column
Plot_interval = np.floor((T-1)/(n_subplot-1))
#This determines when you take snapshots of the process,
                              every "Plot_interval" time
                              steps

# now define the probabilities of the random walk
# notation of motion ([up,right,down,left,no motion])
# I normalized in case probabilities do not add up to 1...
motion_prob = motion_prob/np.sum(motion_prob)
# define the change in coordinates of every motion:
motion_xy = np.array([[0,1],[1,0],[0,-1],[-1,0],[0,0],])
motion_prob_percentile = np.cumsum(motion_prob)
# this is used later to draw from with randomizer

# make an empty data grid from where you are going to count
                              the amount occurrences and
                              hits against the wall
Data = np.zeros((x_grid+2,y_grid+2)) # "+2" for wall data
Data_resized = np.zeros((Data.shape[0]+1,Data.shape[1]+1))
# This for plotting purposes, I need to add an extra row an
                              column for some reason
# Now create a new empty data set for the intermediate plots:
Data_TimeVarying = np.zeros((n_subplot,Data_resized.shape[0],
                              Data_resized.shape[1]))

# construct some arrays for plotting later on:
xx, yy = pylab.meshgrid(
    pylab.linspace(-1,x_grid+1,x_grid+3),
    pylab.linspace(-1,y_grid+1,y_grid+3))
```

```
# Here, start loop over all the particles after each other:
for i in range(1, n_particles+1):
    # Initialize simulation
    t = 0
    HitWall = False
    Pos = Pos_init
    Subplot = 1

    # start simulation
    while t < T and not HitWall:

         # Now continue with the motion simulation
        MotionRandom = random()
        IndexMotion = np.argmax(motion_prob_percentile>
                                    MotionRandom)
        Pos = Pos + motion_xy[IndexMotion,:] #this works

        # Now check for hitting the wall
        if Pos[0] == 0 or Pos[1] == 0 or Pos[0] == x_grid+1
                                    or Pos[1] == y_grid+1:
            if Subplot <= n_subplot:
                Data_TimeVarying[Subplot-1,Pos[0],Pos[1]] =
                                            Data_TimeVarying
                                            [Subplot-1,Pos
                                            [0],Pos[1]]+1
            HitWall = True

        # Now record the position for time dependent plotting
                                     purposes
        if t % Plot_interval == 0 and Subplot <= n_subplot
                                    and not(HitWall):   #
                                    so create a subplot
                                    every Plot_interval
                                    time steps
            Data_TimeVarying[Subplot-1,Pos[0],Pos[1]] =
                                        Data_TimeVarying[
                                        Subplot-1,Pos[0],
                                        Pos[1]]+1
            Subplot = Subplot+1

        t = t+1


    #This was basically the whole simulation, now save the
                                    results
    Data[Pos[0],Pos[1]] = Data[Pos[0],Pos[1]] + 1
    # I need to give resize Data with an extra row and column
                                    , since pcolor doesn't
                                    plot the full range of the
```

```
                                    matrix...
    Data_resized[:-1,:-1] = Data


# Now I need to do some post-processing of the intermediate
                                    measurements.
# I need to add the particles that hit the wall in earlier
                                    time steps to the
# later plots, so the total amount of particles is always n
Data_TimeVarying_Corrected = np.cumsum(Data_TimeVarying,axis=
                                    0)
Data_TimeVarying_Corrected[:,1:x_grid+1,1:y_grid+1] =
                                    Data_TimeVarying[:,1:x_grid+1,
                                    1:y_grid+1]


# This loop creates subplots at several time instances
plt.figure()
for j in range(1, n_subplot+1):
    #Now visualize the outcomes
    pylab.subplot(subplot_row, subplot_column, j)
    pylab.pcolor(xx,yy,np.transpose(
                                        Data_TimeVarying_Corrected
                                        [j-1,:,:]))
    TitleString = 'Distribution at t = ' + str((j-1)*
                                        Plot_interval+1)
    pylab.title(TitleString)
    # and a color bar to show the correspondence between
                                        function value and color
    pylab.colorbar()
    pylab.hold(True)
    pylab.plot([0, x_grid],[0, 0], 'r',[0, x_grid],[y_grid,
                                        y_grid], 'r',[0, 0],[0,
                                        y_grid], 'r',[x_grid,
                                        x_grid],[0, y_grid], 'r')
    pylab.plot(Pos_init[0]-0.5,Pos_init[1]-0.5,'ro')

pylab.show()

# This plot shows the final distribution, including
                                    distribution along the walls
plt.figure()
pylab.pcolor(xx,yy,np.transpose(Data_resized))
pylab.title('Final distribution at t = %d, including hitting
                                    walls' %T)
# and a color bar to show the correspondence between function
                                     value and color
pylab.colorbar()
pylab.hold(True)
```

```
pylab.plot([0, x_grid],[0, 0], 'r',[0, x_grid],[y_grid,
                                y_grid], 'r',[0, 0],[0, y_grid
                                ], 'r',[x_grid, x_grid],[0,
                                y_grid], 'r')
pylab.plot(Pos_init[0]-0.5,Pos_init[1]-0.5,'ro')
pylab.show
```

Results In this section we included two simulation for both a $10x10$ grid and a 20×20 grid, with a different simulation time horizon T. Both simulations use 10,000 particles and have a motion probability of 0.2 in all directions.

It is clearly visible how the particles spread out over time and make their way to the walls over time. The more particles that are simulated per grid resolution, the smoother the distribution becomes. You can clearly see that 10,000 particles simulated lead to a clean distribution in the smaller plot of figure 3.2.4, while the larger plot of 3.2.4 shows a more grainy distribution.

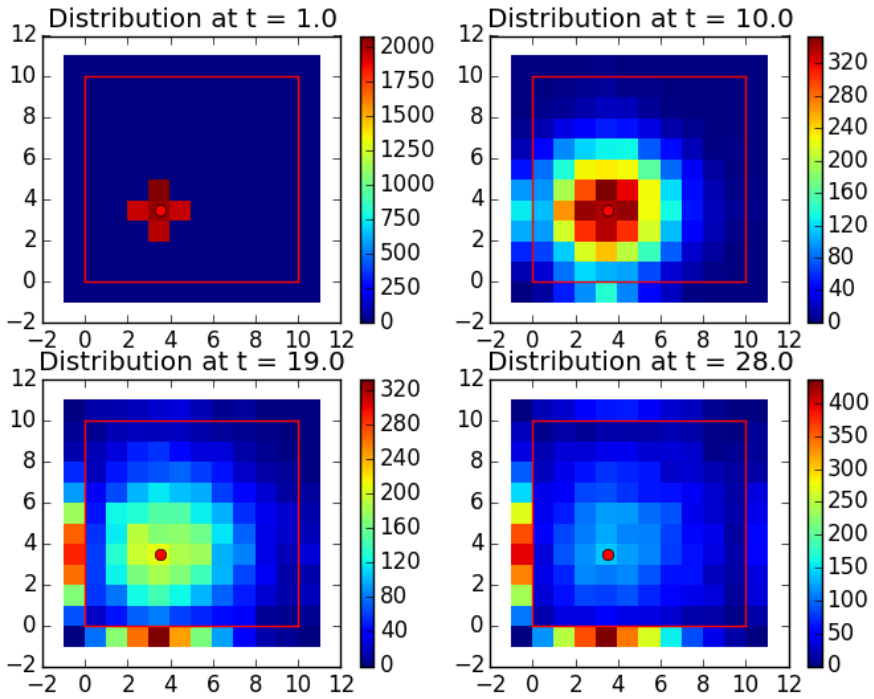

Fig. 3.13: Particle distribution for 10×10 grid at different time steps

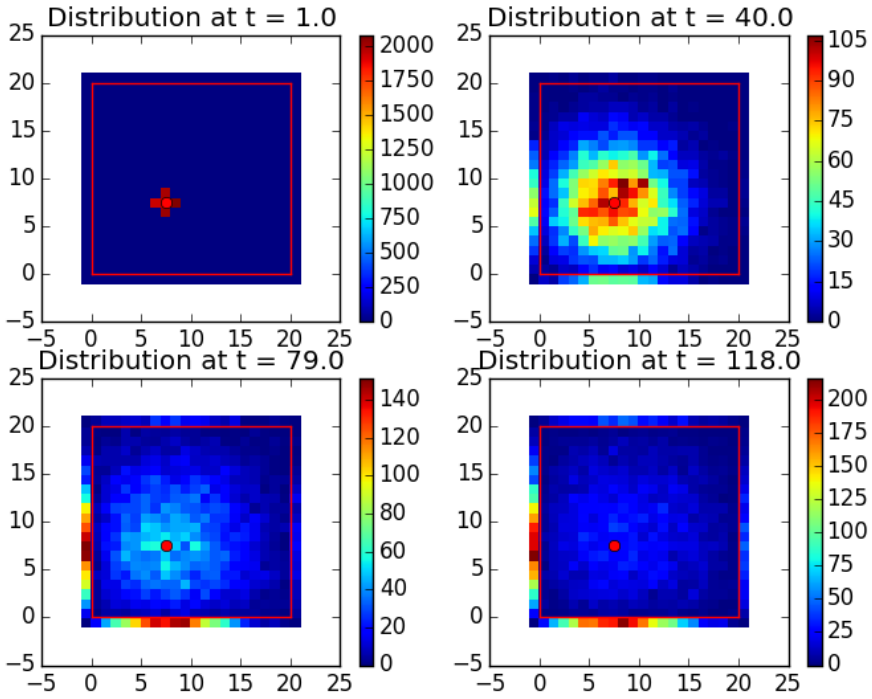

Fig. 3.14: Particle distribution for 20×20 grid at different time steps

Interpretation *Relate these quantities to the model and to theoretical knowledge of the course.*

I think this motion is described by Fick's Law in 2 dimension. The concentration on a specific point changes over time, depending on the concentration of its surroundings. Should we derive why the second derivative matters? ϕ is the concentration, D the diffusion coefficient.

$$\frac{d\phi}{dt} = D\nabla\phi = D\left(\frac{d^2\phi}{dx^2} + \frac{d^2\phi}{dy^2}\right) \tag{10}$$

Conclusion *What have we learned? Is everything aligned (theory and practice)? What was difficult? Provide perspectives.*

The dynamics of the random walk were easy to model. The challenge in this simulation was to save the data for the intermediate time steps. Since the particles are simulated one by one for the full time horizon, we had to write some non-intuitive code to save the location of every particle at the

relevant intermediate time steps.

From the lecture we recall that diffusion distance scales with the square root of time. Here we tried to simulate that. When doubling the grid size and taking a four times higher simulation horizon, the distribution looks similar. However, we have the idea that the scaling did not work for 100%. At the end of the scaled simulation horizon, it seems as if the smaller grid has relatively more particles in the grid than the larger grid has. What could cause this difference?

Chapter 4

Control

4.1 Objectives