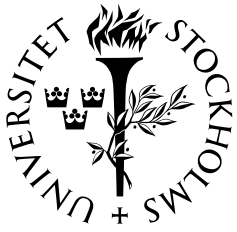

EXAMENSARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Diffusion equation and Monte Carlo

av

Anders Österling

2007 - No 9

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET, 10691 STOCKHOLM

Diffusion equation and Monte Carlo

Anders Österling

Examensarbete i matematik 20 poäng, fördjupningskurs

Handledare: Jan-Erik Björk

2007

Abstract

Introducing the Brownian motion in the way of Einstein and Wiener we find the connection between a Wiener Process and the Heat Diffusion PDE. We solve the PDE analytically for some boundary conditions and then use the connection to the Wiener Process to solve more complex BVP's using Monte Carlo simulations in Matlab.

Acknowledgements

To write a thesis is the final step in a long and sometimes discouraging process and therefore I would like to thank my supervisor Professor Jan Erik Björk at Stockholm University. The immense knowledge that professor Björk has in vast areas of mathematics was invaluable for me in order to elaborate this particular thesis building on theories as various as Einstein and Wiener.

I also apologize for any language mistakes contained within. English is not my first language and I have not been able to find a decent spell-or grammar checker for LaTex application. I therefore thank the committee for having taken this in consideration while evaluating my academic work.

Last but not least, I want to thank my parents for their constant encouragement and care for me and my studies.

Contents

1 Introduction

This MSc thesis focuses on the solutions of the heat diffusion partial differential equation (PDE) for different, often very complicated, boundary value problems (BVP's) with the Dirac delta operator as initial condition. Some BVP's are solved analytically using standard PDE-solving techniques whilst most of the solutions are found using Monte Carlo simulations of random walks.

We describe briefly the elegant connection between random walks and the diffusion equation, which is due mainly to research carried out by Einstein and Wiener. Throughout the thesis we use only basic Monte Carlo algorithms derived by the author in an attempt to emphasize the extreme ease with which one can solve very complicated BVP's on a normal PC. We are not trying to solve a particular real life problem, but rather derive (from scratch) a computer intense method that one can apply to solve different BVP's. In the scope of this thesis, we are only interested in proof of concept, rather than in dept study.

The Wiener process used throughout this thesis is commonly used in Mathematical Finance and as such the repeated use of this process hopefully aim to give a better understanding of the processes behind the models for Derivative Pricing, Risk analysis etc...

The source code for the simulations are presented in the appendix, where I also include an introduction to the Black-Scholes (BS) framework along with a transformation from BS formula to the diffusion equation. Thus showing that the above Monte Carlo solutions could actually solve the BS formula and be used to price Financial Derivatives.

In the context below the heat diffusion PDE is often referred to as simply the diffusion equation.

In chapter 2 we solve the diffusion equation analytically for three different BVP's: unbound, with terminating boundaries (cold walls) and with bouncing boundaries (isolated walls).

Chapter 3 is devoted to solving the above problems using Monte Carlo simulations of random walks, and in chapter 4 we show how easy it is to modify the Monte Carlo simulations to solve problems that would be very hard indeed to solve using analytical methods. In chapter 4 we also introduce drifts when bouncing on the walls, whilst in chapter 5 the drift is introduced at deterministic time intervals.

Stochastic slalom is dealt with in chapter 6, where we first explain how one could solve this kind of problems analytically using convolution. But since the convolutions would get increasingly complicated we solve the problems using Monte Carlo simulations.

2 The heat-diffusion PDE

The heat-diffusion PDE, sometimes referred to simply as the *diffusion equation*, is probably the most well-studied of all PDE's, and its name derives from the use to describe the evolution and transmission of heat in a metal rod. The diffusion equation was first discovered in the 1820's by Fourier and Laplace.

In the year 1900 Louis Bachelier presented his Ph.D. thesis *Théorie de la Spéculation* which was the beginning of mathematical finance as we know it today. The content was not fully mathematically rigorous but it was intuitively correct, and many of the included ideas amazed, amongst others, his supervisor Poincaré. The most striking ideas were:

- The market assumes that a stock-price process evolves under a martingale measure (i.e that the Brownian motion governing the stock price has no drift except possibly that of the riskless asset).
- The stock-price evolves as a continuous Markov Process and this process satisfies what is now know as the Chapman-Kolmogorov equation.
- The Normal distribution function solves the Chapman-Kolmogorov equation.

Bachelier also observed that the distribution functions of the stock price process and hence, by the above, the Gaussian distribution function, satisfies the diffusion-equation [ea00].

Section 4 in Einsteins paper on Brownian Motion (1905) is titled *On the Irregular Movement of Particles Suspended in a Liquid and the Relation of this to Diffusion* and within Einstein assumes that the irregular movement generated by thermal molecular movement is a Brownian motion. Einstein then derives the heat-diffusion equation from the Brownian motion and hence demonstrating that they are essentially the same thing [Ein05].

The great research carried out by of Bachelier and Einstein in the early 20th century connects the diffusion equation with the Brownian motion, but it took another 30 years before Paley and Wiener found the connection between a discrete random walk and the normal distribution. These two connections together will let us simulate PDE's[1] and Brownian motion by repeatedly tossing of a coin. As such, these are key elements in the theory behind stochastic simulations.

[1]It is a well known fact in the theory of stochastic calculus (and hence mathematical finance) that under some chosen martingale measure one often obtains a PDE when equating the drift coefficient under the 'observed' dynamics equal to that of the chosen martingale measure. For example we can take the geometrical Brownian motion which, when equating the drift under the objective martingale measure to that of the risk-free martingale measure, generates the famous Black-Scholes PDE.

2.1 Diffusion Equation and the Gaussian Distribution

As demonstrated by Einstein, we shall now show that the normal distribution function may solve the diffusion equation, based on the following definitions:

Definition 2.1.1 (Stationary Increments) *A stochastic process is said to have* stationary increments *if the events that occur in the time interval* $(t, t+s)$ *have the same distribution* $\forall t$*, where* s *and* $t \geq 0$*. Stationary increments thus mean that the events that occur in an interval does not depend on when the interval occurs but only on its length. In formulae this means that* $(W(t+s) - W(t))$ *is dependent on* s *only.*

Definition 2.1.2 (Independent Increments) *A stochastic process is said to have* independent increments *if the events that occur in different adjoint time intervals are independent. I.e. that* $\forall t_1 < t_2 < t_3 < t_4$ $(W(t_2) - W(t_1))$ *is independent of* $(W(t_4) - W(t_3))$*.*

Definition 2.1.3 (The Wiener Process) *A stochastic process* $W = \{W(t) : t \geq 0\}$ *is called a* Wiener Process *if*

1. $W(0) = 0$
2. *W has stationary and independent increments*
3. $\forall\, t > 0, W(t)$ *follows a normal distribution with zero mean and variance* σ^2*.*

The process is called a Normalized Wiener Process *if* $\sigma = 1$*.*

It it important to note that part 3 in the above definition can be interpreted as

$$\begin{aligned} W(t+s) - W(t) &\sim N(0, \sigma^2 s), \text{ or} \\ Prob(W(t+s) = y | W(t) = x) &\sim N(x, \sigma^2 s) \end{aligned} \tag{2.1}$$

where the "|" reads *conditional on*.

Most of the highlights in Einsteins research was in the Physical theory. But in one of his *Annus Mirablis Papers*, more specifically the paper on Brownian Motion, he also demonstrated (from a mathematical point of view) that a connection between the Brownian motion and the diffusion equation. See also [NW34] p. 157 for a further discussion.

I will shortly describe Einsteins realisation of this connection, but then another proof of the connection will be demonstrated which is a little more mathematically straightforward.

Einstein models the "irregular movement caused by thermal molecular movement" by using a Gaussian process. He finds the number of particles (n), which moves the distance $(\Delta, \Delta + \partial\Delta)$ under the time interval $(t, t+\tau)$ as

$$dn = n\phi(\Delta)d\Delta,$$

where

$$\begin{aligned}\int_{-\infty}^{\infty}\phi(\Delta)d\Delta &= 1, \text{ and}\\ \phi(\Delta) &= \phi(-\Delta)\end{aligned}$$

and "ϕ only differs from zero for very small values of Δ". τ is chosen as a small number and hence ϕ will be recognized as the distribution function of a Gaussian random variable with the variance τ.

In investigating how the coefficient of diffusion depends on ϕ, Einstein defines $v = f(x,t)$ the number of particles per unit volume (assuming v depends only on x,t), and calculates the distribution of the particles at time $t+\tau$ conditional on the distribution at time t. He finds the number of particles that are located between two planes perpendicular to the x-axis, the first plane at $x = x$ and the second at $x = x + \partial x$, at a future time $t + \tau$ as

$$f(x, t+\tau)dx = dx\int_{-\infty}^{\infty} f(x+\Delta)\phi(\Delta)\partial\Delta.$$

Einstein uses the fact that τ is really small and then finds the Taylor expansion for f, which he then integrate over $d\Delta$. Every other term of the Taylor series will disappear due to fact that $\phi(\Delta)$ is an even function, and terms of $O(\Delta^3)$ or higher will disappear since Δ is small. After doing these manipulations Einstein arrives at the equation

$$\frac{\partial f}{\partial t} = D\frac{\partial^2 f}{\partial x^2}$$

which is the diffusion equation with diffusion coefficient D:

$$D = \frac{1}{\tau}\int_{-\infty}^{\infty}\frac{\Delta^2}{2}\phi(\Delta)\partial\Delta.$$

The above derivation was useful to physicists, but the latter parts of it are complex. The following version is an attempt to provide a clarification.

Consider a Wiener process W such that $W(t) = x$ for some t. Starting off just like Einstein, we want to find the probability that, after a short time δt, the process will be in state y, where $y = x + \delta x$, and δx small. From probability theory, we know that

$$P(W(t+\delta t) \le y | W(t) = x) = F(t+\delta t, y|t, x)$$

where F is the joint probability distribution function for $t+\delta t$ and y, conditional on t and x. We also know that

$$\frac{\partial}{\partial y}F(t+\delta t, y|t, x) = f(t+\delta t, y|t, x),$$

where f is the joint probability density function for $t+\delta t$ and y. This probability density function is the same as the one Einstein defined as "number of particles per unit volume". By (2.1) we get

$$W(t+\delta t) - W(t) \sim N(x, \sigma^2(t+\delta t - t)) = N(x, \sigma^2\delta t)$$

and the density function for such a normal distribution is given by

$$f(\xi,t)=\frac{1}{\sqrt{2\pi\sigma^2\delta t}}e^{-(\xi-x)^2/2\sigma^2\delta t}. \tag{2.2}$$

Accurately combining partial derivatives lead us to discover the desired PDE. In differentiating, it might help to look at δt as $\delta t=\delta * t$, where δ is almost zero. By using the chain-rule and the fact that $\frac{\partial}{\partial t}f^n(t)=nf^{n-1}\frac{\partial f}{\partial t}$ we get

$$f_t(\xi,t)=\frac{1}{2}e^{\frac{-(\xi-x)^2}{2\sigma^2\delta t}}\left(\frac{-1}{\sqrt{2\pi}\sqrt[3]{\sigma^2\delta t}}+\frac{(\xi-x)^2}{\sqrt{2\pi}\sqrt[5]{\sigma^2\delta t}}\right), \tag{2.3}$$

$$f_x(\xi,t)=e^{\frac{-(\xi-x)^2}{2\sigma^2\delta t}}\frac{(\xi-x)}{\sqrt{2\pi}\sqrt[3]{\sigma^2\delta t}}, \text{ and}$$

$$f_{xx}(\xi,t)=e^{\frac{-(\xi-x)^2}{2\sigma^2\delta t}}\left(\frac{-1}{\sqrt{2\pi}\sqrt[3]{\sigma^2\delta t}}+\frac{(\xi-x)^2}{\sqrt{2\pi}\sqrt[5]{\sigma^2\delta t}}\right). \tag{2.4}$$

Equating equations (2.3) and (2.4) above gives us the relation

$$f_t=\frac{1}{2}f_{xx}. \tag{2.5}$$

Equation (2.5) is the same, up to a constant, as the equation that Einstein found.

Important to note is that the f used in the above is a probability density function and should in no way be confused with the $f(x,0)$ used in later sections to denote the initial condition. In sections where confusion might arise we denote the probability density function by u.

2.2 Discrete Random Walks and Gaussian Distribution

The fact that Brownian motion can be "created" using a random sequence of ± 1's, essentially dates back to 1934 and the work on *Random Functions* by Norbert Wiener and Raymond E. A. C. Paley. The authors showed that by using a denumerable[2] set of random numbers they could create a random variable (approximately Gaussian in the X-direction via The Central Limit Theorem) with a continuous range. The proof is rather deep and can be found in [NW34].

As curiosa can be mentioned that the proof was based on a transformation from a random variable uniformly distributed over the interval $(0, 1)$ to a complex random variable with independent and Gaussian real and imaginary parts. The result was the following:

Suppose that α_i, $\alpha_j \sim Un(0, 1)$ and consider

$$\rho = \sqrt{-log(\alpha_j)}e^{2\pi i\alpha_j}.$$

Then both the real and the imaginary parts of ρ have independent, Gaussian (and hence distributed on $(-\infty, \infty)$) distributions. For a proof of this transformation see [NW34] page 146.

[2] A set is denumerable, or countably infinite, if there exists a bijective function from the set to a subset of the natural numbers.

2.3 Solutions to the Heat Diffusion Equation

According to Einstein's results of Einstein we have derived the heat-diffusion equation (2.5) and now we aim to solve it for a few different boundary conditions. If no boundaries are present, we already know that (2.2) is a solution to (2.5), but in order to find the solution when BC's are present, we will seek the general solution to (2.5) by using the method of separation of variables.

Suppose that u could be written in the form

$$u(t,x) = \Theta(t) * \Xi(x).$$

Then we get the derivatives as

$$\begin{aligned} u_t &= \Theta'(t)\Xi(x), \text{ and} \\ u_{xx} &= \Theta(t)\Xi''(x). \end{aligned}$$

Via (2.5) we get

$$\begin{aligned} \Theta'(t)\Xi(x) &= \frac{1}{2}\Theta(t)\Xi''(x) \\ \Rightarrow \frac{\Theta'(t)}{\Theta(t)} &= \frac{\Xi''(x)}{2\Xi(x)}. \end{aligned}$$

The left hand side of the above equation depends only on t and the right hand side depends only on x. One can not vary one without the other, thus they must be constant, and we can rearrange to get

$$\frac{\Theta'(t)}{\Theta(t)} = \frac{\Xi''(x)}{2\Xi(x)} = -\lambda,$$

where λ is an arbitrary constant and we assume $\lambda > 0$. This reduces the PDE problem into two ODE's, which we now aim to solve.

The first ODE,

$$\Theta'(t) + \lambda\Theta(t) = 0, \tag{2.6}$$

is a *first order linear homogeneous ODE with constant coefficients* and it's general solution takes the form

$$\Theta(t) = Ce^{-\lambda t}, \tag{2.7}$$

where C is some constant.

The second ODE,

$$\Xi''(x) + \lambda\Xi(x) = 0, \tag{2.8}$$

is a *second order linear homogenous ODE with constant coefficients.* Since we assumed $\lambda > 0$ we get the solution

$$\Xi(x) = De^{\pm i\sqrt{\lambda}x}, \tag{2.9}$$

where D is some constant.

With the relation $-i = \frac{1}{i}$ in mind, consider the following standard definitions

$$\sin x = \frac{ie^{-ix} - ie^{ix}}{2}, \text{ and} \tag{2.10}$$

$$\cos x = \frac{e^{ix} + e^{-ix}}{2}. \tag{2.11}$$

The *Principle of Superposition* (see, for example [Hab04], page 37) states that if two functions satisfy a linear homogeneous equation, then any linear combination of these two functions also satisfy the same linear homogeneous equation. We now try

$$c_1 \cos\sqrt{\lambda}x + c_2 \sin\sqrt{\lambda}x$$

as a solution for the PDE. Inserting (2.10) and (2.11) into this gives

$$\frac{c_1 e^{i\sqrt{\lambda}}x + c_1 e^{-i\sqrt{\lambda}}x}{2} + \frac{ic_2 e^{-i\sqrt{\lambda}}x + ic_2 e^{i\sqrt{\lambda}}x}{2},$$

which can be re-arranged as

$$\frac{1}{2}\left((c_1 - ic_2)e^{i\sqrt{\lambda}x} + (c_1 + ic_2)e^{-i\sqrt{\lambda}x}\right),$$

which is a linear combination of (2.9). So by the Principle of Superposition

$$\Xi(x) = c_1 \cos\sqrt{\lambda}x + c_2 \sin\sqrt{\lambda}x \tag{2.12}$$

is also a solution to the ODE (2.8).

We started by assuming $u(t,x) = \Theta(t) * \Xi(x)$ and then went on to find $\Theta(t)$ (2.12) and $\Xi(x)$ (2.7), and so we get the solution for the PDE (2.5) as

$$\Xi(x)\Theta(t) = u_\lambda(x,t) = \left(c_1 \cos\sqrt{\lambda}x + c_2 \sin\sqrt{\lambda}x\right) Ce^{-\lambda t}, \tag{2.13}$$

where we add the subscript $_\lambda$ to emphasize the dependency of λ.

To finish this section we conclude that for each $\lambda > 0$ the function $u_\lambda(x,t)$ solves the Diffusion-Equation $u_t = \frac{1}{2}u_{xx}$, where c_1, c_2 and C can be chosen to suit ones purpose. This is easily checked by differentiating u_λ once w.r.t t and twice w.r.t. x and then equating the resulting equations.

2.3.1 Dirac-Delta Function

$u_\lambda(x,t)$ can be interpreted as the probability to be at position x, at time t, for a particle moving according to a standardized Wiener process. This is a useful and general result which can be built upon to solve more specific problems, such as bound processes with either terminating or "bouncing" boundaries. First, however, we need to familiarize ourselves with Initial Conditions.

Initial conditions define, as the name implies, what happens initially with the process. We want to model a process that starts at some specific point $x = x_0$ at time $t = 0$. There should only be one process in the system at every

time, for else they might crash into each other and interact, and it turns out it is a good idea to use the initial condition $u(x,0) = f(x) = \delta(x - x_0)$, where $\delta(x - x_0)$ is known as the *Dirac Delta function*, to achieve this.

It is important to note that the 'crashing' of atoms that gave the process it's name *Brownian Motion* is not what we are trying to model. We assume that every single one of our processes move according to Brownian motion when run separately, and we do not know what would happened if we let many of them run at the same time. Maybe they would interact and arrive in a fashion not independent of each other? Then our calculations would be inaccurate.

Briefly, the *Dirac Delta function* is a function that is virtually zero everywhere apart from at x_o where it is ∞. However $\delta(x-x_0)$ is not really a function, since a function can not be defined to equal infinity at any point, and instead the Dirac Delta function can be thought of as an operator. The nature of this operator is quite valuable and is defined as follows:

Definition 2.3.1 (Dirac Delta Function) *The Dirac Delta function is an operator defined as*

$$\delta(x - x_0) \quad = \quad \begin{cases} 0 & \text{if } x \neq x_0 \\ \infty & \text{if } x = x_0. \end{cases}$$

It has the properties that

$$\int_{-\infty}^{\infty} \delta(x - x_0)dx = 1, \tag{2.14}$$

$$\int_{-\infty}^{\infty} g(x)\delta(x - x_0)dx = g(x_0), \text{ and} \tag{2.15}$$

$$\int_{\gamma}^{\infty} \delta(x - x_0)dx = 0 \; \forall \; \gamma > x_0. \tag{2.16}$$

Since the Dirac Delta function is defined to be symmetric about x_0 equation (2.16) is also true when $\lim_{A\to-\infty} \int_A^{\gamma}$, $\gamma < x_0$. For convenience, we sometimes write $\delta(x - x_0) = \delta_{x_0}$.

By using the Dirac Delta operator as the Initial Condition, we tell the Heat-Diffusion equation that all processes shall start at $x = x_0$ at $t = 0$, and hence the drunken man Erik will commence his drunken stroll at the same pub, at the same time, many times around[3]. We shall also apply Boundary Conditions to tell the process what shall happen when some specified boundaries (walls, ditches) are used, but to fully understand this some more theory will be explained in the paragraphs below.

[3]The number of times Erik will walk drunken equals the number of independent universii he lives within.

2.3.2 Fourier Series

In his famous work on heat flow "Théorie analytique de la chaleur" (1822), Joseph Fourier developed what is today known as the Fourier Series. The results, although somewhat faulty[4], was a scientific breakthrough at the time but are today considered standard techniques: techniques taught at undergraduate levels in mathematics and engineering on how to solve Partial Differential Equations.

A definition of the Fourier Series is provided below. For an in-depth explanation, please consult [Hab04] chapter 3, and [Rud76] chapter 8.

Definition 2.3.2 (Fourier Series) *The Fourier series of a function $f(x)$ on the interval (-A,A), where $A \in \mathbb{R}$, is written as*

$$\textit{Fourier series} = A_0 + \sum_{n=1}^{\infty} A_n \cos\left(\frac{n\pi x}{A}\right) e^{-\lambda t} + \sum_{n=1}^{\infty} B_n \sin\left(\frac{n\pi x}{A}\right) e^{-\lambda t} \quad (2.17)$$

with the Fourier coefficients *defined as*

$$A_0 = \frac{1}{2A}\int_{-A}^{A} f(x)dx \quad (2.18)$$

$$A_n = \frac{1}{A}\int_{-A}^{A} f(x)\cos\left(\frac{n\pi x}{A}\right)dx \quad (2.19)$$

$$B_n = \frac{1}{A}\int_{-A}^{A} f(x)\sin\left(\frac{n\pi x}{A}\right)dx. \quad (2.20)$$

It should be noted that the Fourier Series of $f(x)$ and the function $f(x)$ are not equal. The Fourier series may not converge at all (it is, after all, quite a general infinite series), and if it does converge it may not converge to $f(x)$. Only if the series converge are the Fourier Coefficients above valid.

The aim of the next few sections is to find Fourier Series for $u(x,t)$, with special restrictions (BC's and IC), and that does indeed converge to $u(x,t)$.

[4] According to the article on J. Fourier on wikipedia.org. The internet certainly holds a lot of erroneous facts, but since it does not really matter in what follows whether Fourier was right or wrong I will not dwell any further on the topic.

2.4 Scenario I: No Boundaries

It is important to first find the solution to the diffusion equation with no boundaries using the Dirac Delta function as the initial condition. As Einstein showed us, in section 2.1, the Normal distribution can be a solution to the diffusion equation in general. However, we know nothing about the particular case, when using the Dirac Delta as the initial condition.

I came across a very good solution, using *Fourier Transforms*, to this problem explained in chapter 10 of [Hab04]. This solution was applied using matlab's `plotNoBoundriesFourierSer.m` to illustrate this solution at different times (see 2.4). To better explain this solution the following theorem and proof is included.

Theorem 2.4.1 (No Boundaries) *The solution to the diffusion equation* $u_t = \frac{1}{2}u_{xx}$*, with the initial condition* $u(x,0) = \delta_{\bar{x}}$*, is given by*

$$u(x,t) = \frac{1}{\sqrt{2\pi t}} e^{\frac{-(x-\bar{x})^2}{2t}}. \tag{2.21}$$

This is called the fundamental solution *of the diffusion equation.*

Proof: *see [Hab04] chapter 10.*

The equation (2.21) is nothing but the probability density function of a Normally distributed random variable with mean $\bar{x}$ and variance $t > 0$, and, at least intuitively, it is possible to see how, as $\Delta \to 0$, the normal distribution with variance Δ will tend to the Dirac Delta function.

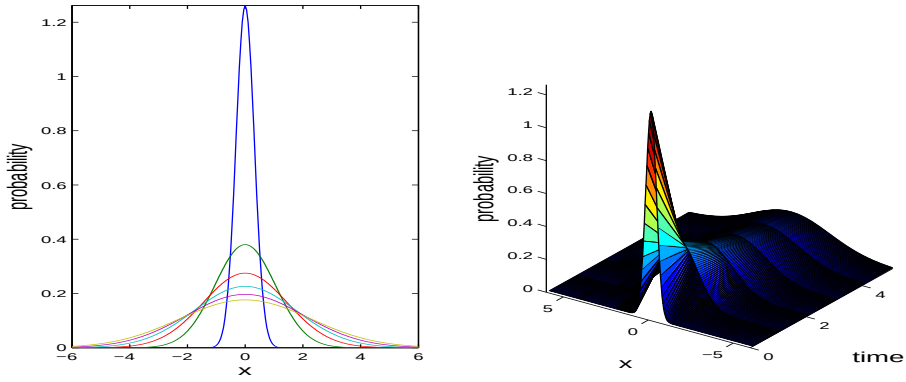

Figure 1: Plotting the solution for Theorem 2.4.1 for discrete times $t \in (.1, 1.1, \cdots, 5.1)$.

2.5 Scenario II: Boundries make Erik Fall off a Bridge

Let us get back to imagining the drunken man Erik and his Friday evening walk home from the local pub. Somewhere along this road there is an un-fenced bridge, acting as the boundry, that Erik has to pass over in order to get back to the comfort of indoors. We now focus on this bridge and the horrible scenario that Erik would fall over the edge of the bridge.

2.5.1 Boundary Conditions

The edges of the bridge constitute our boundaries and by scaling the width of the road to be π wide, the boundary conditions $u(0,t) = u(\pi,t) = 0$ are defined. We now apply these BC's to equation (2.12) to get the solution to the PDE for these specific boundaries:

$$\begin{aligned} \Xi(0) = 0 \quad &\Rightarrow \quad 0 = c_1 \cos 0 + c_2 \sin 0 \\ &\therefore c_1 = 0. \end{aligned}$$

$$\begin{aligned} \Xi(\pi) = 0 \quad &\Rightarrow \quad 0 = c_2 \sin \sqrt{\lambda}\pi \\ &\Rightarrow \sqrt{\lambda} = n, n \in \mathbb{N} \\ &\therefore \lambda = n^2. \end{aligned}$$

We have

$$\Theta(t) = Ce^{-\lambda t} \tag{2.22}$$

$$\Xi(x) = c_2 \sin(nx). \tag{2.23}$$

c_1 is the cosine-coefficient and setting this to zero thus corresponds to zeroing all the $A_n's$ in equation (2.19). The product solution of the heat-diffusion PDE, with boundary conditions $\Xi(0) = 0$ and $\Xi(\pi) = 0$, is thus

$$u(x,t) = c_2 \sin(nx) e^{-n^2 t},$$

and the Fourier Series solution is given as

$$u^F(x,t) = \sum_{n=1}^{\infty} B_n \sin(nx) e^{-n^2 t}, \tag{2.24}$$

with B_n as in equation (2.20). This is a correct solution for the diffusion-equation with given BC's, but it can be narrowed down further in order to fit the Dirac-Delta initial condition as explained in the next paragraph.

2.5.2 Initial Condition (IC)

We want to find the Fourier series u^F of a function u that, at $t = 0$, should equal $\delta_{\frac{\pi}{2}}$. Initially we have $t = 0$ and equating (2.24) with the Dirac Delta function

$\delta_{\frac{\pi}{2}}$ we are looking for a series of the form

$$\sum_{n=1}^{\infty} a_n \sin(nx) \approx \delta_{\frac{\pi}{2}}. \tag{2.25}$$

Using (2.15) we get

$$\begin{aligned} g\left(\frac{\pi}{2}\right) &= \int_0^{\pi} \delta_{\frac{\pi}{2}} g(x)dx, \text{ and via (2.25)} \\ &= \sum_{n=1}^{\infty} a_n \int_0^{\pi} g(x)\sin(nx)dx, \end{aligned} \tag{2.26}$$

which is valid $\forall g \in \mathscr{C}[0,\pi] : g(0) = g(\pi) = 0$, where $\mathscr{C}$ denotes the set of all once differentiable functions. We have infinitely many different g to choose from and so let us try a g of a form similar to (2.25):

$$g(x) = a_k \sin(kx),$$

where k is some integer and trivially $g(0) = g(\pi) = 0 \forall x$. Further, we put $g(x) \approx \delta_{\frac{\pi}{2}}$ and get

$$\sum_{n=1}^{\infty} a_n \sin(nx) \approx a_k \sin(kx)$$

which means that $a_n = 0 \forall n \neq k$ and a_k is yet to be found. Via (2.26) we get

$$g\left(\frac{\pi}{2}\right) = \sin\left(\frac{k\pi}{2}\right) = \begin{cases} 0 & \text{if } k = 2n \text{ (even)} \\ (-1)^n & \text{if } k = 2n+1 \text{ (odd).} \end{cases}$$

a_k is defined as

$$\begin{aligned} a_k &= \frac{2}{\pi}\int_0^{\infty} g(x)\sin(kx)dx, \text{ and since } g(x) \approx \delta_{\frac{\pi}{2}} \\ &= \frac{2}{\pi}\sin\left(\frac{k\pi}{2}\right) \text{and via (2.15)} \\ &= \begin{cases} 0 & \text{if } k = 2n \text{ (even)} \\ \frac{2}{\pi}(-1)^n & \text{if } k = 2n+1 \text{ (odd).} \end{cases} \end{aligned}$$

Using these coefficients would give us

$$u(x,0) = \frac{2}{\pi}\left(\sin(x) - \sin(3x) + \sin(5x) - \cdots\right).$$

In order for this series to represent a Dirac-Delta condition, we require that $\int_0^{\pi} u(x,0)dx = 1$. Since integration can be split over the sum, we can integrate each of the above sine functions separately as follows:

$$\int_0^{\pi} \sin(x)dx = 2, \ \int_0^{\pi} \sin(3x)dx = \frac{2}{3}, \ \int_0^{\pi} \sin(5x)dx = \frac{2}{5}, \cdots$$

Plugging in all the integrals and their respective coefficients we now get

$$\begin{aligned} I = \int_0^\pi u(x,0)dx &= \frac{2}{\pi}\left[2 - \frac{2}{3} + \frac{2}{5} - \frac{2}{7} + \cdots\right] \\ &= \frac{4}{\pi}\sum_{m=0}^{\infty}\frac{(-1)^m}{2m+1} \end{aligned}$$

$\sum_{m=0}^{\infty}\frac{(-1)^m}{2m+1} \to \frac{\pi}{4}$ as $n \to \infty$ and thus $I = 1$.

We summarize the results of this section in a Theorem:

Theorem 2.5.1 (Terminating Boundaries) *The diffusion equation $u_t = \frac{1}{2}u_{xx}$, with boundary conditions $u(0,t) = u(\pi,t) = 0 \forall t$, and initial condition $u(x,0) = \delta_{\frac{\pi}{2}}, x \in (0,\pi)$, is given by*

$$u(x,t) = \frac{2}{\pi}\sum_{m=0}^{\infty}(-1)^m \sin\left((2m+1)x\right) e^{-(m+1)^2 t}. \tag{2.27}$$

As an extra confirmation, and also illustration, of the results of Theorem 2.5.1, MATLAB was used to plot the solution for a few different discreete times t. The plots are shown in figure 2 and produced using
`plotTerminatingFourierSer.m` and `sumTerminatingFourierSer.m`.

The plots in figure 2 express conditional probability in order for the process to be alive at the end of the time interval. As a result of this condition the area of the probability distributions tend to zero as T gets larger. To find the survival percentage one simply integrates (2.27):

$$\int_0^\pi u(x,t)dx = \frac{2}{\pi}\sum_{m=0}^{\infty}(-1)^m \frac{2}{2m+1} e^{-(m+1)^2 t}$$

This is plotted in figure 3, using `plotIntegratedFourierSer.m` and `sumIntegratedFourierSer.m`, to show how rapidly the chance of survival tends to zero.

Remark: From a mathematical point of view it is interesting also to observe only the surviving processes on $0 \le x \le \pi$. This would be a distribution with total mass 1 and as t grows this would be (if it exists) the normalized stationary arrival distribution for the process.

The normalized arrival distribution is $u/\int_0^\pi u dx$ and to find the stationary distribution let

$$\frac{u(x,t)}{\int_0^\pi u(x,t)dx} = \rho(x,t)$$

and seek

$$\lim_{t\to\infty} \rho(x,t). \tag{2.28}$$

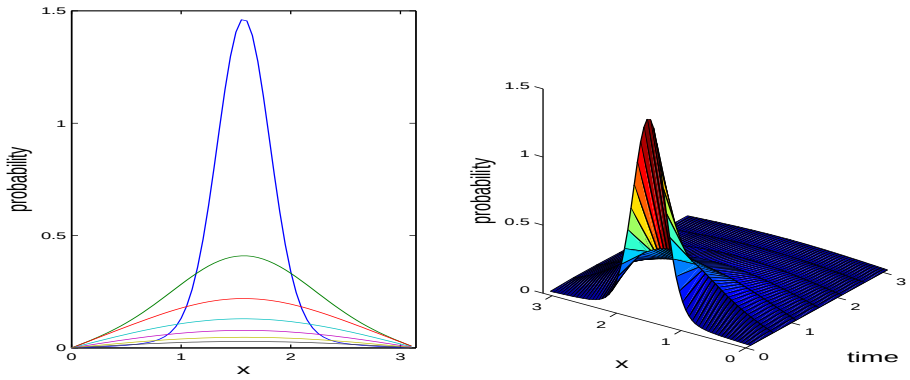

Figure 2: Plotting the solution in Theorem 2.5.1 for m=0,...,1000, and at different discrete times $t \in \{0.1, 0.6, 1.1, \cdots, 3.1\}$.

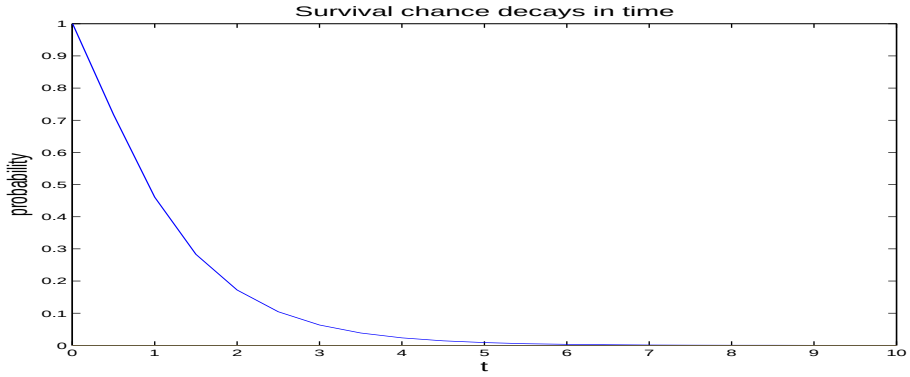

Figure 3: Plot showing how the survival chance decrease in time.

Trick: $\forall t \gg 0$ let $\delta = e^{-t}$. Then $\delta \ll 1$ and $e^{-(m+1)^2 t} \approx \delta^{(m+1)^2}$. By theorem 2.5.1 we conclude

$$\begin{aligned} \frac{u(x,t)}{\delta} &= \frac{2}{\pi}\left[\sin(x) - \delta^3 \sin(3x) + \delta^8 \sin(5x) - \ldots\right] \\ &\approx \frac{2}{\pi}\sin(x), \text{ since when } t \gg 0 \end{aligned} \tag{2.29}$$

Plug (2.29) into (2.28) and recall that $\int_0^\pi \sin(x) = 2$ to get

$$\begin{aligned} \lim_{t\to\infty} \rho(x,t) &= \frac{\frac{2}{\pi}\sin(x)}{\int_0^\pi \frac{2}{\pi}\sin(x)dx} \\ &= \frac{1}{2}\sin(x) \end{aligned} \tag{2.30}$$

Thus, (2.30) is the normalized stationary arrival distribution of the survival percentage when $t \gg 0$. We plot this convergence in figure 4, which is generated using `plotNormalizedFourierSer.m`

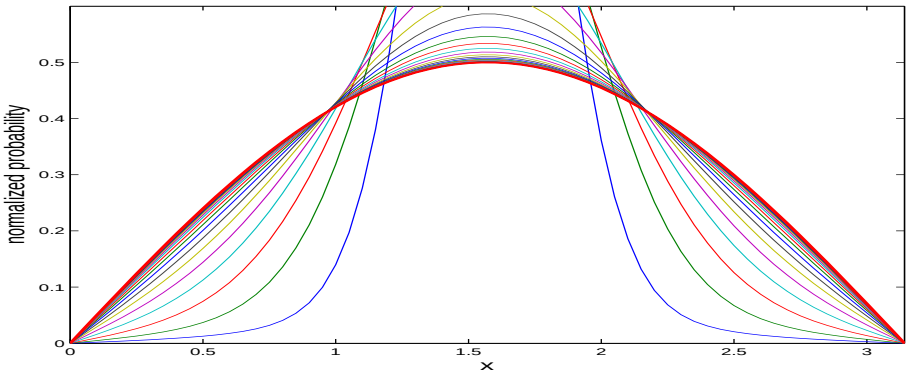

Figure 4: Plot showing normalized arrival distributions $u/\int_0^\pi u dx$. The thick red line is the function $\sin(x)/2$, and we see that convergence towards this function is rather quick (plotting for $t \in \{0.1, 0.2, 0.3, \cdots, 3.1\}$).

2.6 Scenario III: Bouncing Boundary Conditions

2.6.1 Boundary Conditions

In the last section the process is terminated when reaching any of the edges, and this termination is achieved by choosing certain boundary conditions. We will now choose different BC's to make the process *bounce* at the boundaries rather than being terminated. The basic set-up is still the same and once again we refer to the solutions of the ODE's given in (2.7) and (2.12), obtained by assuming that the heat diffusion PDE have a separable solution. We shall now choose new boundary conditions to impose the bounce, as cleverly proposed by professor Jan Boman of Stockholm University, during a conversation at the coffe table.

By differentiating (2.12) w.r.t. x we readily get

$$\Xi'(x) = -c_3 \sin(\sqrt{\lambda}x) + c_4 \cos(\sqrt{\lambda}x), \tag{2.31}$$

where $c_3 = \sqrt{\lambda}c_1$ & $c_4 = \sqrt{\lambda}c_2$ and the $'$ represent the x-derivative. Assuming that the process is bound by straight lines at $x = L$ and $x = -L$, we need to force the process never to cross these lines (boundaries). The way to do this is to put the process' x-derivative equal to zero at the boundaries, thus making the process change direction whenever it hits these boundaries. We also need to make sure the upper boundary produces a maximum and the lower boundary produces a minimum (not a saddle point).

In mathematical terms the above conditions can be written as $\Xi'(L) = \Xi'(-L) = 0$. Setting $L = \pi$ we get the BC's as $\Xi'(\pi) = \Xi'(-\pi) = 0$.

The first BC, $\Xi'(\pi) = 0$, together with (2.31) gives

$$-c_3 \sin(\sqrt{\lambda}\pi) + c_4 \cos(\sqrt{\lambda}\pi) = 0. \tag{2.32}$$

For this equation to be true it is required that either both the sin and the cos term above are equal to zero, or that $c_3 \sin(\sqrt{\lambda}\pi) = c_4 \cos(\sqrt{\lambda}\pi)$.

The latter case is trivial, since cos and sin are equal only when $\sqrt{\lambda} = \left(n + \frac{1}{4}\right)$, $n \in N$. This choice of λ would mean that the original function (2.12) is always zero, and would give the rather simple *zero-everywhere-*$\forall t$ solution to the diffusion-equation. Although simple, it is nonetheless a valid solution.

Of more interest is the case when the sine and cosine terms above are both zero. For the sine term we have

$$c_3 \sin(\sqrt{\lambda}\pi) = 0 \quad \text{if} \quad \begin{cases} \text{either} & \lambda = n^2,\ n \in N \\ \text{or} & c_3 = 0, \end{cases}$$

and similarly for the cosine term

$$c_4 \cos(\sqrt{\lambda}\pi) = 0 \quad \text{if} \quad \begin{cases} \text{either} & \lambda = \frac{(2n+1)^2}{4},\ n \in N \\ \text{or} & c_4 = 0. \end{cases}$$

The second BC, $\Xi'(-\pi) = 0$, gives the same results.

There are two[5] options to chose the coefficients from, and we have to figure out which one to use. Either we set $\lambda = n^2$ and $c_4 = 0$, or we set $\lambda = \frac{(2n+1)^2}{4}$ and $c_3 = 0$.

The first case gives

$$\Xi'(x) = -c_3 \sin(nx), \text{ and}$$

$$\Xi(x) = c_1 \cos(nx).$$

while the latter case gives

$$\Xi'(x) = c_4 \cos(nx), \text{ and}$$

$$\Xi(x) = c_2 \sin(nx).$$

2.6.2 Initial Condition

Both of the choices above satisfy the diffusion-equation with the given BC's, but it turns out only one of them will also satisfy the Dirac-Delta Initial Condition. To explain this we need the Fourier Series for the diffusion equation, and setting $A = \pi$, this is given as

$$u_\lambda(x,t) = a_0 + \sum_{n=1}^{\infty} a_n \cos(\sqrt{\lambda}x)e^{\lambda t} + \sum_{n=1}^{\infty} b_n \sin(\sqrt{\lambda}x)e^{\lambda t}. \tag{2.33}$$

To reinforce that the IC is satisfied we require that (2.33)$\rightarrow f(x)$ as $t \rightarrow 0$. That is, we want

$$f(x) = a_0 + \sum_{n=1}^{\infty} a_n \cos(\sqrt{\lambda}x) + \sum_{n=1}^{\infty} b_n \sin(\sqrt{\lambda}x). \tag{2.34}$$

$f(x)$ is chosen to be the Dirac delta function and thus we want $f(0) = \infty$[6] and $f(x) = 0 \forall x \neq 0$. With a proper IC in place, we now have enough information to proceed to choose the correct values for λ.

We start by forcing the series to satisfy the condition $f(0) = \infty$. Since the sine series are always zero at $x = 0$ it makes sense to set also the $b_n = 0$. Setting $b_n = 0$ in the Fourier series cancels out the sine series and thus corresponds to setting c_2 (and c_4) to zero in the product solution. Then, in order to avoid the aforementioned zero-everywhere-$\forall t$ solution we are forced to choose c_1 (and c_3) to be non-zeros, which means setting $\lambda = n^2$.

Choosing $\lambda = n^2$ and $b_n = 0$ is all we need to do in order to make sure $f(0)$ $\neq 0$. The sine series vanish and we are left with

$$f(x) = a_0 + \sum_{n=1}^{\infty} a_n \cos nx. \tag{2.35}$$

[5] Disregarding the case $c_3 = c_4 = 0$ since this produces the same simple *zero-everywhere*-$\forall t$ solution as the $\sqrt{\lambda} = \left(n + \frac{1}{4}\right)$ case.

[6] This is not a correct notation, and what I really mean is that $f(0) \rightarrow \infty$ as $n \rightarrow \infty$.

Approaching an acceptable solution: $\cos 0 = 1$ and $\sum_{n=1}^{\infty} a_n \to \infty$ for many sequences a_n. It now only remains to find these a_n and make sure they are not of $O\left(\frac{1}{n^2}\right)$ or less, since that would make the series converge.

Finding a_0 is easy; remembering that $f(x) = \delta_0$ we use (2.16) to see that

$$\int_{\pi}^{\infty} \delta_0 dx = \int_{-\infty}^{-\pi} \delta_0 dx = 0$$

$$\Rightarrow \int_{-\infty}^{\infty} \delta_0 dx = 1, \tag{2.36}$$

and combining (2.36) and (2.18) we get

$$a_0 = \frac{1}{2\pi}. \tag{2.37}$$

To find the rest of the a_n's we again look towards the result in (2.16). $f(x)$ is defined $\forall\, x \in (-\pi, \pi)$ and integrating (2.35) between γ and π gives

$$\begin{aligned} \int_{\gamma}^{\pi} f(x)dx &= \int_{\gamma}^{\pi} \frac{1}{2\pi} dx + \sum_{n=1}^{\infty} a_n \int_{\gamma}^{\pi} \cos(nx)dx \\ \Rightarrow 0 &= \frac{\pi - \gamma}{2\pi} - \sum_{n=1}^{\infty} \frac{a_n}{n} \sin(n\gamma) \end{aligned}$$

Rearranging this and putting $g(\gamma) = \frac{\pi - \gamma}{2\pi}$ and $c_n = \frac{a_n}{n}$ we can apply the results in (2.20)[7]. We get

$$\begin{aligned} c_n &= \frac{2}{\pi} \int_0^{\pi} \frac{\pi - \gamma}{2\pi} \sin(n\gamma) d\gamma \\ &= \frac{1}{n\pi} \text{ (via integration by parts)} \\ \therefore a_n &= \frac{1}{\pi}. \end{aligned} \tag{2.38}$$

Plugging (2.38) and (2.37) into (2.35) we get

$$f(x) = \frac{1}{2\pi} + \sum_{n=1}^{\infty} \frac{1}{\pi} \cos nx,$$

and this immediately gives us the equation for $u_\lambda(x,t)$ as

$$u_\lambda(x,t) = \frac{1}{2\pi} + \sum_{n=1}^{\infty} \frac{1}{\pi} \cos(nx) e^{-tn^2}. \tag{2.39}$$

This solution behaves as required at $x \in \{-\pi, 0, \pi\}$, but let us now confirm that it behaves as we want elsewhere.

[7] Note that $g(x)$ and $\sin x$ are both odd functions, so $g(x)\sin x$ is an even function and $\int_{-\pi}^{\pi} g(x)\sin x = 2\int_0^{\pi} g(x)\sin x$.

Extrema. The solution in (2.39) clearly satisfy the divergence criteria (page 29), since $\frac{1}{\pi} > \frac{1}{n^2}\ \forall\ n > 1$. But we also require the process to turn direction and bounce at $x = \pm\pi$, and thus we need to make sure that the process attains its maximum at π and minimum at $-\pi$. This is easily checked with some elementary calculus: First we verify what happens with the x-derivative at $\pm\pi$

$$\begin{aligned} u_x(x,t) &= \sum_{n=1}^{\infty} \frac{-n}{\pi} \sin(nx) e^{-tn^2} \\ \Rightarrow u_x(\pm\pi,t) &= 0 \Rightarrow \text{turning point.} \end{aligned}$$

Thus $\pm\pi$ is a turning point. To verify maxima and minima, we look at the sign of the second derivative

$$\begin{aligned} u_{xx}(x,t) &= \sum_{n=1}^{\infty} \frac{-n^2}{\pi} \cos(nx) e^{-tn^2} \\ \Rightarrow u_{xx}(\pi,t) &= \sum_{n=1}^{\infty} \frac{-n^2}{\pi} \cos(n\pi) e^{-tn^2} < 0\ \forall\ x \Rightarrow \text{maximum at } \pi. \\ \Rightarrow u_{xx}(-\pi,t) &= \sum_{n=1}^{\infty} \frac{-n^2}{\pi} \cos(-n\pi) e^{-tn^2}, \text{ and cosine is even so} \\ &= \sum_{n=1}^{\infty} \frac{n^2}{\pi} \cos(n\pi) e^{-tn^2} > 0\ \forall\ x \Rightarrow \text{minimum at -}\pi. \end{aligned}$$

Total probability. The last criterium is the law of total probability should always be satisfied. Since this process bounces whenever it reaches a wall the total probability implies that, for every $0 < t < T$, the integral between the boundaries, $x = \pm\pi$, shall equal one. Again, this is easily verified using simple calculus;

$$\begin{aligned} \int_{-\pi}^{\pi} u(x,t)dx &= \int_{-\pi}^{\pi} \frac{1}{2\pi} dx + \sum_{n=1}^{\infty} \frac{1}{\pi} \left[\int_{-\pi}^{\pi} \cos(nx)dx \right] e^{-tn^2}. \\ &= \left[\frac{x}{2\pi} \right]_{-\pi}^{\pi} + \sum_{n=1}^{\infty} \frac{1}{\pi} \underbrace{\left[\frac{\sin(nx)}{n} \right]_{-\pi}^{\pi}}_{=0} e^{-tn^2}. \\ &= 1. \end{aligned}$$

Thus, equation (2.39) satisfies all our requirements and is therefore a accurate solution for the heat diffusion PDE with given BC's and IC.

Let us formulate the above as a theorem:

Theorem 2.6.1 (Bouncing Boundaries) *A non-trivial solution of the equation $u_t = \frac{1}{2}u_{xx}$, with boundary conditions $u_x(-\pi) = u_x(\pi) = 0$ and initial*

condition $u(x,0) = \delta_0$*, is given by*

$$u(x,t) = \frac{1}{2\pi} + \sum_{n=1}^{\infty} \frac{1}{\pi} \cos(nx) e^{-tn^2}.$$

This final solution can be interpreted as the probability of being at position x at time t, and it is illustrated in Figure 5[8].

Remark: All processes survive and thus the area under the the distribution functions displayed in figure 5 are already normalized. The limiting distribution is easily found, since if $t \gg 0$ the sum $\sum_{n=1}^{\infty} \frac{1}{\pi} \cos(nx) e^{-tn^2} \ll 1$ and hence $u(x, t \gg 0) \to \frac{1}{2\pi}$. (This is very reasonable since a unit size rectangle of width 2π must have height $1/2\pi$.)

[8] Generated via `sumBouncingFourierSer.m` and `plotBouncingFourierSer.m`

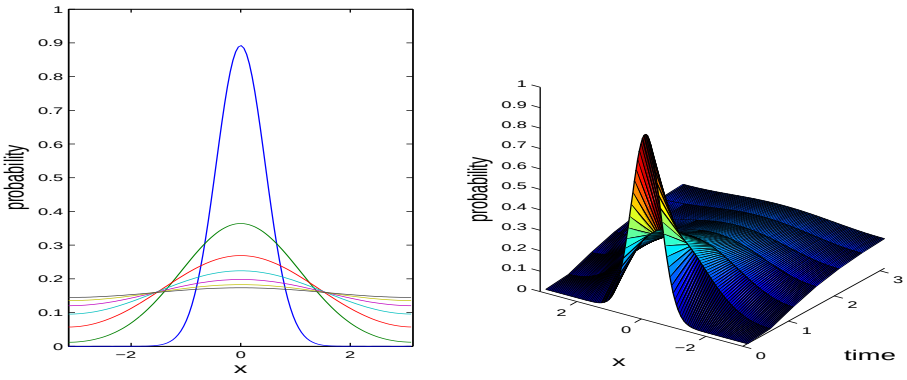

Figure 5: Plotting the solution in Theorem 2.6.1 for n=1,. . .,1000, and for discrete times $t \in \{0.1, 0.6, 1.1, \ldots, 3.1\}$.

3 Monte Carlo Solutions to Previous BVP's.

In chapter 2 we described the connection between the random walk and the diffusion equation and managed to solve the diffusion equation analytically for three distinct sets of BC's. Through this we know that a large number of random walks will generate essentially the same arrival probabilities as the corresponding diffusion equation. Thus, we do not really need to solve the diffusion equation analytically, since it suffices to simulate a large enough[9] number of drunken men.

If the *large enough* is *small enough* to fit in the computer memory, typically a few hundred thousand simulations on a field that is 1000 steps long can be computed in only a few minutes on an ordinary PC. In what follows, I will show a few examples of stochastic simulations of such boundary value problems, and for the three cases solved in chapter 2, I also perform and compare to the Monte Carlo counterpart solution.

When performing the simulations I have been using either my laptop with 512mb memory and a 1,3GHz processor, or my home PC with 1024 megabyte memory and 1,8GHz processor. The MATLAB student version, release 14, was used for the simulations and the source code can be found in the appendix.

It should be noted that the code and design ideas within are derived from first principle. This might seem as drawback to some, but I wanted to experiment how well one could solve these problems without expert knowledge. Hence, the methods within should be easy to follow for anyone with a little knowledge in programming and mathematics.

The point I try to emphasize by these simulations is the possibility of solving PDE's by doing repeated stochastic experiments. The main focus will be on the arrival distributions of the process which, after normalization, can be interpreted as the probability distribution of the arrivals at the far end of the street.

3.1 Basic Design Ideas.

Let me start by guiding you through the general structure of the MATLAB code, and then the specific's will be given when I deal with the matching simulation. The code begins with the definition of the crucial variables. If there are boundaries, they will be defined as $N_+ \& N_-$, where $N_\pm$ can be either a constant or a function of t. Often the length of the road, field or whatever you like to call it, will be predetermined and we denote this by T[10]. The number of simulations to be run is set to S. The time-spatial location on the road is often referred to in coordinates (x, t), where x is the spatial distance from the center line and t is the time the process has been running since the start. Setting $x_0 = \frac{N_+ + N_-}{2}$ the center line is defined as (x_0, t), where $t \in (0, T)$.

[9] Exactly how large *large enough* is, is not given too much thought in this text. I just chose a "large enough" that I think will do the job

[10] For simplicity I also set the number of time steps to equal T. This may not be normal practise, but it makes the binomial tree-structure of the process easier to explain and the code will be much clearer.

We create a vector named Pos of length S. Each element in Pos represents the current spatial position ($\pm$ the distance from the center line) of the corresponding simulation. At $t = 0$ all processes will start at x_0 and thus we get the position vector in the first time step as $Pos = x_0 * (1, 1, 1, \ldots, 1)^{\mathrm{T}} \in \mathbb{R}(1 \times S)$.

We iterate through all the discrete trime steps, which means we will go through the loop T times. At each time step $t \in (0, 1, \cdots, T)$, we create a vector v_t of length S where each element is either a $+1$ or a -1 such that $P(v_t) = 1 = P(v_t) = -1 = 0.5$. Each element of v_t represents whether the corresponding simulation is to step up one, or down one, at this time step. We then add this vector to the position-vector Pos and receive a new position vector that we again call Pos. We repeat this T times. The last version of Pos to come out of the iteration contains the arrival positions for each and every simulation after T time steps. If there are boundary conditions, we perform a check in every iteration to see if any process have reached a boundary, and if it has we apply the corresponding action.

It can also be noted that for the most trivial case of random walks, the case without boundaries, one can simply generate a large matrix of ± 1's and then sum all the columns[11]. This way do not need any iterations, which makes for a very fast code, but it only works good for matrices small enough to fit comfortably in the computer memory.

To obtain $v_t = \pm 1$ I have been using two different methods: the first using uniform and the second using binomial random variables:

Uniform: Take $\theta \sim Un(0, 2)$ and then truncate the decimals, leaving only ones and zeroes. These will occur with the same probability so we need to take $2 * \theta - 1$ to obtain ± 1 with equal probabilities of 0.5.

Binomial: $\theta \sim Bin(1, .5)$ will give us 0's and 1's. $2\theta - 1$ will then be ± 1 with probability .5 respectively.

Through rather unsophisticated experiments I have come to the conclusion that the binomial method is a bit less computer intense, and hence a bit faster, than the uniform method.

In MATLAB it is possible to randomize whole vectors at the time, which is very effective to minimize computational time.

Even though some of the examples contained are Markovian and thus could be simulated using *transition probability matrices*, I chose to use the same style for all the simulations.

3.2 1D Random Walk - Scenario I

The most basic and traditional example is the one dimensional random walk. The drunken man starting at a point $(t = 0, x = 0)$ to walk a pre-specified number of steps until he arrives at some location at the final time T. According to chapter 2 the probability of arriving at (ξ, T) is equal $f(\xi, t)$, where f is the

[11] Using, for example, the `cumsum` function in matlab

probability density function of a normally distributed random variable with zero mean and variance $\sqrt{T}$.

One can actually simulate the above random walk without iterating through the code: Create a matrix with the size #simulations$\times T$ and fill it with randomly chosen $\pm 1 : Prb(+1) = .5 = Prb(-1)$. By taking the cumulative sum[12] in the T direction we add the T ± 1's onto each other, just as the case of a discrete random walk.

Storing the paths in a matrix makes the calculation in MATLAB very efficient, but the amount of memory used by the computer will increase quadratically with the number of simulations, and thus the computer will eventually run out of memory. It seems that the best way would be to generate a matrix that is a bit below what the computer can handle, and then iterate this many times.

I did a simulation of the process and at the same time compared it with the corresponding solution to the unbound diffusion equation, using the density function for $N(0, \sqrt{T})$. The solutions are illustrated in figure 6, generated by `simulate1dRandomWalk.m`.

3.3 1D Random Walk with Terminating Boundaries - Scenario II

Recall the boundary conditions for the diffusion PDE with terminating boundaries were defined for $x = \{0, \pi\}$ and $\forall t$. This BVP is very simple to solve using Monte Carlo random walks and the algorithm will now be presented.

After defining the variables(a number of discretee timesteps T, the distance from the centre line to the boundries $\pm N$, the number of simulations Sim) we create a $Sim \times N$ size matrix called Pos that contains random ± 1's. We take the cummulative sum, in the T direction, of the Pos matrix and then take the cummulative sum (still in the T direction) of all the elements of Pos that does not exceed the boundary values $\pm N$. Then we plot the arrival-locations against the arrival percentages. The plot is shown in figure 7 and is generated using `simulate1dRandomWalkTermBC1.m`.

3.4 1D Random Walk with One-step Bounce - Scenario III

When solving the diffusion PDE for the bouncing boundary conditions, we put the spatial derivative $u_x = 0$ at the boundaries to make sure the process stays within the boundaries. For the stochastic simulations, I solve the analogous problem by forcing the process to have the probability 1 of stepping away from the boundary, whenever it hits the boundary. This is illustrated in figure 8, which is generated by `simulate1dRandomWalkBounceBC1.m`. It shows how the arrivals do indeed converge towards the uniform distribution, as T gets larger.

[12] In MATLAB the command `cumsum` does this without iteration.

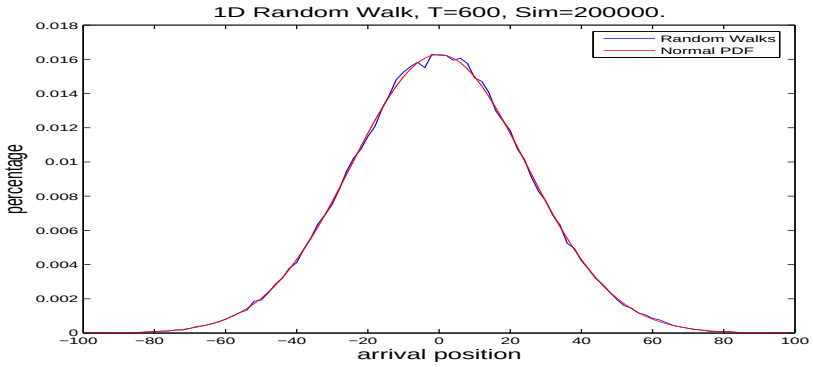

Figure 6: Random Walk versus Normal distribution. Computational time: 30.1sek on my laptop.

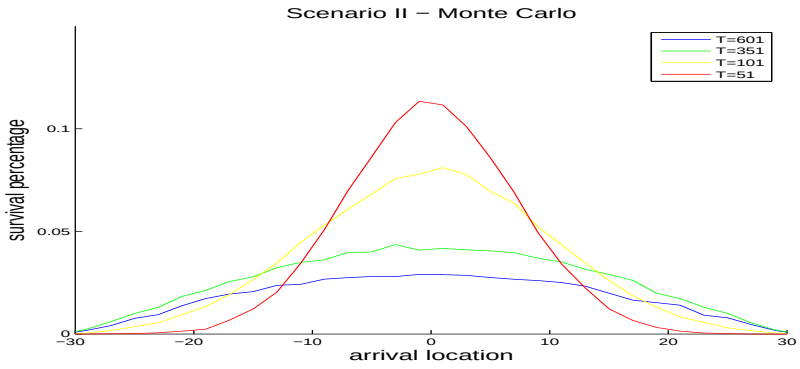

Figure 7: Terminating boundaries. I chose the field to be $2N$ wide where $N = 30$, and plotted for $T \in \{51, 101, 351, 601\}$. The un-smoothness of the curves is due to the fact that we use unit steps, which could be compensated for by using a much wider field. Computational time for 30000 simulations =141 seconds (on my laptop).

The ups and downs retain the Markovian property[13] and are independent of each other. We start as above and generate a matrix of ± 1's and then take the cumulative sum of these to obtain the unbound random walk. Then we iterate through all time steps and check at each time step, whether or not, the process is at (or exceeds) the boundary level $\pm N$. If so, we subtract/add 1 to all the remaining time steps of this process, thus pulling the process up/down according to the bounce.

Since the drift is introduced for one step only, I call this process a *one-step bounce* process. Obviously, when $T < N$ this would be exactly the same as in Scenario I (since it is impossible for the process to reach the boundary for $T < N$). As time increases, the arrival probabilities will converge towards the uniform distribution in accordance with the corresponding solution of the boundary value problem.

[13] A process is called a Markov process if the next step is completely determined by the current position of the process. This is true in this process since we step up/down one step only if we are at the boundary, and otherwise we keep going as usual. This property is also called the memory-less property and one think of the drunken man Erik hitting a wall next to the street. He takes one step away from the wall, but he is so drunk that he immediately forgets the existence of the wall, and might step right back into it.

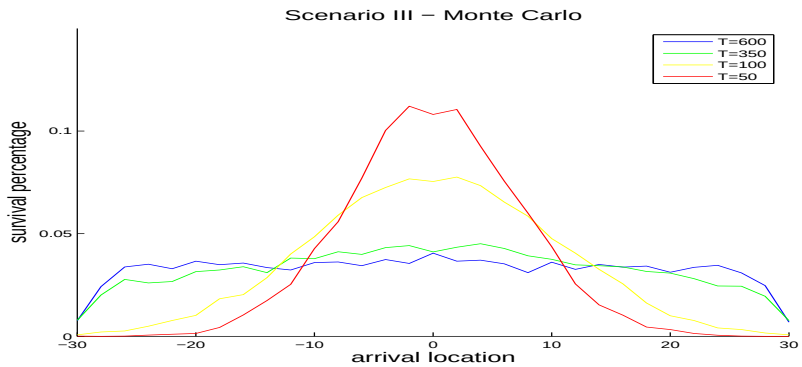

Figure 8: Bouncing boundaries. I chose the field to be $2N$ wide where $N = 30$, and plotted for $T \in \{50, 100, 350, 600\}$. The un-smoothness of the curves is due to the fact that we use unit steps, we could compensate for this using a much wider field. However, this plot illustrates the main point. Only 10000 simulations were done and thus the graphs are rather *stochastic*. The code used for this application is "raw" and not optimized in any way, and thus computational time = 650 seconds on my laptop.

4 Drifts, Bounce Counting and 1.5D.

4.1 1D Random Walk with a Target Fixation Drift

Imagine that we want to introduce a drift to the random process. The most common solutions are to consider a stochastic differential equation (SDE) of the form

$$dX(t) = \alpha(t)dt + \sigma(t)dW(t), \tag{4.1}$$

where $dX(t)$ represents the change in X (distance from the center line) as the process takes a tiny deterministic step $\alpha(t)dt$, plus some stochastic diffusion term $\sigma(t)dW(t)$. For some systems this can be solved rather easily using standard techniques of SDE's, but it is often easier to solve the problems using Monte Carlo simulation, where the drift will correspond to changes in the probabilities of stepping up and down.

Imagine that the drunken Erik walks along a road and all of a sudden he stumbles into the sidewalk. In the confusion, he looks up and sees the beer can at the end of the street. From now on Erik knows where the beer can is and he is aiming for it. Although still walking very drunkenly, we have introduced what we choose call a *target fixation drift*, i.e. a drift in the direction of the beer can.

Assume that $x_0 = 0$ and $|N_-| = |N_+| = N$, where N is the distance from the barrier to the center line. In the model of target fixation the drift is introduced when the process hits the barrier, and therefore denote by t the hitting time of the barrier. The target fixation drift would then equal the slope the process need to walk along to arrive at $(0, T)$. This slope would be $\frac{-N}{T-t}$, as shown in figure 9.

Note: Due to the structure of the binomial tree this would only be accurately described, when considering the first $T - N$ steps. If the process hits the boundary in any of the last N steps the process would have zero possibility of reaching $(0, T)$, even though the diffusion equation describing the process would allow this, and thus we would have introduced an extra constraint on the system which is not compatible with the diffusion equation.

In what is not the limit, but the unit (since we simulate discretely) we introduce the drift by altering the simulated probabilities according to the reasoning above. This is explained in figure 10. We no longer want $P(v_{t+1} = 1) = P(v_{t+1} = -1) = 0.5$ as was used in the previous, non drifted, cases. Instead, we choose $P(v_{t+1} = -1) = \frac{1+N/(T-t)}{2}$ and $P(v_{t+1} = +1) = \frac{1-N/(T-t)}{2}$, where v_t is the step vector for time t. The law of total probability holds, since $P(v_{t+1} = 1) + P(v_{t+1} = -1) = \frac{1+N/(T-t)}{2} + \frac{1-N/(T-t)}{2} = 1$, which is the first check one should perform when dealing with anything probabilistic, just to make sure one is not outside, riding a bike.

I have simulated the solution of two different systems, one where the drift is taken away when the process crosses the center line, and one where the target fixation drift is still intact. See figure 11, which was generated by `RandomWalkLookBackVect1.m`.

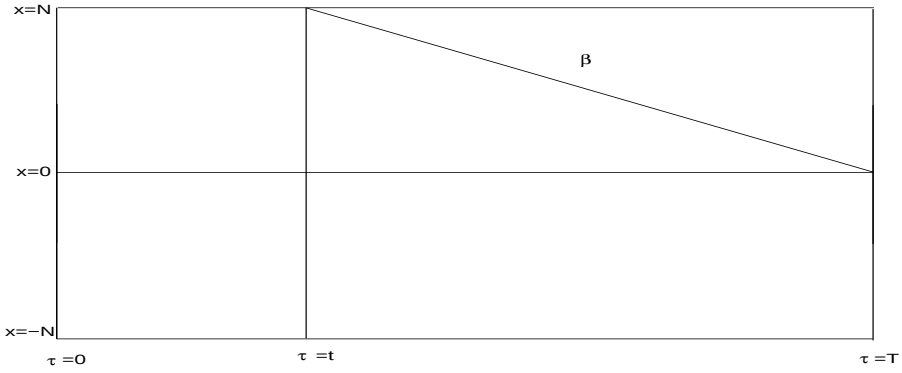

Figure 9: The slope of the line β from (t,N) to (T,0) is $\frac{-N}{T-t}$. This is chosen to represent the drift of the process, i.e. would the process behave deterministically ($\sigma(\tau) = 0 \forall \tau \in (t, T)$ in (4.1)) it would arrive at $(T, 0)$. When $\sigma(\tau) \neq 0 \forall \tau$ this drift would imply that the mean of a large sample of arrivals would be approximately (T,0).

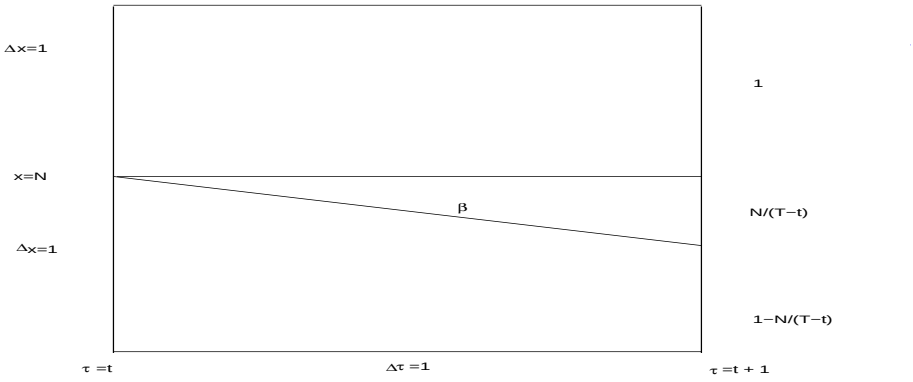

Figure 10: Altering the probabilities of stepping left and right to match up with the drift-line β. Basically what we want is for half of the simulations to end up beneath the β line and half of them above. So, P(above β)=P(below β)=0.5. However, we are using discrete time steps and we only have the two choices, either we end up in $+1$ or we land at -1. Therefore, we need to alter the probabilities so that $P(v_{t+1} = -1) = \frac{1+N/(T-t)}{2}$ and $P(v_{t+1} = 1) = \frac{1-N/(T-t)}{2}$. By doing this, we essentially make sure that half of the time we land above β and thus at $+1$, and half of the times we land below β at -1. To understand that this is correct, recall that the β-line is the drift for the process, and half of the processes should end up on each side of β. Essentially all we are doing is adding an extra probability (equal to the drift coefficient) to follow the direction we are drifting.

4.2 1D Random Walk with a Look-back Drift

Let the process, when reaching the wall, look back to where it started and then introduce a drift equal to the angle at which the process reaches the wall - just like a rubber ball would bounce on the wall if thrown from the starting point[14]. If the process reaches the wall again a new drift is calculated, still based on the angle from the start position. Thus, if the process reaches the wall at time t the angle would be N/t (and hence the drift coefficient should be $-N/t$).

Two results are presented: one where the drift is taken away when (if) the process crosses the middle-line, and one where the drift is left until the end (or when the process reaches a wall again). Obviously, if the drift is taken away at the center line the process would be re-set and again have zero expected value, hence this process ought to be more centered than the process whose drift is never taken away.

Important Remark: Due to the nature of the drifts these processes no longer attain the Markovian Property[15] and can thus not be modeled using a transition probability matrix as described in appendix A. However, after some modification,the Monte Carlo simulation approach is still applicable and valid.

The graphs of the look back drift cases are shown in figure 12 (generated via `RandomWalkLookBackVect1.m`).

4.3 Bounce Counting Function for Target Fixation

We now alter our setup in a new fashion, according to a later study of *the lighthouse* (see chapter 5) . Instead of just adding new walls (BC's) we want to gather data from the process. So using the target fixation drift case above, and introduce a function calculating the number of times a process reaches the wall.

We set the field to d wide and T long. Our aim is to find a function $\Phi(d, T)$ describing the average number of bounces for a process in a field of this size. The way this is done is by simply iterating through the above code for different values of T and recording the average number of bounces for each T. The results are shown in Fig. 13 and were generated using the files `countBounces2.m` and `plotBounces2.m`.

Above the number of bounces varies with the length of the field for a fixed width, whereas now the relation will be fixed between the length and the width. We define the length to be $T = 10 * N$ and then we vary the N. The results of this experiment are shown in Fig. 14 and were generated using the files countBounces2.m and plotBouncesCTSapprox.m.2

[14] Of course the actual angle at which the process reaches the wall will always be 45 degrees since we simulate using a binomial structure. Clearly this *infinitesimal* angle is of no importance to us. Instead we consider the average angle at which the process have been traveling since the start.

[15] A process is said to be Markovian, or a Markov Chain, if the next step is dependent on the current position only. Since the drift introduced here (as well as in the target fixation case) depends on when (if) the process hits the wall this is no longer a Markov Chain

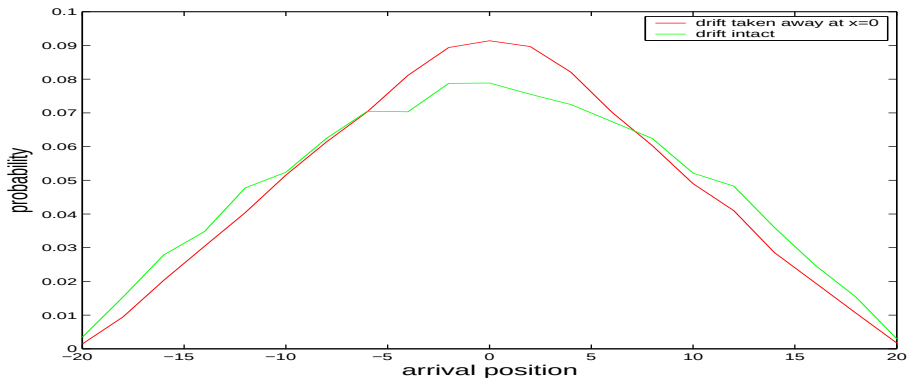

Figure 11: Random Walk target fixation drift. We see that if the drift is taken away when the process crosses the centre line we get a more centralized arrival distribution. 2000 steps and 15000 simulations, took about 15sec each on my PC.

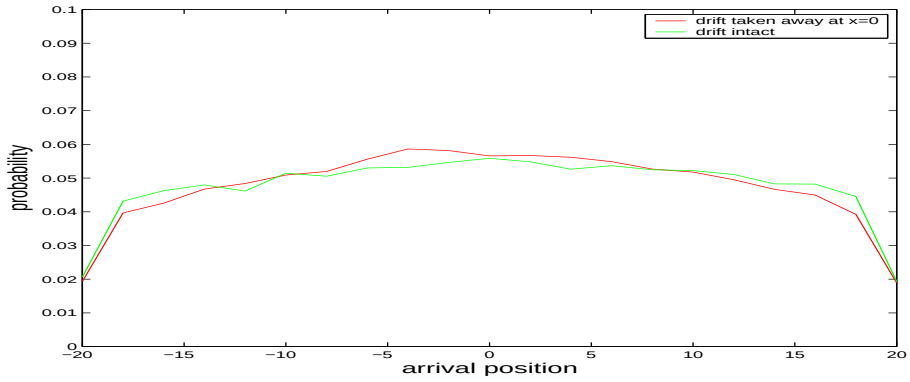

Figure 12: Random Walk look-back drift. There is not a great deal of difference when we take away the drift at $x = 0$ and when we leave it be. 999 steps and 30000 simulations took about 15 sec per graph on my PC

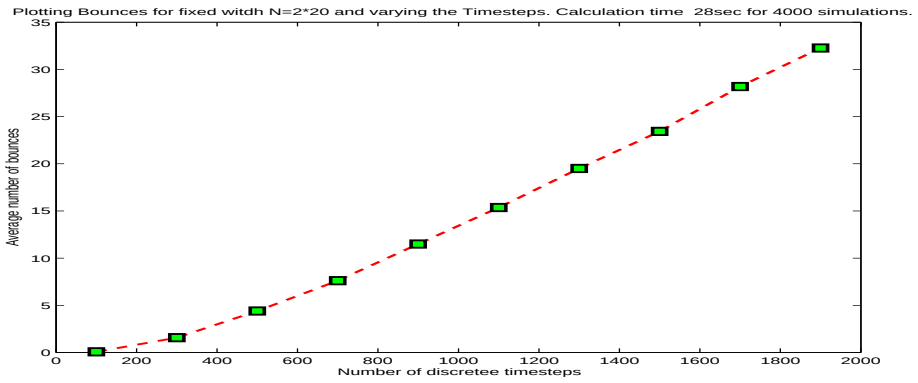

Figure 13: Plot of $\Phi(N,T)$ with fixed $N = 2 * 20$ and varying T. As one might expect, the longer the field is, the more times the process will bounce.

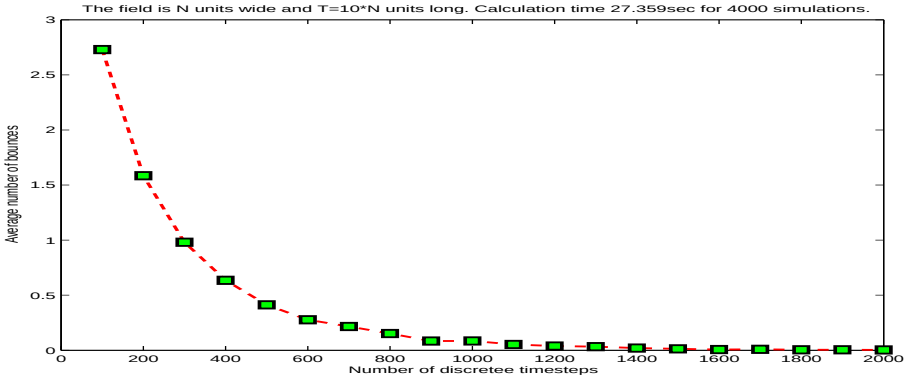

Figure 14: Plot of the number of bounces when we change the width N and the number of time steps according to $T = 10 * N$.

4.4 1.5D Random Walk

So far the processes treated has been moving according to a 1D Brownian Motion which basically means one step forward at and one step either up or down for every time step. We have been concerned with the distribution of the arrival locations of these processes whilst the arrival times have been fixed. Let us now change the process so that at each time step the process *either* goes straight forward ($Prb = \frac{1}{3}$), left ($Prb = \frac{1}{3}$) or right ($Prb = \frac{1}{3}$). We also define the field in which the process is moving in to be twice as wide as it is long and the boundaries to be terminating. The process starts at ($x = \frac{\text{field width}}{2}, t = 0$) and due to this new setup it will take far longer than the earlier setups, in fact in average three times longer than before, to reach the end of the field.[16]

To begin with let us plot the arrivals at the far end of the street. This 1.5D Monte Carlo simulation is much more computer intense than the 1D version and thus we constrain ourselves to a field that has size 100×50 steps and let the process start at $(x = 50, t = 0)$. Figure 15 shows a plot of both the arrival positions and the time it takes to arrival. The arrival positions assume a normal distribution (just as for 1D case) while the arrival times seem to be skewed.

Two arbitrary boundary intervals are chose, one at the far end of the box and one at the left side of the box.

First we choose a segment somewhere on one of the walls and record the processes that reaches that wall-segment. Both the time and the location are recorded and are plotted in 16.

After extending the field to be twice as long as it is wide, that is (100,50), I counted the processes being terminated in a segment on the left side, and the results are presented in Figure 17. The length of the field had to be extended to increase the standard deviation of the field enough to let a sufficient number of processes pass the chosen segment. However, there's still not a lot of processes passing through, but there is at least some processes passing the gates (in contrast to when tried with the field size used in the above examples, where zero processes passed through the selected interval).

[16] Please note that I will not do any further analysis on the results in this section as the 1.5D Brownian motion is included only as an illustration of the concept.

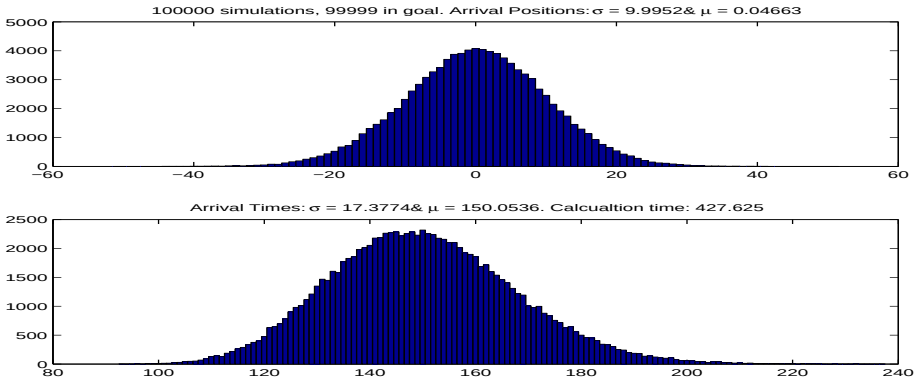

Figure 15: 1.5D random walk, when hitting the far end of the box. The box is 50steps long and 100steps wide and the process starts at (50,0). We count all processes hitting the far end and as you can see there's only a 0.001% chance that the process doest not arrive at the far end of the street, and instead hits a wall somewhere on the way.

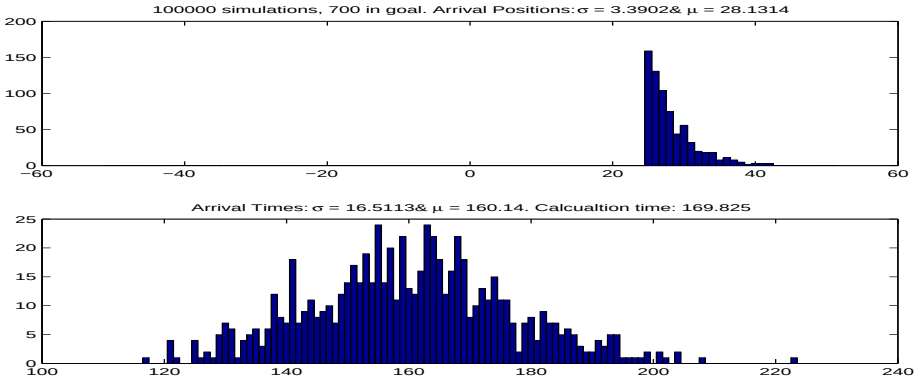

Figure 16: 1.5D random walk, when hitting the far end of the box. The box is 100steps wide, 50 steps long and the process starts at (50,0). Where only count processes that arrive within the interval $[25, 45]$.

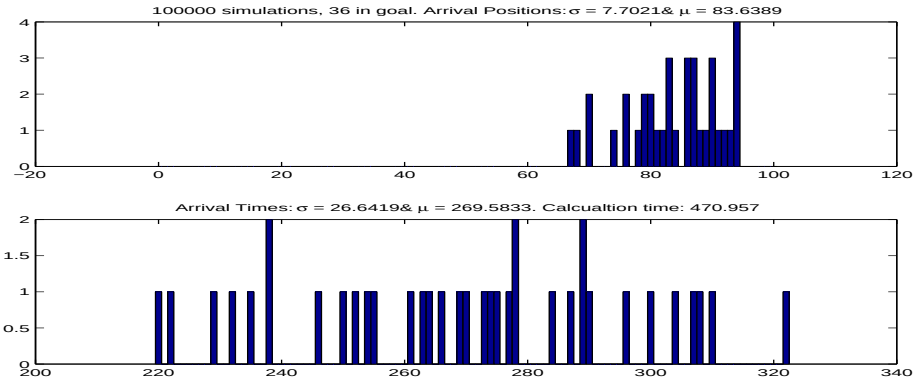

Figure 17: 1.5D random walk, when hitting the left side interval $[55, 95]$ of the box. The box now has size 100×100 and the process starts at (50,0). As you can see there's not too many processes passing this interval and to be able to get any accuracy one needs to do substantially more simulations.

5 The Lighthouse

5.1 Coastal Harbour

Let us travel by imagination back to the olden days: the sun has set and we are out at sea, on our way to a harbour on shore. After nightfall it is dark , but luckily the old lighthouse-keeper Bill has just finished his breakfast and every now and then the gas-light flashes before our eyes and we can steer in its direction. The lighthouse is situated just outside the harbour and if we can make it to the lighthouse we are safe and sound. Unfortunately, there are currents and waves and our route towards the lighthouse is rather stochastic. An immediate question is "how often would Bill have to flash the lights for us to arrive in the harbour?" and we now seek to find the answer using stochastic simulation.

We can make a Monte Carlo simulation of the night-sailing-scenario by using a mind framework similar to that above; Let us have a field of length T that has no boundaries. Whenever the lighthouse flashes a drift towards the lighthouse is introduced, and this drift is calculated in the same manner as in the target fixation case above. The aim of the simulation is to find a value of how often the lighthouse should flash (drift be re-calculated) for us to have a 95% chance of arriving in the harbour.

This simulation and the results can be found in Figure 18. The Matlab code used is found in `firehouse1.m` and `plotFirehouse1.m`.

5.2 Harbour at the end of a Norwegian Fjord

Not all sailors have the luxury of harbouring directly by the seaside. Sometimes, especially in Norway with it's lovely landscape, the sailors have to sail through a long channel before anchoring. If we imagine this channel to be straight and having the harbour centralized at the far end, we can perform simulations similar to those of the drunken man walking on the bridge, but with the extra feature that the lighthouse is flashing every so often. The aim is once again to find the required flashing frequence to safely get into the harbour.

This is easily found using the well defined framework, by plotting the arrival-percentage in the harbour against the number of lighthouse-flashes, as shown in figure 19. The plots are generated by `firehouseFjord1.m` and `plotFirehouseFjord1.m`.

5.3 A wider fjord is easier to sail through

In the last section we choose a fjord-width ad-hoc and used the same width N for all times T. Clearly a more narrow fjord would need more flashes in order for the boat to arrive safe and sound into the harbour than the wider fjord would need. We are led to investigate some sort of relationship between the probability of arriving at the end of a fjord and the width of the fjord (for some specified length T).

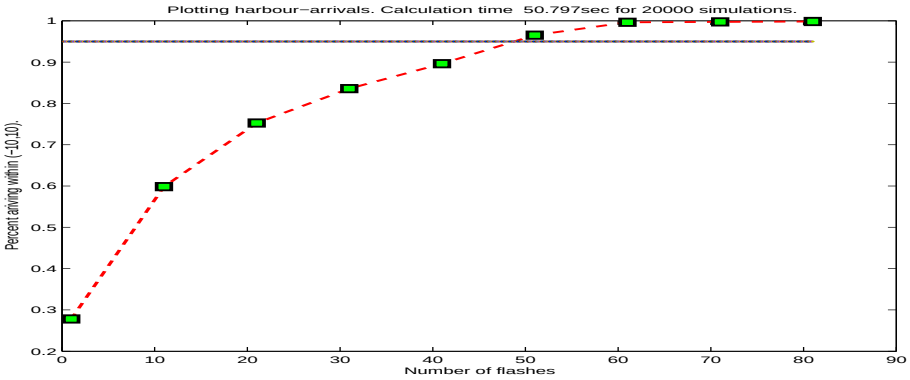

Figure 18: Plot of harbour arrivals. There are no boundaries but drift is introduced, in the direction of the lighthouse, every time the light flashes. We see that in order to arrive in the harbour with a probability of 95% or more, we need to have a bit more than 50 flashes. This time around we did 20000 simulations and the distance from the ship and the harbour was initially 400 trime steps, and the ship was initially located right outside of the harbour. (The simulations took 50 seconds on my home PC.)

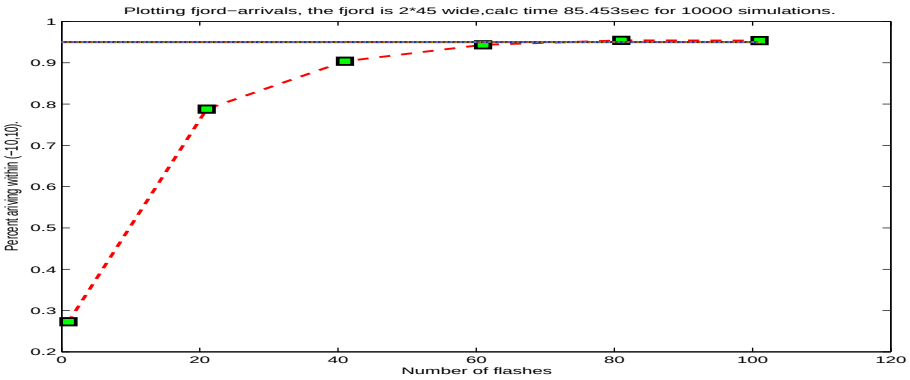

Figure 19: Plot of harbour arrivals at the far end of the Fjord. There are boundaries, rocks and such, at ±45. Drift is introduced in the direction of the lighthouse every time a the lighthouse flashes. We see that in order to arrive in the harbour with a probability of 95% or more, we need to have about 70 flashes. This time around we did 10000 simulations and the distance from the ship and the harbour was initially 1000 time steps. (It took 85 seconds on my home computer. Note that in the unbound case we had only 400 time steps whilst here we have 1000.)

Suppose that we want a 95% probability that the process (boat) will make it through the fjord and into the harbour. How many flashes are then required to satisfy this 95% condition for different width fjords? Let us call the obtained result the *optimal flash interval curve*. The results are obtained by simply looping through the code for the previous example and quit as soon as we reach a probability of survival ≥ 0.95. This is a very time consuming calculation. The plot can be seen in figure 20, and was generated by `plotFirehouseFjordWidthCI.m` and `firehouseFjordWidthCI.m`.

It seems reasonable that the number of flashes should be monotonically decreasing as the field width increases. The curve in figure 20, however, is not always decreasing why we are led to doubt it's accuracy. However, testing accuracy is outside the scope of this thesis.

5.4 The boundaries are linear but at an angle

Up to now we have concentrated on linear boundaries parallel to the centerline. These boundaries are sometimes easy to model with the differential equation approach used above. In reality one often wants to use other boundaries, boundaries that might be more complicated to model with non-computational mathematics, and thus we refer, once again, to the drunken old man Erik for our results.

We start by assuming the road is 10 times as long as it is wide at the start, the width of the beginning of the road to be $2N$ (and thus it is $20N$ long). We also define a goal, in the same fashion as in the 1.5D case above, of width $2M$. For a process to survive we require that it passes through $(-M, M)$ at $t = T$. See Figure 21 (generated by `plotSmallerGoalRW.m` and `smallerGoalRW.m`) for a plot of survivals for different goal-widths when there is no terminating barrier.

By introducing a terminating barrier from $(0, N)$ to (T, M) (and one from $(0, -N)$ to $(T, -M)$) we aim to find the probability for a process to survive until time T. Obviously we expect that more processes are being terminated when there are terminating boundaries than when there are no terminating boundaries. Hence we require the solution curve for this setup always to be below the unbound-case-curve obtained above. Of course the survival rate is dependent not only on N and T but also on the width of the goal M, hence we aim to plot the survival percentage as a function Ψ from $M = 0$ to $M = N$. The plot is shown in Figure 22, which is generated with the files `angleBarrier.m` and `plotAngleBarrier.m`.

It is clear that the function $\Psi(M)$ is an increasing function in M and that it will reach its max as $M = N$. I.e. the wider the goal, the more processes will score.

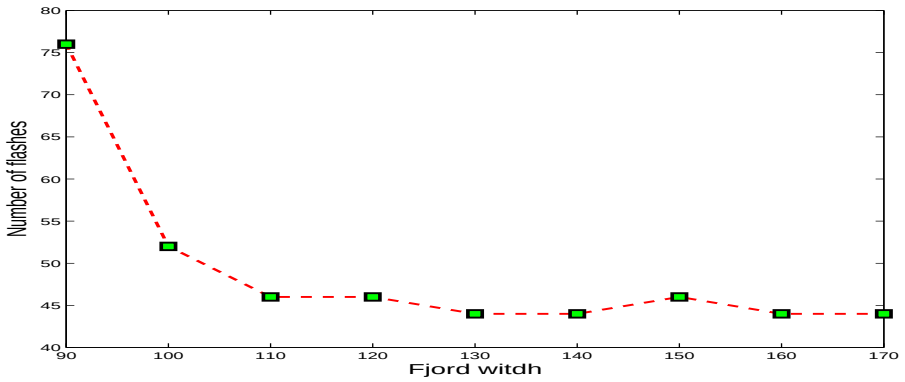

Figure 20: Plot of how many flashes is needed to have a 95% chance of arriving in the harbour for fjords of different width. I simulated 10000 ships for each number of flashes and each fjord width. I've used a fixed time $T = 1000$, and it took 24 minutes to run on my PC.

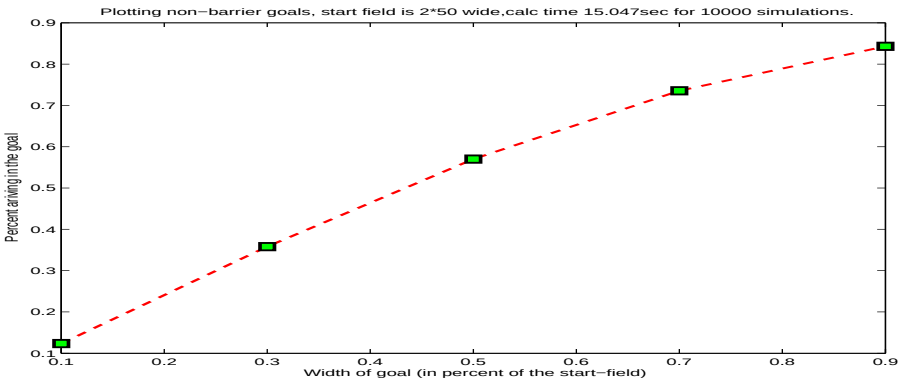

Figure 21: Plot of survival-percentage for different widths of the goal when there is no terminating barrier. The Field is 100 wide and 1000 long, and the goal-sizes range from 10,30,...,90.

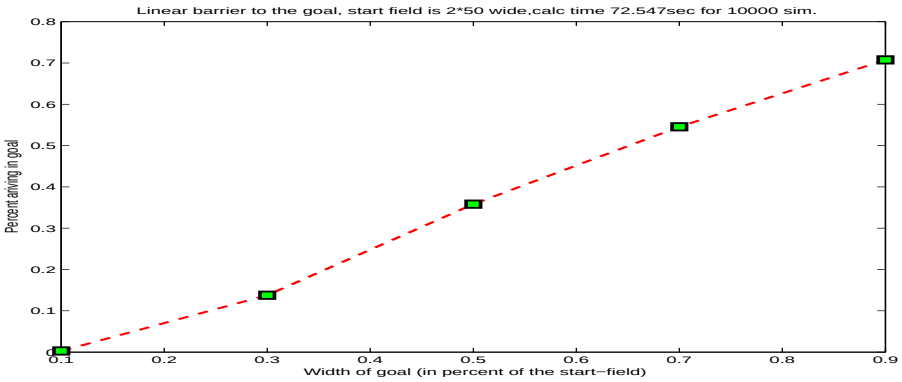

Figure 22: Plot of survival-percentage for different widths of the goal when there is a terminating barrier from $\pm N$ to $\pm M$, where $2 \cdot N$ is the width of the field at the start and $2 \cdot M$ is the width of the goal. The Field is 100 steps wide at the start, 1000 steps long, and the goal-sizes range from 10 to 90 in intervals of 10.

6 Stochastic slalom

The previous sections dealt with different examples of how Stochastic simulations can be used to solve problems related to the diffusion equation, and in some cases we have also shown the equivalence between the solution obtained via Monte Carlo simulation and the analytical one. This chapter will continue in the same manner, and as the title implies deal with *Stochastic Slalom*. To explain the principles of Stochastic Slalom I use one of my childhood friends, Filip. Filip is a professional, Olympic, snowboarder who specializes in Slalom snowboarding. In the rules of the International Olympic Committee (IOC) it clearly states that one have to be sober when participating in the competitions, but if Filip, for some strange reason, would choose to break these rules and get awfully drunk before a competition he might constitute a good example of a stochastic slalom.

Simply put Stochastic Slalom is when there are obstacles on the course that will terminate the process if hit - much like the drunken slalom-snowboarder who would loose his race if he was drunk and missed a gate. This chapter will start by explaining mathematically how to solve the simplest case and then expand to more complex cases using Monte Carlo.

6.1 Unbound Field of Snow

6.1.1 Preamble: One Obstacle

Imagine a side-wise unbound field of the length T and a process starting at $(t = 0, x = 0)$. Between $(t = T/2, x = -1)$ and $(t = T/2, x = 1)$ there is a terminating obstacle (perpendicular to the (zero) drift of the process).

Let us start by deriving an analytical solution which will later be confirmed via some Quasi Monte Carlo simulations[17].

Mathematical Statistics Approach

Recall the fact from mathematical statistics stating that *"The probability distribution of the sum of independent random variables is equal to the* convolution *of their individual probability distributions"*. We start by decomposing the process into two parts, the first being $t \in (0, \frac{T}{2})$ and the latter being $t \in (\frac{T}{2}, T)$. The second part will then be independent of the first in the sense that the processes are Wiener processes and as such future movement is dependent only on the current location of the process and not on how it got there.

At $t = \frac{T}{2}$ the process will arrive according to a normal distribution with mean zero and variance $\sigma^2 = \frac{T}{2}$, i.e. with a density function $\Phi(x, T/2) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}}$. To figure out what will happen at $t = T$ we consider the shape of the water grave. Since the random walk has zero drift we can assume that those processes that

[17] We have already performed a bunch of "normal" Monte Carlo simulations, and so I figured it might be more fun to do some clever abbreviation of the algorithm. The "clever algorithm abbreviation" is what i call Quasi Monte Carlo.

miss the barrier, i.e $\forall t = \frac{T}{2}, x \in (-\infty, -1] \bigcup [1, \infty)$, will re-start as a random walk in $(\frac{T}{2}, x)$. If the process falls in the water grave it is terminated.

The definition for a *convolution* of two functions is shown below (from [Ros03], page 58);

Definition 6.1.1 (Convolution) *Assuming X, Y are continuous and independent random variables having probability density functions (pdf) $f(x)$ and $g(y)$ respectively. Let F_{X+Y} be the cumulative distribution function (CDF) for X and Y. We have:*

$$\begin{aligned} F_{X+Y} &= P(X+Y \leq a) \\ &= \int\int_{X+Y \leq a} f(x)g(y)dxdy \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{a-y} f(x)g(y)dxdy \\ &= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{a-y} f(x)dx\right) g(y)dy \\ &= \int_{-\infty}^{\infty} F(a-y)g(y)dy. \end{aligned}$$

F_{X+Y} is the cumulative distribution function of X and Y and is called the convolution of the distribution of F_X and F_Y. Differentiate this and obtain the probability distribution function of $X + Y$.

$$\begin{aligned} f_{X+Y} &= \frac{d}{da}\int_{-\infty}^{\infty} F(a-y)g(y)dy \\ &= \int_{-\infty}^{\infty} \frac{d}{da}F(a-y)g(y)dy \\ &= \int_{-\infty}^{\infty} f(a-y)g(y)dy. \end{aligned}$$

Let us now get back to where we started and our Normal distribution at $t = \frac{T}{2}$. W.l.o.g we can normalize and set $\frac{T}{2} = 1$, and we get that $\sigma^2 = 1$. Let $\xi = x$ and let $\ltimes_{t=1}(\xi)$ be the distribution function for the random walk at $t = 1$ and we have $\ltimes(\xi) \sim N(0, 1)$:

$$\Rightarrow \ltimes_{t=1}(\xi) = \frac{1}{\sqrt{2\pi}} e^{\frac{-\xi^2}{2}}.$$

We now split the function $\ltimes_{t=1}(\xi)$ in two parts, one for the left tail and one for the right. The left tail is $\forall \xi \in (-\infty, -1]$ and the right tail $\forall \xi \in [1, \infty)$.

The distribution for $t \in (1, 2)$ is denoted by $\rho_{(t=2-1)}(y)$ and is a normal distribution with mean=ξ and variance one, i.e $\rho_{(t=2-1)}(y) = \frac{1}{\sqrt{2\pi}} e^{\frac{-y^2}{2}}$. The subindex $t = 2 - 1$ serves as a reminder that this part of the process start at $t = 1$ and arrive at $t = 2$ (and hence the variance is one).

By Definition 6.1.1 we get the convolution of the right hand tail as

$$\begin{aligned} f_{\text{right tail}}(t) &= \int_1^\infty f(a-y)g(y)dy \\ &= \int_1^\infty \ltimes_{t=1}(a-y)\rho_{(t=2-1)}(y)dy \\ &= \int_1^\infty \frac{1}{\sqrt{2\pi}} e^{\frac{-(a-y)^2}{2}} \frac{1}{\sqrt{2\pi}} e^{\frac{-y^2}{2}} dy \\ &= \frac{1}{2\pi} \int_1^\infty e^{\frac{-a^2+2ay-y^2}{2}} e^{\frac{-y^2}{2}} dy \\ &= \frac{1}{2\pi} e^{\frac{-a^2}{2}} \int_1^\infty e^{\frac{2ay-2y^2}{2}} dy \\ &= \frac{1}{2\pi} e^{\frac{-a^2}{2}} \int_1^\infty e^{ay-y^2} dy \end{aligned} \tag{6.1}$$

For the left tail we get similar, but mirrored, results;

$$\begin{aligned} f_{\text{left tail}}(t) &= \int_{-\infty}^{-1} \ltimes_{t=1}(a-y)\rho_{(t=2-1)}(y)dy \\ &= \int_{-\infty}^{-1} \frac{1}{\sqrt{2\pi}} e^{\frac{-(a-y)^2}{2}} \frac{1}{\sqrt{2\pi}} e^{\frac{-y^2}{2}} dy \\ &= \frac{1}{2\pi} e^{\frac{-a^2}{2}} \int_{-\infty}^{-1} e^{ay-y^2} dy \end{aligned} \tag{6.2}$$

The total solution for the problem at $T = 2$ is simply the sum of the left and the right tails:

$$\begin{aligned} f_{X+Y}(t) &= f_{\text{right tail}}(t) + f_{\text{left tail}}(t) \\ &= \frac{1}{2\pi} e^{\frac{-a^2}{2}} \left(\int_1^\infty e^{ay-y^2} dy + \int_{-\infty}^{-1} e^{ay-y^2} dy \right) \end{aligned} \tag{6.3}$$

Instead of solving the integrals in equation (6.3) by hand MATLAB was used to find the solutions. A plot can be found in figure 23.

Quasi Monte Carlo Approach

As mentioned above, the full random walk simulations for this setup will not be carried out, but instead the result of the analytical solution for the simulation is considered. That is, first we simulate a large number of random variables from a N(0,1) distribution to arrive at the water grave. Then $\forall$ the processes who survived the water grave and arrived outside the barrier, we do a new simulation, again from the $N(\mu, 1)$ distribution, where μ is the place of arrival (outside of the barrier). The arrival positions of the second lot of drunken men should compare rather well with the analytical results obtained above.

Comparing the two solutions

Just to show that the two solutions, the theoretical and the simulated, does indeed coincide, figure 23 shows both plots. I chose to plot both the left and right tail arrivals and the total (sum) arrivals. The smooth and nice paths are from the analytical results (generated using `plottaMatlabIntegral.m`) whilst the not so smooth paths come from the drunken men (generated using `quasiMCRightAndLeftOfTheWater1.m`).

The two solutions does not coincide exactly, but the general trend is similar and one can assume (via some central limit theorem) that the Quasi Monte Carlo method will converge to the Analytical method, as the number of simulations grows.

6.1.2 Proper slalom, with more than one obstacle.

In the last section there was only one obstacle, at a determined location, that the process might hit. If Filip *did* compete drunk, he would probably have more than one obstacle (gate) to hit. Thus we extend our model to contain more obstacles.

The set up is as follows: We have a random process, the same old drunken man as we always have, and we put him out on a big football field. On the field there are a number Γ of barriers, such that when the drunken man walks in to these he will fall over and never walk again, simulating the termination of the process. In the T direction these are evenly spread out with a distance $\frac{T}{\Gamma}$ between them. In the x direction they are randomly placed according to a normal distribution with mean around the center line (we assume that the drunken man has zero drift), and variance equal to the time the obstacle is reached[18].

We realize that it is possible to find the probabilities of this by using the mathematical framework demonstrated in the last section, but it would be rather tedious. Instead we apply a Monte Carlo simulations to find the resulting probabilities. The results are found using `noEdgesSlalomObstacles.m` and `slalomObstaclesTimeNoEdge.m` and are plotted in figure 24.

Intuitively one could imagine that $\rho(T)$ would be an increasing function since as T increases so does the variances, and hence spread, of the process and thus the probability of hitting a fixed-width object would decrease. As seen in the aforementioned figure, this is also the case.

6.1.3 Bigger obstacles implies less drunken men

Let us now fix the time T and vary the width δ of the terminating obstacles. The obstacles are placed in the same manner as above (evenly in the T direction and according to a normal distribution with mean at the center line and variance= T). This is plotted in figure 25 which is generated by `slalomObstaclesNoEdgeDifferentSizeRocks.m` and `noEdgesSlalomObstacles.m`.

[18] We say the drunken man reaches an obstacle when he either passes next to it or hits it.

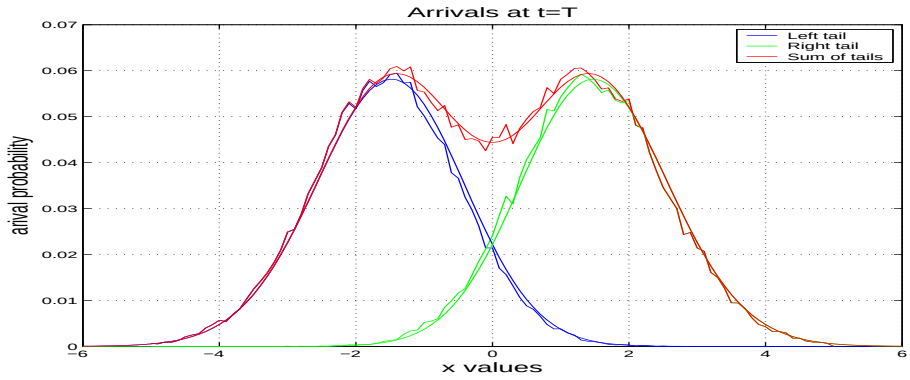

Figure 23: Plot showing both the Quasi Monte Carlo result versus the Theoretical result on the x-interval (-6.6). The Analytical solutions come from equations (6.2), (6.1) and (6.3). The Quasi Monte Carlo solutions used 250 000 simulations and took 7.03 sec on my PC.

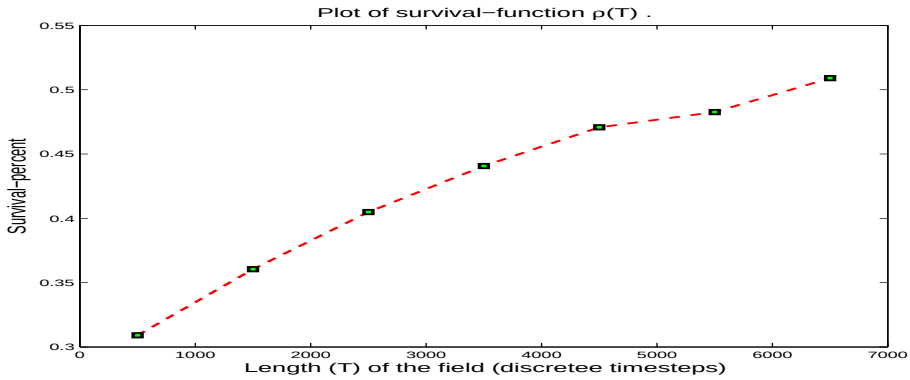

Figure 24: Plot of $\rho(T)$, the probability that the process survives a field with 10 obstacles, of width 10, for different values of T. (Computational time: 69 seconds on my PC, using 10000 simulations on each time step.)

6.1.4 Ramdom gates rather than obstacles

Let us invert the whole scenario and instead of having terminating objects that would terminate the process if hit, we now have *gates* that allows passing through, however, if the gate is missed, the process is terminated. The gates are rather small and thus this would mean that there is a substantially smaller chance for the process to survive than it would be when solid obstacles are in the way.

Let us imagine that the setup is the same as in the last scenario with the only difference that the obstacles have been inverted to gates. We plot this scenario in figure 26, which was generated by `noEdgesSlalomGates.m` and `slalomGatesNoEdges.m`, and the plot shows the process probability of passing through the ten gates.

6.1.5 Nice slalom is easy, hard slalom is hard.

Up until now the gates and obstacles have been randomly placed according to some normal distribution. In this example we change this and introduce deterministically placed gates. The gates are still evenly spread out in the T direction, but now every second gate is placed a distance Δ right of the center line, while the other gates are placed the same distance left of the center line. To illustrate the results we plot in figure 27, which was generated by `slalomGatesNicenessNoEdges.m` and `noEdgesSlalomGatesNiceness.m`, the survival function ρ for different values of Δ.

6.2 On a bridge

6.2.1 Terminating obstacles

Consider a field with terminating boundaries (i.e. the corresponding pde would have BC's of the type $X(\pm N) = 0$). In this field there are a number of small obstacles s.t. when the process hits these obstacles, it is terminated. We seek to find the probability of survival of the process, i.e. what is the probability that the process does not hit an obstacle, and does not fall off the bridge?

Let Γ denote the number of obstacles we place on the bridge, δ_i denote the width of obstacle $i, i = (1, ..., \Gamma)$, and α_i be the distance from obstacle i to the top (or left) barrier. For simplicity we start by giving all the obstacles the same width δ. We have an analogous, but unbound, problem in section 6.1.2 where we placed the obstacles, in the x-direction, according to a normal distribution. When we have boundaries and a normal distribution does no longer seems like the most natural choice[19], and hence we use a uniform distribution. For each

[19] According to the properties of a normal distribution this would mean that there is a possibility for the obstacles to be placed outside of the boundaries, and then they would be pointless. One could, of course, use either a truncated normal distribution, or not bother about those gates outside of the boundaries. None of those methods seemed too pleasing to me.

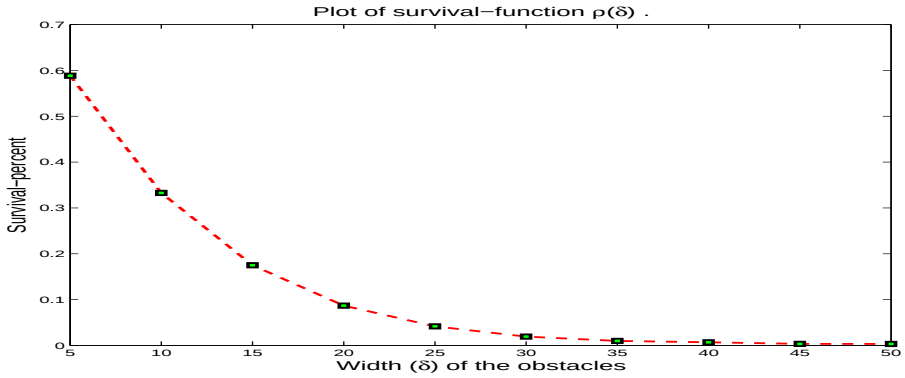

Figure 25: Plot of $\rho(T)$, the probability that the process survives a field with 10 obstacles of width 10 and $T = 1000$. Computational time: 16 seconds on my PC, using 10000 simulations on each time step.

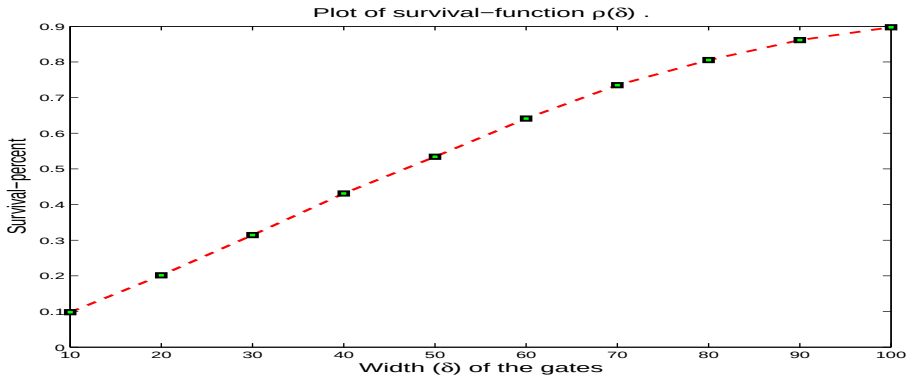

Figure 26: Plot of $\rho(T)$, the probability that the process survives a field with 10 gates of width 10 and $T = 1000$. (Computational time: 15 seconds on my PC, using 10000 simulations on each time step.)

simulated process we simulate a new and fresh set of α_i's uniformly between the boundaries.

The first thing we are interested in finding is a function $\rho(T)$, which describes how many (in percent) of the processes survive a walk along this slalom-like bridge. In figure 28, generated from `slalomObstaclesTime.m` and `terminateEdgesSlalomObstacles.m`, is shown how such a function might look like for certain values of N,δ and Γ.

6.2.2 The bigger the obstacles, the harder to survive.

How dependent is the above function $\rho(T)$ of the width of the obstacles? Why did we choose the obstacles to be 1/10'th of the field-width? In figure 29 is shown how the survival rates depend on the obstacle widths, and it is clear that a 10unit wide obstacle is about 25% of the processes survive and should thus be a good choice. The Matlab code for the plot can be found in `slalomObstacles.m` and `terminateEdgesSlalomObstacles.m`.

6.2.3 Stochastic Slalom with gates.

Let us now make a bounded analogy of the process described in section 6.1.4. We have Γ gates of width δ, they are uniformly placed in the x-direction and evenly spread in the T direction. If the process fail to pass through these gates it will be terminated. Filip Fisher, the snowboarder who broke the IOC rules and got really drunk before his race, will now be disqualified not if he hits a rock, but he fails to pass through the randomly placed gates on the ski-slope. If the gates are narrow this will be hard, and if the gates are wider it will be easier.

We seek a function $\rho(\delta)$, which is the probability of survival when the gates are δ wide. This is shown in Fig. 30 and the graph is obtained from `slalomGates.m` and `terminateEdgesSlalomGates.m`.

6.2.4 Hard vs. Easy slalom

In the last example we simulated α_i from a uniform distribution, so that the gates could be anywhere in $(-N, N)$. It would also be interesting, however, to see how the survival rate depends on the "niceness" on the slalom. A nice course is defined as one where all the gates are centered about the center-line, and a nasty course is one where the gates are really close to the boundaries. We let Δ denote the distance between the gate and the center-line of the field, and thus we want to plot a function $\rho(\Delta)$ of the survival percent of the processes. Every second gate is placed Δ above the center-line and every second is placed Δ below the center-line.

The function $\rho(\Delta)$ is plotted in figure 31, and the figure is generated via `slalomGatesNiceness.m` and `terminateEdgesSlalomGatesNiceness.m`.

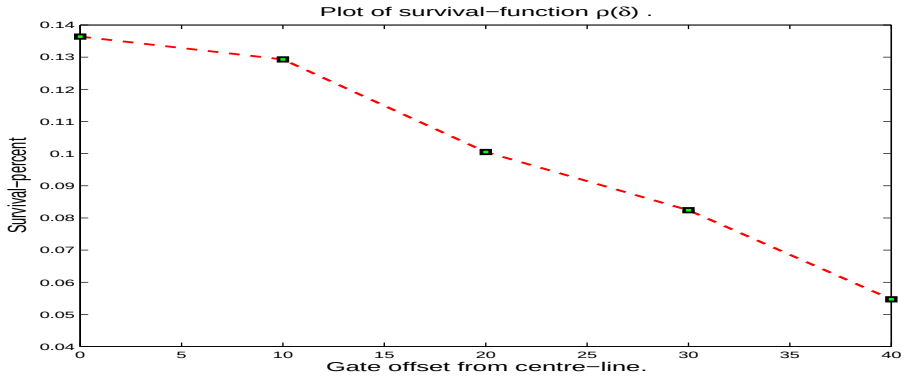

Figure 27: Plot of $\rho(\Delta)$, the probability that the process survives a field with 10 gates, of width 10, with alternating gate-offset $\pm\Delta$ from the center line. (Computational time: 19 seconds on my PC, using 10000 simulations for each Δ.)

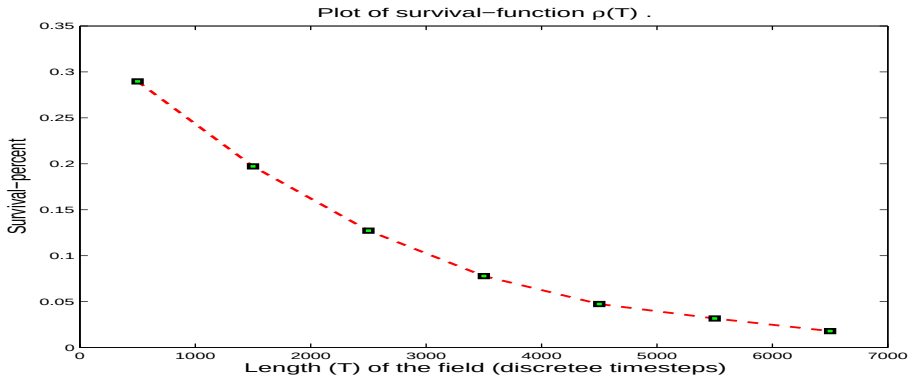

Figure 28: Plot of $\rho(T)$, the probability that the process survives a road (bridge) with $\Gamma = 10$ obstacles of width $\delta = 10$ for different values of T. The road was $2N = 100$ units wide and it took the my PC about 55 seconds to do 10000 simulations for each time step.

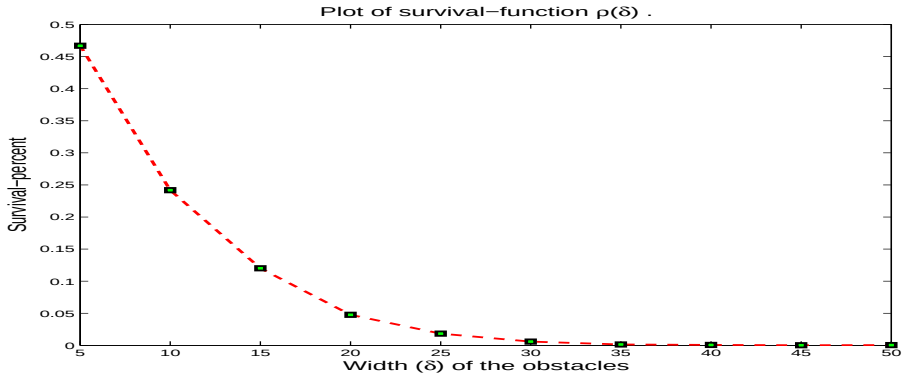

Figure 29: Plot of $\rho(\delta)$, the probability that the process survives a road (bridge) with $\Gamma = 10$ obstacles. We set the road to be $T = 1000$ discrete trime steps long, and $2N = 100$ steps wide (so $T = 10 * (2N)$). It took my computer about 23 seconds to make 10000 simulations for each obstacle width.

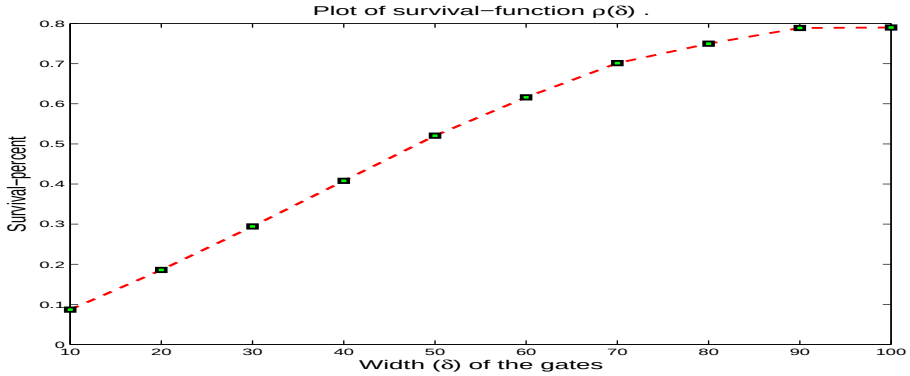

Figure 30: Plot of $\rho(\delta)$, the probability that the process survives a road (bridge) with 10 gates. We set the road to be 1000 discrete trime steps long, and 100 steps wide (so T=10*N). It took my computer 53 seconds to make 10000 simulations for each gate width. The gates are placed, in the T direction, with equal distance from each other and in the N direction they are randomly (uniformly between the barriers) laid out for each simulation. By missing any of the gates the process is terminated.

6.3 How easy is easy?

In figure 31 it is clear that the smaller the gate-offset the more processes will survive. If we look upon the problem the other way around we could ask ourselves "how much easier is it to pass through a road of width $2N$ and Γ gates of width δ (distributed around the center line), than it is to pass through a road of width δ?". To aid in the understanding of the question figure 32 was plotted.

The two different scenarios have been solved separately before - the first scenario in section 6.2.4, and the latter in section 3.3. We now combime the solutions to compare the difference between the δ wide gates case and the δ wide field case. In other words are we trying to figure out how much you are helped by the possibility of slaloming in between the walls.

I have plotted a solution curve in figure 33, which was generated by `slalomGatesVSwalls.m`, `terminateEdgesSlalomWOGates.m` and `terminateEdgesSlalomGates.m`.

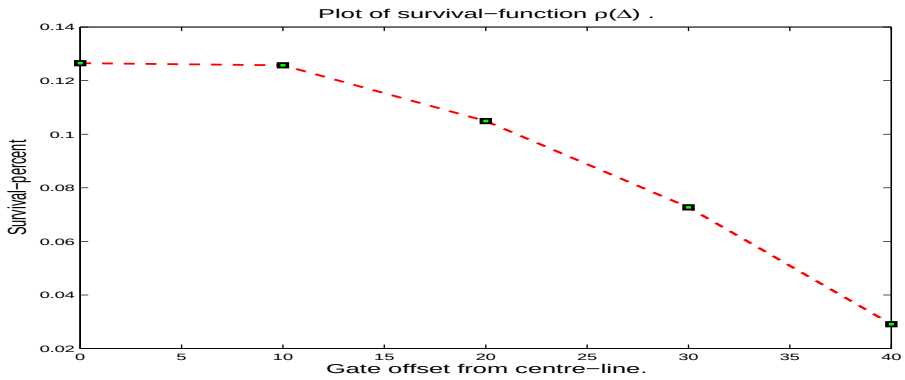

Figure 31: Plot of $\rho(\Delta)$, the probability that the process survives a road (bridge) with 10 gates of width 10. We set the road to be $T = 1000$ discrete trime steps long, and $2N = 100$ steps wide (so T=10*2N). It took my computer 23 seconds to make 10000 simulations for each Δ offset. The gates are placed, in the T direction, with equal distance from each other and in the N direction they are put (alternating) at $\pm\Delta$.

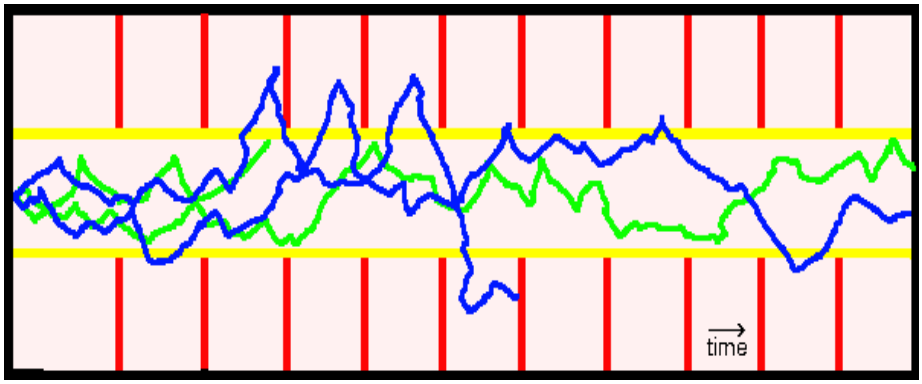

Figure 32: The Scenario. The green processes are terminated by the yellow boundary whilst the blue are terminated by the black and red boundaries.

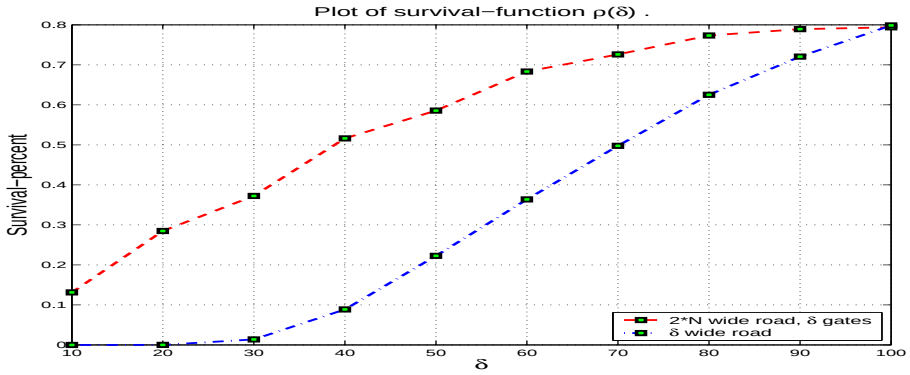

Figure 33: 1200 processes on a road that is $2N = 100$ steps wide with $\Gamma = 10$ gates. We see that the two solutions approach each other as the with of the gates tends to the width of the field ($\delta \to 2N$). Computational time: 105sek on my PC

7 Conclusion

By using Monte Carlo simulations we have demonstrated the extreme powers of a modern day PC to solve complicated BVP's for the heat diffusion PDE. Even though some of the examples could be solved analytically the bulk would be very hard.

Remark 1: There remain many problems of finding confidence intervals (and other matters that arise when Monte Carlo simulations are used) that might please a mathematical statistician. However, these problems are outside the scope of this thesis.

Remark 2: For future work I would do some sort of speed-optimization of the simulations. In the present work I have only used rather standard vectorization ideas to speed up the code, but I am sure there are many more things that could be implemented. It would also be interesting to compare the matlab code to a C++ equivalent to see which is the fastest. The object oriented approach made possible in a language like C++ would permit for a much more elegant code. In the present matlab code I have used "copy and paste" to re-use the code, something most programmers strongly disapprove of.

References

[Bjo04] Tomas Bjork.
Arbitrage Pricing in Continuous Time.
Oxford University Press, second edition, 2004.

[ea00] Jean-Michel Courtault (et al).
Louis bachelier on the entenary of théorie de la spéculation.
Mathematical Finance, 10(3):341–353, July 2000.
Found through http://www.stochastik.uni-freiburg.de/bfsweb/LBachelier/.

[Ein05] Albert Einstein.
Investigations on the Theory of The Brownian Motion.
PhD thesis, 1905.
English translation, re-published by Dover Publications Inc 1956.

[Hab04] Richard Haberman.
Applied Partial Differential Equations.
Pearson Prentice Hall, fourth edition, 2004.

[NW34] R. E. A. C. Paley N. Wiener.
Fourier Transforms in the Complex Domain.
American Mathematical Society Colloquium Publications, Volume XIX, 1934.

[Ros03] Sheldon M. Ross.
Introduction to Probability Models.
Academic Press, eight edition, 2003.

[Rud76] Walter Rudin.
Principles of Mathematical Analysis.
McGraw-Hill International Editions, third edition, 1976.

Appendices

A Another approach using Transition Probabilities

This section will be a very brief introduction to how one can model a bounded random walk using Markov chains. Markov chains are easy to simulate using a computer, but unfortunately a lot of the processes mentioned earlier (for instance those introducing a drift) do not retain the Markovian property.

As we know from the above the bounded random walk can be modelled using the repeated toss of a coin: If you get a head you step right, and if you get tails you you step left. At each toss there is an equal, binomial 0.5, probability of getting heads or tails. after a large number of tosses the amount one has earned follows a normal distribution with zero mean and variance equal to the square root of the number of tosses (via The Central Limit Theorem).

A.1 Markov Chains

By standard definition, a Markov chain is a random process where the next state depends only on the current state and not on any other, older, information. The bounded random walk is a good example of a discrete time Markov chain - independent of how far away from the middle of the road Erik is, he still have the same chance of taking a step to the right as taking a step to the left.

When (if) Erik hits the edge of the road he might bounce back or fall off the bridge. Clearly if he falls off (cf Scenario II above) he will go into an *absorbing state*, and if he bounces (cf Scenario III) this bounce is introduced so it depends only on the current state. I.e the process will go on as before, but if Erik reaches the edge of the bridge it will move him one step away from the edge.

A well-known way to deal with Markov Chains is to create a so-called *Transition Probability Matrix*. A Transition Probability Matrix is the same,up to a scalar vector, as the *Adjacency Matrix* used in Graph Theory.

To exemplify let us model Scenario III above. Define A to be a transition Probability Matrix. The Elements of A, a_{ij}, then describe the probability that when the process is in state i the next state the process enters will be j. So in the case with the random walk one can let

$$a_{ij} = \begin{cases} 1/2 & \text{if } |i-j| = 1 \\ 0 & \text{otherwise} \end{cases} \tag{A.1}$$

$$a_{12} = a_{(n-1)n} = 1 \tag{A.2}$$

The last line, equation (A.2), forces the process to bounce on the wall and not to escape outside the matrix . This is called an irreducible Markov Chain where all states communicates. All states are *positive recurrent*, i.e. when the process have been in i the expected time until it returns to i is finite, and have period of 2, since it is a binomial tree there are always places we cannot get to if we are at an even number of steps (the same applies for odd step-numbers).

So we have a matrix looking like

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & \dots & a_{0k} & \dots & a_{0n} \\ a_{10} & a_{11} & a_{12} & \dots & a_{1k} & \dots & a_{1n} \\ a_{20} & a_{21} & a_{22} & \dots & a_{2k} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & & & \\ a_{k0} & a_{k1} & a_{k2} & \dots & a_{kk} & \dots & a_{kn} \\ \vdots & \vdots & \vdots & & & \ddots & \\ \vdots & & & & & & \\ a_{n0} & a_{n1} & a_{n2} & \dots & a_{nk} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} 0 & .5 & 0 & \dots & 0 & \dots & & & 0 \\ 1 & 0 & .5 & \dots & 0 & \dots & & & 0 \\ 0 & .5 & 0 & \dots & 0 & \dots & & & 0 \\ \vdots & \vdots & \vdots & \ddots & & & & & \\ 0 & \dots & & .5 & 0 & .5 & \dots & & 0 \\ \vdots & \vdots & \vdots & & & \ddots & & & \\ 0 & \dots & & & 0 & \dots & 0 & .5 & 0 \\ 0 & \dots & & & 0 & \dots & .5 & 0 & 1 \\ 0 & \dots & & & 0 & \dots & 0 & .5 & 0 \end{pmatrix}$$

This is called a one-step probability matrix, and totally defines how Erik can walk in one step. Assuming that n is an odd number a_{kk} is defined to be the central element of the array and this is where Erik starts. To the left of a_{kk} is the left of the road, and to the right is the right part of the road. For example, $a_{(k-1)k}$ is the probability that Erik will, when at a distance 1 from the middle of the road take one step and end up in the middle of the road.

The diagonal symmetry is obvious, since Erik *have* to take a step either right or left there is probability 0 that he will stay in the same place, and the probability that he will walk one step either left or right is 0.5. The upper-right and lower-left corners are zero because there is no chance for Erik to walk, in only one step, from the far left of the road to the far right.

By taking the n-th power of A we get the n-step probability matrix. So a_{ij}^n signifies the probability that a process in state i will, after n steps, be in state j. For example, A^{600} is the matrix holding the probabilities that starting in state i the process will be in state j after 600 steps. As the powers increase the matrix is spreading to the upper-left and lower-right corners.

By summing all these powers of matrices we can get the cumulative n-step probability matrices. Then $\sum_{k=1}^{n} a_{ij}^k$ determines the probability that the process starting in state i has ever been in state j after n steps. Since these values quite often exceeds one, one can use them as the expected number of times the process hits a certain level.

For example, a random walk on a road that is 201 steps wide and 600 steps wide we get the values in the table below. To verify that this technique generates the same results as the Monte Carlo simulations of above I made a comparison of the two methods aswell. The simulated values are to be found next to the purely theoretical values.

Distance from Centre	Theoretical Values	Simulated Values
0	$a_{00} = 18.5685$	18.6187
10	$a_{010} = 11.1726$	11.1882
20	$a_{020} = 5.7358$	5.7358
30	$a_{030} = 2.6215$	2.5564
40	$a_{040} = 1.0575$	1.0031
50	$a_{050} = 0.3738$	0.3673
60	$a_{060} = 0.1151$	0.1209
70	$a_{070} = 0.0307$	0.0278
80	$a_{080} = 0.0071$	0.0063

As you can see the simulated values match the ones from the Cumulative 600-step transition probability matrix quite well. As i only did 40 000 simulations of the process it is not strange that the simulated results does not exactly match the Markov Chain values.

A.2 Is there an end?

As the number of steps increases it seems likely that the n-step probability matrix would converge to a limiting probability matrix - that it would spread out and converge towards a uniform distribution. The crudest way to prove a result like this would be to let n grow until $A^n - A^{n-1} = 0$, where 0 is the zero matrix. I have not done this but it seems, by looking at the Fig. 34 that a result like this might be true.

In figure (Fig. 34) is shown a typical convergence sequence for a random walk with $N = 201$. The first (blue) is when the road is only 2 steps long, and then the road is extended 300 steps 40 times. The source code for this can be found elsewhere in the appendix (`fiveTransProb1.m`).

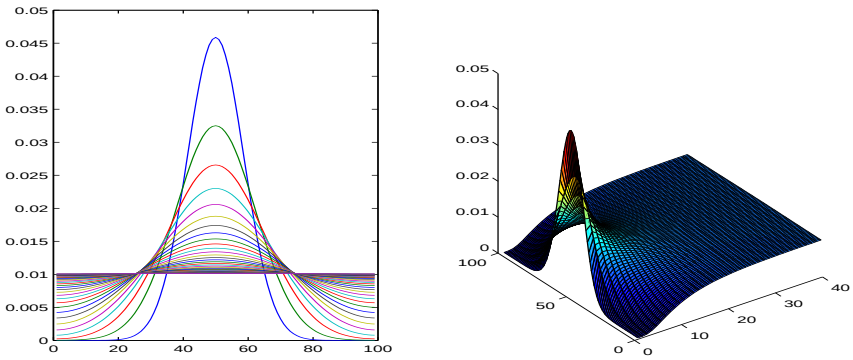

Figure 34: Convergence of Transition Probabilities. $N = 100$ and T starts at 2 and goes up, in increments of 300 steps, 40 times.

B Introducing the Black-Scholes framework

In the latter part of my rather exciting life I have become interested in Financial Markets in general and in option pricing in particular. Thus I chose to include an appendix where I quickly go through the basics of the Black-Scholes framework and then show how one can transform the solution of the Black-Scholes equation (for European Calls) into the diffusion equation - thus linking the topic of the thesis to my personal interest. The bulk of the text is a rewrite of my notes from a lecture by Professor Peter Vamos at University of Exeter in 2004, and the calculations arose as part of a homework assignment of that same lecture.

B.1 Black, Scholes and Merton

The most famous formula for the valuation of stock options is the Black-Scholes (B-S) Model. The model was first published in 1973 by Fisher Black and Merton Scholes, Robert Merton devised another way of deriving the formula and generalized it in many directions. Black passed away in 1995 whilst Merton and Scholes got the Nobel Prize in Economics in 1997 for their work on the Black-Scholes formula.

B.1.1 Bank interest

Let us define a risk free asset as an asset that changes in the same way as the money would do if they were in the bank, i.e according to the interest rate. Let us therefore define the interest rate[20] to be proportional, by the amount of money, to the rate of growth of the money in the bank.

Definition B.1.1 (Interest Rate) *We let* M = amount of money in the bank *and* r = rate of interest*, we then define* r *as*

$$\frac{dM}{dt} = r \cdot M. \tag{B.1}$$

Equation (B.1) is easy to solve and has the General Solution (G.S)

$$M(t) = C \cdot e^{rt},$$

where C is some constant. An immediate question that arise is how much money M should we put in the bank today, if we want an amount E in some future time T≥t ? We find this via the Particular Solution (P.S) to equation (B.1);

$$\begin{aligned} & E = C \cdot e^{rT} \\ \Rightarrow \quad & C = E \cdot e^{-rT} \\ \Rightarrow \quad & M(t) = E \cdot e^{-r(T-t)}. \end{aligned} \tag{B.2}$$

Plugging in $t = T$ in the last equation we see that $M(T) = E$ as we wished. This leads on to the definition of the *No free lunch principle*.

[20] To be precise, the constant continuous compounding/instantaneous interest rate.

Definition B.1.2 (The No Free Lunch Principle (NFL)) *There is no opportunity, for any length of time for a risk-free profit above the bank (instantaneous) interest rate.*

I will not prove the NFL, but will give a justification as to why it should work. Suppose there would be such an risk-free arbitrage opportunity. We could then borrow money from the bank at rate r and then invest in this other asset and earn a rate $n > r$. Since it is *risk free* to invest in this new investment, everybody would do so. The result of this would be that the bank interest goes up and the gap closes. In the modern society world this will happened, due to the principle of the efficient market, practically instantaneously.

B.2 European Put- and Call options

An option is a contract giving the buyer the right to buy or sell a specified amount of a specified product to a specified price at a specified future time. There are many different kinds of options and the two major are called European and American. The European option can only be exercised at the specified date whilst the American Option can be exercised up to and including the specified date. A put option is one where you buy the right to sell an underlying (i.e. stock) in the future, and a call option is one where you buy the right to buy an underlying in the future.

To exemplify the European Call option let us imagine there is a company called Björks Mathematics Consultants AB (BMC), with a current stock price of \$10 per stock unit. Let us assume that Erik buys a call option with the strike price \$12 and the expiry date three months in ahead. So in three months time, if the stock price is above \$12 then Erik will exercise his option and earn the difference between the stock price and \$12. If, however, the stock price is below \$12 Erik will not have to exercise the option which is thus worth \$0.

B.2.1 Boundary Conditions

To be able to price the derivatives we need to translate how they work into mathematics. So let us derive the Boundary Conditions (BC's) for the European Call (and Put) Options. We denote the value of a call option $C(S,t)$, where S is the current value of the underlying (stock), and t is the current time. The expiry date (length of contract) is denoted T and the exercise price in the contract is denoted E. Since E is a price, and prices here are always positive, we get $E > 0$.

The value of the option at any time t is easily seen from the example with Erik and the BMC above.

$$\begin{aligned} C(S,t) &= \begin{cases} S-E & \text{if } S > E \\ 0 & \text{if } S \leq E \end{cases} \\ &= max(S-E,0) \end{aligned} \tag{B.3}$$

If the underlying cost nothing the option has no value, so it is worth $max(0 -$

$E, 0) = 0$ since $E > 0$. We get the BC as

$$C(0, t) = 0 \forall (0 \leq t \leq T). \tag{B.4}$$

If the price of the stock goes off towards infinity, we want the value of the option to follow. In other words we can say that we want the price of the option to be asymptotic to the price of the underlying, as the price of the underlying tends towards infinity. In mathematics we see that $\lim_{S\to\infty} max(S - E, 0) \to \infty$, so

$$\frac{\lim_{S\to\infty} C(S, t)}{S} = 1. \tag{B.5}$$

Similarly for the European Put option we get the price of the option as

$$\begin{aligned} P(S, t) &= \begin{cases} E - S & \text{if } S > E \\ 0 & \text{if } S \leq E \end{cases} \\ &= max(E - S, 0) \end{aligned}$$

If $P(0, t)$ then this gives holder risk-free guaranteed return of E at the future time T. We can use the NFL to conclude that this has to equal the amount of money in the bank which return E at time t=T. Hence, via equation (B.2), we get

$$P(0, t) = E{\cdot}e^{-r(T-t)} = E{\cdot}e^{r(t-T)}.$$

By arguments similar to those of the European Call option we also have that

$$P(S, t) \to 0 \text{ as } S \to \infty.$$

B.2.2 Put-Call Parity Rule

By using methods of hedging one can create portfolios with different levels of risk. The most basic risk-free portfolio is used to derive the *Put-Call Parity Rule*. Consider a portfolio of 1 share of underlying, 1 Call-option on 1 share of the underlying (with specified T and E), and one Put option on 1 share of the underlying (with the same T and E as the Call-option). A Call option gives us the right to buy and underlying and so it is a "+", a Put option on the other hand gives us the right to sell an underlying and thus there is a "-". Denoting the value of the portfolio $V(S, t)$ we get the value of the portfolio at time $t = T$, the time of expiry of the options, as

$$V(S, T) = \left\{ \begin{array}{ll} S + \underbrace{0}_{put} - \underbrace{(S - E)}_{call} = E & \text{if } S \geq E \\ S + \underbrace{(E - S)}_{put} - \underbrace{0}_{call} = E & \text{if } S < E \end{array} \right\} = E. \tag{B.6}$$

We see that whatever happens in the stock market the portfolio is still worth E at time $t = T$, so by the NFL we get that

$$V(S, T) = E{\cdot}e^{r(t-T)} \tag{B.7}$$

and so we get the rule:

Proposition B.2.1 (The Put-Call Parity Rule) *By taking one unit of stock, one call option and one stock option, we get the Put-Call parity rule:*

$$S + P(S,t) - C(S,t) = E \cdot e^{r(t-T)}. \tag{B.8}$$

B.2.3 Black-Scholes Equation

In order to price derivatives and options, we need to make some further assumptions about how the market works, so let us start with the following assumptions:

1. Asset prices follow a *log-normal random walk* (history tell us this)
2. Parameters (S,T,E) are known functions of time
3. There are no transaction costs (frictionless trade)
4. The underlying asset pays no dividend during the life of the portfolio/option
5. The NFL principle applies (i.e. no arbitrage opportunity)

As a results of these assumptions we get the Black-Scholes equation (for a derivation, see for example [Bjo04]).

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0. \tag{B.9}$$

This equation is always the same, for all options. What differs from option to option is the BC's, and through the BC's a solution is found. For simple options like the European Put and Calls the BC's are easy to come by, but for more complicated "exotic" options it might be much harder to find the BC's - and thus making solving the Black-Scholes equation very hard indeed. In that case you have to rely on computer intense methods of finding the option price - and Monte Carlo is one such method. For the European Put/Call options, however, the BC's are given above and the solution to the Call option is given by

$$C(S,t) = SN(l_1) - Ee^{-r(T-t)}N(l_2),$$

where

$$N(l) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{l} e^{-(1/2)x^2} dx$$

is the density function for the Normal distribution and

$$l_{1,2} = \frac{ln(S/E) + (r \pm \sigma^2/2)(T-t)}{\sigma\sqrt{T-t}}.$$

B.2.4 Black-Scholes versus Heat-diffusion PDE

We have the Black-Scholes (B-S) formula and it's solutions for the European call option, but we have yet not figured out how and why we can apply the technique of Stochastic Simulation to this set up. In this section I will show how we can, via a few clever choices, transform the Black-Scholes formula to the Heat-Diffusion equation. And the heat-diffusion equation is what we've been dealing with in the previous sections of this text.

The aim of the transformation is to transform the B-S equation into something simpler that we can solve, and then do the reverse transformation on the solution to obtain the solution to the B-S equation. We divide the first transformation into two steps. The aim of the first step is to get rid of the variable S as a coefficient in the equation, leaving us with a constant coefficient PDE. The aim of the second step is to get rid of the first order partial and the function itself.

Step 1 - get rid of variable S as coefficient. We start by introducing new variables $(x, \tau) \mapsto (S, t)$. We know that $0 \leq S \leq \infty$ and $0 \leq t \leq T$ and we now let

$$S = Ee^x, \text{ and} \tag{B.10}$$

$$t = T - \frac{\tau}{\frac{1}{2}\sigma^2}, \tag{B.11}$$

where $-\infty < x < \infty$ and $0 \leq \tau \leq \frac{1}{2}\sigma^2 T$ (time reversal since $\tau = 0$ corresponds to $t = T$). We also introduce a new function f of these variables such that

$$V(S, t) = E \cdot f(x, \tau),$$

and we now want to find the B-S PDE for this new function f. It will turn out later that we will need to know some derivatives, so let us start by calculating these:

$$\begin{aligned} (B.10) \Rightarrow \frac{dS}{dx} = Ee^x \quad &\Rightarrow \quad \frac{dx}{dS} = \frac{1}{Ee^x} = \frac{1}{S} \\ (B.11) \Rightarrow \frac{dt}{d\tau} = -\frac{1}{\frac{1}{2}\sigma^2} \quad &\Rightarrow \quad \frac{d\tau}{dt} = \frac{-\sigma^2}{2} \end{aligned}$$

Remembering the order of dependence, $V \to S \to x$ and $V \to r \to \tau$, we use the chain rule of differentiation along with the derivatives above to compute the various partial derivatives required for the B-S equation:

$$\frac{\partial V}{\partial t} = E \cdot \frac{\partial}{\partial t} \cdot f(x, \tau) = E \cdot \frac{\partial t}{\partial \tau} \cdot \frac{\partial \tau}{\partial t} = E \cdot \frac{\partial f}{\partial \tau} \left(-\frac{1}{2} \cdot \sigma^2 \right), \text{ and}$$

$$\frac{\partial V}{\partial S} = E \cdot \frac{\partial f}{\partial x} \cdot \frac{\partial x}{\partial S} = E \cdot \frac{\partial f}{\partial x} \cdot \frac{1}{S} = \frac{E}{S} \cdot \frac{\partial f}{\partial x}, \text{ and}$$

$$\begin{aligned}\frac{\partial^2 V}{\partial S^2} = \frac{-E}{S^2}\cdot\frac{\partial f}{\partial x} + \frac{E}{S}\cdot\frac{\partial}{\partial S}\cdot\frac{\partial f}{\partial x} &= \frac{-E}{S^2}\cdot\frac{\partial f}{\partial x} + \frac{E}{S}\cdot\frac{\partial^2 f}{\partial S^2}\cdot\frac{\partial x}{\partial S} \\ &= \frac{-E}{S^2}\cdot\frac{\partial f}{\partial x} + \frac{E}{S^2}\cdot\frac{\partial^2 f}{\partial x^2}.\end{aligned}$$

Plug all these into the B-S equation (B.9) and we get

$$E\cdot\frac{\partial f}{\partial t}\left(-\frac{1}{2}\sigma^2\right) + \frac{1}{2}\sigma^2 S^2\left(\frac{-E}{S^2}\frac{\partial f}{\partial x} + \frac{E}{S^2}\frac{\partial^2 t}{\partial x^2}\right) + rS\frac{E}{S}\frac{\partial f}{\partial x} - rEf = 0.$$

Cancel with E and divide by $-\frac{1}{2}\sigma^2$ to get

$$\frac{\partial f}{\partial \tau} = \frac{\partial^2 f}{\partial x^2} + (k-1)\frac{\partial f}{\partial x} - kf, \tag{B.12}$$

where $k = \frac{r}{\frac{1}{2}\sigma^2}$. This is a PDE with constant coefficients!

Boundary Transforms To be able to solve the resulting PDE we also need to transform the Boundary Conditions. For the European Call Option these are given in (B.3)-(B.5), and the transforms are given in (B.10) and (B.11).[21]

BC 1: $V(S,t) = max(S-E,0)$. We first note that at time $t = T$ we have (via (B.11))

$$\begin{aligned} t = T &= T - \frac{\tau}{\frac{1}{2}\sigma^2} \\ \Rightarrow \tau &= 0.\end{aligned}$$

So, $V(S,T) = Ef(x,0)$. We have $V(S,t) = max(S-E,0)$ so we investigate what happens at $S-E$ and 0.

At $V(S,T) = S-E$:

$$\begin{aligned} V(S,T) = S - E &= Ef(x,o) \\ Ee^x - E &= Ef(x,0), \text{ via (B.10)} \\ e^x - 1 &= f(x,0)\end{aligned}$$

At $V(S,T) = 0$:

$$\begin{aligned} V(S,T) = 0 &= Ef(x,o) \\ 0 &= f(x,0).\end{aligned}$$

So $V(S,t) = max(S-E,0)$ becomes $f(x,0) = max(e^x - 1, 0)$ under the given transformation.

BC 2: $V(0,t) = 0$. (B.10) together with the fact that $E \neq 0$ (actually, $E > 0$) $\therefore S = 0 \iff x \to -\infty$.

Now, $t \in [0,T]$ and $t = T - \frac{\tau}{\frac{1}{2}\sigma^2}$ so we see that $\tau = \frac{1}{2}\sigma^2[T-t]$ and $\therefore \tau \in [o, \frac{1}{2}\sigma^2 T]$. So we get

$$V(0,t) = \lim_{x\to-\infty} Ef(x,\tau) = 0$$

[21] We have a portfolio of only one Call Option, so $V(S,t) = C(S,t) \forall S,t$.

$$\Rightarrow \lim_{x\to-\infty} f(x,\tau) = 0 \forall \tau \in \left[0, \frac{1}{2}\sigma^2 T\right]$$

BC 3: $\lim_{S\to\infty} \frac{V(S,t)}{S} = 1$. According to the transform we have $V(S,t) = Ef(x,\tau)$ and $S = Ee^x$, so we get

$$\lim_{S\to\infty} \frac{Ef(x,\tau)}{Ee^x} = \lim_{S\to\infty} e^{-x} f(x,\tau) = 1.$$

Step 2 - Get rid of $\frac{\partial f}{\partial x}$ and f. To get rid of the partial and the function it self we introduce a suitable exponential multiplier:

$$f(x,\tau) = e^{\alpha x+\beta\tau} g(x,\tau), \tag{B.13}$$

where we choose α, β to achieve our aim. Taking the partial derivatives required in the Black-Scholes formula we get

$$\begin{aligned}
\frac{\partial f}{\partial \tau} &= \beta e^{\alpha x+\beta\tau} g(x,\tau) \frac{\partial g}{\partial \tau}, \\
\frac{\partial f}{\partial x} &= \alpha e^{\alpha x+\beta\tau} g(x,\tau) + e^{\alpha x+\beta\tau} \frac{\partial g}{\partial x}, \text{ and} \\
\frac{\partial^2 f}{\partial x^2} &= \alpha^2 e^{\alpha x+\beta\tau} g(x,\tau) + \alpha e^{\alpha x+\beta\tau} \frac{\partial g}{\partial x} + \alpha e^{\alpha x+\beta\tau} \frac{\partial g}{\partial x} + e^{\alpha x+\beta\tau} \frac{\partial^2 g}{\partial x^2} \\
&= \alpha^2 e^{\alpha x+\beta\tau} g(x,\tau) + 2\alpha e^{\alpha x+\beta\tau} \frac{\partial g}{\partial x} + e^{\alpha x+\beta\tau} \frac{\partial^2 g}{\partial x^2}.
\end{aligned}$$

Plugging these equations into (B.12) and cancel with $e^{\alpha x+\beta\tau}$ we get

$$\frac{\partial g}{\partial \tau} = \frac{\partial^2 g}{\partial x^2} + (2\alpha + k - 1)\frac{\partial g}{\partial x} + \left[(\alpha^2 + \alpha(k-1) - (k+\beta)\right] g.$$

The aim is to get rid of the partials and the non-differentiated function, so we want to choose α and β such that

$$\begin{aligned}
2\alpha + k - 1 &= 0, \text{ and} \\
\alpha^2 + \alpha(k-1) - (k+\beta) &= 0.
\end{aligned}$$

From the first of these equations we see that

$$\alpha = \frac{1-k}{2},$$

and plugging this into the second equation and re-arranging we get

$$\beta = \alpha^2 + \alpha(k-1) - k = \frac{(1-k)^2}{4} + \frac{(1-k)(k-1)}{2} - k = -\frac{(k+1)^2}{4}.$$

With the above values of α and β we have rewritten the Black-Scholes equation as

$$\frac{\partial g}{\partial \tau} = \frac{\partial^2 g}{\partial x^2} \tag{B.14}$$

which is of course the Heat-Diffusion equation!

Boundary Transforms. Using the European Call option boundaries as before, we now transform the boundary conditions.

BC 1:

$$\begin{aligned} f(x,0) = e^{\alpha x} g(x,0) &= max\left(e^x - 1, 0\right) \\ \Rightarrow g(x,0) &= max\left(e^{-\alpha x}\left(e^x - 1\right), 0\right) \\ &= max\left(e^{\frac{k+1}{2}x} - e^{\frac{k-1}{2}x}, 0\right), \text{ since } \alpha = -\frac{k-1}{2}. \end{aligned}$$

BC 2: α and β are constants, so

$$\begin{aligned} \lim_{x \to -\infty} f(x,\tau) = 0 \qquad & \text{becomes} \\ \lim_{x \to -\infty} e^{\alpha x + \beta \tau} g(x,\tau) = 0 \quad & \forall \tau \in [o, \tfrac{1}{2}\sigma^2 T] \end{aligned}$$

BC 3:

$$\begin{aligned} e^{-x} f(x,\tau) = e^{-x} e^{\alpha x + \beta \tau} g(x,\tau) & \\ = e^{(\alpha - 1)x + \beta \tau} g(x,\tau) & \end{aligned}$$

so the boundary condition becomes

$$\lim_{x \to \infty} \left(e^{(\alpha-1)x + \beta\tau} g(x,\tau)\right) = 1.$$

Step 3 - solve the heat-diffusion PDE with transformed boundary conditions. We have transformed the Black Scholes PDE into the diffusion equation. The solution to the diffusion equation (with given BC's) is given by:

$$g(x,\tau) = \frac{1}{2\sqrt{\pi}\tau} \int_{-\infty}^{\infty} g(y,0) e^{-\frac{(x-y)^2}{4\tau}} dy$$

Step 4 - Reverse transformation for solution Remembering the way we did the transformation, we now only need to reverse this for the solution for the Call Option BC. We have then solve the complex looking Black-Scholes SDE using the well studied diffusion PDE!

C Source Code for chapter 2

C.1 Plotting solution to diffusion equation without boundaries

plotNoBoundriesFourierSer.m

```
clear; clf;
xStepLength=0.05;
xM=6;
x=[-xM:xStepLength:xM];
n=10;
t=10;
tSteps=[.1:1:5.1];
fourierX=zeros(length(x),length(tSteps));

for tStep=1:length(tSteps)
    for xStep=1:length(x)
        fourierX(xStep,tStep)=1/sqrt(2*pi*tSteps(tStep))*exp(-(x(xStep)^2)/(2*tSteps(tSt
    end
end

figure(1)
subplot(1,2,1)
grid on
plot(x,fourierX)
xlabel('x','FontSize',15);
ylabel('probability','FontSize',15);
axis([-xM xM 0 max(max(fourierX))])
subplot(1,2,2)
%surface(fourierX)


surface('XData',tSteps,'YData',x,'ZData',fourierX,'CData',fourierX)
xlabel('time','FontSize',15);
ylabel('x','FontSize',15);
zlabel('probability','FontSize',15);
axis([0 tSteps(end) -xM xM 0 max(max(fourierX))])
view(-55,25)
```

C.2 Plotting solution to diffusion equation with terminating boundaries

plotTerminatingFourierSer.m

```
clear; clf;
xStepLength=0.05;
```

```
x=[0:xStepLength:pi];
n=100;
t=10;
tSteps=[.1:.5:3.1];
fourierX=zeros(length(x),length(tSteps));

for tStep=1:length(tSteps)
    for xStep=1:length(x)
        fourierX(xStep,tStep)=sumTerminatingFourierSer(x(xStep),tSteps(tStep),n);
    end
end

figure(1)
subplot(1,2,1)
plot(x,fourierX)
xlabel('x','FontSize',15);
ylabel('probability','FontSize',15);
axis([0 pi 0 1.5])
subplot(1,2,2)
%surface(fourierX)
surface('XData',tSteps,'YData',x,'ZData',fourierX,'CData',fourierX)
xlabel('time','FontSize',15);
ylabel('x','FontSize',15);
zlabel('probability','FontSize',15);
axis([0 tSteps(end) 0 pi 0 1.5])
view(-55,25)
%text(10,10,['Plot of Analytical solution to the diffusion eqn with given BC and IC.'],'
```

sumTerminatingFourierSer.m

```
function serSoln = sumTerminatingFourierSer(x,t,n)
%this is a function for summing up all rest in the cosine term
%of the fourier series solution we got.
%t - the time is fixed. We want to calculte for a specified time.
series = 0;
for count=0:n
    thisTime = ((-1)^(-count))*sin((2*count+1)*x)*exp(-((count+1)^2)*t);
    series = series + thisTime;
end
serSoln = (2/pi)*series;
    %remember the 1/pi.
```

plotIntegratedFourierSer.m

```
clear;% clf;
n=1000;
t=10;
```

```
tSteps=[.1:.5:4];
fourierX=zeros(length(tSteps),1);
for tStep=1:length(tSteps)
        fourierX(tStep)=sumIntegratedFourierSer(tSteps(tStep),n);
end

figure(1)
%subplot(1,2,1)
hold on
plot(tSteps,fourierX, 'Color','Red')
xlabel('t','FontSize',15);
ylabel('probability','FontSize',15);
title('Survival chance decays in time','FontSize',15);
axis([0 t 0 1])
% subplot(1,2,2)
% surface(fourierX)
% %surface('XData',tSteps,'YData',x,'ZData',fourierX,'CData',fourierX)
% xlabel('time','FontSize',15);
% ylabel('x','FontSize',15);
% zlabel('probability','FontSize',15);
% %axis([0 tSteps(end) 0 pi 0 1.5])
% view(-55,25)
% %text(10,10,['Plot of Analytical solution to the diffusion eqn with given BC and IC.']
```

sumIntegratedFourierSer.m

```
function serSoln = sumIntegratedFourierSer(t,n)
%this is a function for summing up all rest in the cosine term
%of the fourier series solution we got.
%t - the time is fixed. We want to calculte for a specified time.
series = 0;
n=100;
for count=0:n
    thisTime = ((-1)^(-count))*(2/(2*count+1))*exp(-((count+1)^2)*t);
    series = series + thisTime;
end
serSoln = (2/pi)*series;
    %remember the 1/pi.
```

plotNormalizedFourierSer.m

```
clear; clf;
xStepLength=0.05;
x=[0:xStepLength:pi+xStepLength];
n=100;
t=10;
tSteps=[.1:.1:3.1];
```

```
fourierX=zeros(length(x),length(tSteps));
fourierXi=zeros(length(tSteps),1);
for tStep=1:length(tSteps)
    for xStep=1:length(x)
        fourierX(xStep,tStep)=sumTerminatingFourierSer(x(xStep),tSteps(tStep),n);
    end
    fourierXi(tStep)=sumIntegratedFourierSer(tSteps(tStep),n);
end

figure(1)
normX=zeros(length(x),length(tSteps));
for jh=1:length(tSteps)
    normX(:,jh)=fourierX(:,jh)./fourierXi(jh,1);
end
plot(x,normX)
hold on
plot(x,.5*sin(x),'Color','red','LineWidth',2);
xlabel('x','FontSize',15);
ylabel('normalized probability','FontSize',15);
axis([0 pi 0 .6])
```

C.3 Plotting solution to diffusion equation with bouncing boundaries

plotBouncinfFourierSer.m

```
clear; clf;
xStepLength=0.05;
x=[-pi:xStepLength:pi];
n=10;
t=10;
tSteps=[.1:.5:3.1];
fourierX=zeros(length(x),length(tSteps));

for tStep=1:length(tSteps)
    for xStep=1:length(x)
        fourierX(xStep,tStep)=sumBouncingFourierSer(x(xStep),tSteps(tStep),n);
    end
end

figure(1)
subplot(1,2,1)
grid on
plot(x,fourierX)
xlabel('x','FontSize',15);
ylabel('probability','FontSize',15);
```

```
axis([-pi pi 0 1])
subplot(1,2,2)
%surface(fourierX)


surface('XData',tSteps,'YData',x,'ZData',fourierX,'CData',fourierX)
xlabel('time','FontSize',15);
ylabel('x','FontSize',15);
zlabel('probability','FontSize',15);
axis([0 tSteps(end) -pi pi 0 1])
view(-55,25)
```

sumBouncingFourierSer.m

```
function serSoln = sumBouncingFourierSer(x,t,n)
%this is a function for summing up all rest in the cosine term
%of the fourier series solution we got.
%t - the time is fixed. We want to calculte for a specified time.
series = 0;
for count=1:n
    thisTime = cos((count)*x)*exp(-((count)^2)*t);
    series = series + thisTime;
end
serSoln = 1/(2*pi)+(1/pi)*series;
    %remember the 1/pi.
```

D Source Code for chapter 3

D.1 Unbounded 1D random walk

simulate1dRandomWalk.m

```
clear, clf;
tic

T=600;
sim=2000;
simX=100;
arrivals=[];
for simme=1:simX
    paths=2*binornd(1,.5,sim,T)-1;
    paths=cumsum(paths,2);
    arrivals=[arrivals; paths(:,end)];
    clear paths;
end
y=ceil(4*sqrt(T));
x=[-y:2:y];% if misplaced try x=[-y+1:2:y-1]
figure(1)
plot(x,histc(arrivals(1:end),x)/(2*sim*simX),'Color','blue')
%(the 2 in the denominator comes from the step-length 2 in x.)
title(['1D Random Walk, T=' num2str(T) ', Sim=' num2str(sim*simX) '.'],'FontSize',15);
xlabel(['arrival position'],'FontSize',15);
ylabel(['percentage'],'FontSize',15);
hold on
plot(x,normpdf(x,0,(sqrt(T))),'Color','red')
legend('Random Walks','Normal PDF');
toc
```

D.2 Terminating boundaries 1D random walk

D.3 1D random walk with one-step bounce

fiveRandomWalkBounce1.m

```
%\begin{verbatim}

% Monte Carlo simulations of the 1D random walk.
% Because many of the cases are so similar I have
% put most of the files in the same place and just commented
% out the parts you need to add in order to get the file
% working for your particular need.
```

```
clear;
%set t0 for the clocktimer and c0 for the cputimer.
t0 = clock;

%Define the total number of simulations
Simulations = 200;
%define the width (N) and the time (T) of the river
N = 20;
T = 201;
%the witdh to use in the distrubation histogram for the arrivals
distrW = N;
%how many simulations to plot. to plot the whole lot uses a whole lot of
%memmory.
plottingCurves = 5;

%Allocate memmory for the array that will contain the path it will be a 1
%by T array since t='the place in the array' it needs not be incorporated:
path = zeros(T, Simulations);

  for sim = 1:Simulations
      %(re)Set the starting positions for each run, x is the displacement
      %in the N-direction and t in the T-direction.
      x = 0;
      t = 0;

      %now, need reset the bounce parameter for each simulation
      %%%normal random walk has no bounce so set it always equal to zero

      bounce = 0;

      %We have to start at t=2 since t=0 can not be used matrix indexing
      %and t=1 would mean x=+-1 and we want x=0.
      for t = 2:T
          y = rand;


          %-with one-step bounce.
          %now we want drift. Want drift to aim at 0 at end. i.e. add a
          %stochastic drift of N/(T-t).
          %this drif is not to be introduced if we are already on the
          %last probability. So;
          if t < T
              if x >= N
                   bounce = 1/2;
              elseif x <= -N
                   bounce = -1/2;
```

```
                %elseif x == 0
            else
                bounce = 0;
            end
        elseif t == T
            bounce = 0;
        end
          %-end of with bounce.

        if y > (1/2 + bounce)
            x = x + 1;
            %its a very small probability that y == 0.5 but it might
            %actually make a tiny difference and thus i added this
            %extra bit. There should not really be possible
            %for y==0.5 twice!
            elseif y == (1/2 + bounce)
                    y = rand;
                    if y > (1/2 + bounce)
                        x = x + 1;
                    else
                       x = x - 1
                    end
            else
                x = x - 1;
            end
      %Save the data in the path matrix.
      path(t,sim) = x;
      %end of this step.
    end
    %end of this Simulation, save the arrival location.
    arrivals = path(T,:);
end


allarrivals = [transpose(arrivals)];

subplot(2,2,1:2)
plot(path(:,1:plottingCurves));
axis([0 T -N N]);
%had to add a line-break here for the LaTeX. take it away to run
title([int2str(Simulations) ' simulations (plotting ' int2str(plottingCurves) '), T=' in
subplot(2,2,4)
%to generate the uniform sample to compare with in the qqplot.
for apa=1:Simulations
    apan(apa)=rand;
end
```

```
qqplot(arrivals, apan)
subplot(2,2,3)
binsN = -distrW:2:distrW;
hist(allarrivals,binsN);
%had to add a line-break here for the LaTeX. take it away to run
title(['Arrival spread. \sigma = ' num2str(mean(std(allarrivals))) '& \mu = ' num2str(me

%h = 0 if there is a good fit to any normal distribution.
[h p l c] = jbtest(arrivals);

%prints the spot estimates and 95% confidence intervalls for
%std and mean.

[mu,sigma,muci,sigmaci] = normfit(arrivals)

%timing - just to see how fast the computer is! ;)
etime(clock,t0)


%\end{verbatim}
```

E Source Code for chapter 4

E.1 1D random walk with target fixation drift

change the bounce on line 53 and 60 in the fiveRandomWalkLookBack1.m below.

E.2 1D random walk with look-back drift

RandomWalkLookBackVect1.m

```
%by Anders Österling, University of Stockholm, 2007.
tic
clear;
Simulations = 30000;
T = 999;
N = 20;%this is only half the field.


P = 0.5*ones(Simulations,1);
upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
Pos = zeros(Simulations,1)+upDown; %setting the position at T=0. Since the lowest vector
countBounce = zeros(Simulations,1);
drift = P;%.5 is means no drift.
saving=zeros(Simulations,1);

one = ones(Simulations,1);%for application speed, define ones.
for stepping=1:T-N;

    bouncing = zeros(Simulations,1);%reset the bounce.
    bouncing = -(Pos<=(-N)) + (Pos>=N);%bounce coefficients. for the drift.
    %Now we have to choose - uncomment the one that you want to use:
    %TARGET FIXATION
    %bounceDrift = .5.*abs(bouncing) - (bouncing.*N)./(2*(T-stepping));%Probabilities of
    %LOOK BACK
    bounceDrift = .5.*abs(bouncing) - (bouncing.*N)./(2*(stepping));%Probabilities of st

    drift = drift.*(one-abs(bouncing)) + bounceDrift;

    %if process crosses middle - take away drift - i.e. set drift =.5.
    %uncomment the next line if you want drift taken away at x=0.
    %drift=(1-(Pos==0)).*drift+(Pos==0)*.5;


    upDown = 2*binornd(1,drift,Simulations,1) -1;
    Pos = Pos + upDown;
    Pos = Pos.*(one-(Pos>N))+(N-1)*(Pos>N);%bouncing down
```

```
    Pos = Pos.*(one-(Pos<-N))-(N-1)*(Pos<-N);%bouncing up

end

aa=[-N:2:N];
histogram=histc(Pos(:,end),aa)/Simulations;
plot(aa,histogram, 'Color','green')
hold on
axis([-N N 0 .1])
legend('drift taken away at x=0','drift intact')
xlabel('arrival position','FontSize',15)
ylabel('probability','FontSize',15)
toc
```

E.3 Bounce counting function for target fixation

countBounces2.m

```
%COUNTBOUNCES - Count the number of times a process bounces
%               on the walls (with target fixation drift)
%
%countBounces(Sim,T,N) returns a Sim-by-1 vector with the number of times
%each process has bounced in a field that is N units wide and T units long.
%
%by Anders Österling, University of Stockholm, 2006.


function output = countBounces2(Simulations, T, N)
% clear;
% Simulations = 100;
% T = 1000;
% N = 10;%twice the width of the field!


    P = 0.5*ones(Simulations,1);
    upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
    Pos = zeros(Simulations,1)+upDown; %setting the position at T=0. Since the lowest ve
    countBounce = zeros(Simulations,1);
    drift = P;%.5 is means no drift.
    saving=zeros(Simulations,1);

    for stepping=0:T-N;

        one = ones(Simulations,1);%for application speed, define ones.
        bouncing = zeros(Simulations,1);%reset the bounce.
        bouncing = -(Pos<=(-N)) + (Pos>=N);%bounce coefficients. for the drift.
```

```
        countBounce = countBounce + abs(bouncing);
        bounceDrift = .5.*abs(bouncing) - (bouncing.*N)./(2*(T-stepping));%Probabilities
        drift = drift.*(one-abs(bouncing)) + bounceDrift;
        upDown = 2*binornd(1,drift,Simulations,1) -1;
        Pos = Pos + upDown;
        Pos = Pos.*(one-(Pos>N))+(N-1)*(Pos>N);%bouncing down
        Pos = Pos.*(one-(Pos<-N))-(N-1)*(Pos<-N);%bouncing up

        %saving = [saving Pos];
    end
    output = countBounce;
```

plotBounces2.m

```
clear, clf;

Simulations = 4000; %number of simulations to run for every timestep
N = 20; % width of the field
times = [100:200:2000];%vector with the different timesteps to try
means=zeros(length(times),3);
tic;
for loop = 1:length(times)
    data = countBounces2(Simulations,times(loop),N);
    means(loop,1)=mean(data);
    sorted = sort(data);
 %   means(loop,2)=sorted(round(length(sorted)/100*2.5));
 %   means(loop,3)=sorted(round(length(sorted)/100*97.5));
end
toc
figure(1)
plot(times,means(:,1),'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','
hold on
%plot(times,means(:,2),'--rs','LineWidth',2,'MarkerEdgeColor','blue','MarkerFaceColor','
%plot(times,means(:,3),'--rs','LineWidth',2,'MarkerEdgeColor','green','MarkerFaceColor',
xlabel(['Number of discretee timesteps']);
ylabel(['Average number of bounces']);
title(['Plotting Bounces for fixed witdh N=2*' num2str(20) ' and varying the Timesteps. (
```

E.4 1.5D random walk

seven2dWalk1.m

```
\begin{verbatim}
clear;
```

```
t0 = clock;
Simulations = 100000;
N = 100;
T = N;
%when using N direction, note that max(intervalN)==N/2.
%the interval limits DOES include limit points.
intervaln = 65;
intervalN = 95;
%direction:
% N == 0
% Tp == 1
% Tm == 2
% ,where Tp means Tplus == when x is positive, and Tm == Tminus, x <0.
direction = 1;

arrivals = zeros(Simulations,3);

for sim = 1:Simulations
    x = 0;
    t = 0;
    ttime = 0;
    alive = 1;

    while alive == 1
        ttime = ttime + 1;
        y = rand;
        if y < (1/3)
            x = x-1;
        elseif (y < (2/3))
            x = x+1;
        else
            t = t+1;
        end

        if x == N/2
            alive = 0;
        elseif x == -N/2
            alive = 0;
        elseif t == T
            alive = 0;
        end
    end %alive
    arrivals(sim,1) = ttime;
    arrivals(sim,2) = x;
    arrivals(sim,3) = t;
end %sim
```

```
goalnr = [0];
goalarr = zeros(1,2);
if(direction == 0) % at the far end of the box.
    for apa=1:Simulations
        if(arrivals(apa,3)==T)
            if(arrivals(apa,2) <= intervalN)
                if(arrivals(apa,2) >= intervaln)
                    goalarr = [goalarr; arrivals(apa,1), arrivals(apa,2)];
                end
            end
        end
    end
elseif(direction == 1) % the left (positive) side of box.
    'hej1'
    for apa=1:Simulations
        'hej 2'
        goalnr = [goalnr; arrivals(apa,2)];
        if(arrivals(apa,2)==(N/2))
            'hej 3'

            if(arrivals(apa,3) <= intervalN)
                if(arrivals(apa,3) >= intervaln)
                    goalarr = [goalarr; arrivals(apa,1), arrivals(apa,3)];
                end
            end
        end
    end
elseif(direction == 2) % the right (negative) side of the box.
    for apa=1:Simulations
        if(arrivals(apa,2)==-(N/2))
            if(arrivals(apa,3) <= intervalN)
                if(arrivals(apa,3) >= intervaln)
                    goalarr = [goalarr; arrivals(apa,1), arrivals(apa,3)];
                end
            end
        end
    end
end


goalarr = goalarr(2:end,:);

compTime = (t0 - clock);

subplot(2,2,1:2)
```

```
if(direction == 0)
    limiten = round((N+1)/2)
    binsW = -limiten:1:limiten;
else
    binsW = 0:1:T;
end
hist(goalarr(:,2),binsW);
%add the lines together and remove %-sign
%title([num2str(Simulations) ' simulations, ' num2str(length(goalarr(:,2))) '
    %in goal. Arrival Positions: \sigma = ' num2str(mean(std(goalarr(:,2))))
    %'& \mu = ' num2str(mean(mean(goalarr(:,2))))]);

subplot(2,2,3:4)
binsN = min(goalarr(:,1)):1:max(goalarr(:,1));
hist(goalarr(:,1),binsN);
%add the lines together and remove %-sign
%title(['Arrival Times: \sigma = ' num2str(mean(std(goalarr(:,1)))) '&
    %\mu = ' num2str(mean(mean(goalarr(:,1)))) '. Calcualtion time: '
    %num2str(etime(clock,t0))]);

-compTime(6)
etime(clock,t0)
\end{verbatim}
```

F Source Code for chapter 5

F.1 Coastal harbour

anovaAlpha.m

```
clear;
sim=100000;
N=20;%number of steps (flashes +2)
flashes=N-2;%no flash at beginning or at end.
endtime=400;
theta=pi/200;%degrees.
T=endtime/flashes;
ar = zeros(sim,N);
vvar= zeros(1,N);
ar(:,1)=normrnd(0,sqrt(T),sim,1);
vvar(:,1)=var(ar(:,1));
for f=2:N
    xi_f=(((N-2)-(f-2))/(N-2))*ar(:,(f-1));%xi_f
    theta_f=atan((ar(:,f-1)-xi_f)/T);
    theta_f=(theta_f>theta)*theta+(1-(theta_f>theta)).*theta_f;
    xi_f=ar(:,f-1)-T*tan(theta_f);

    ar(:,f)=normrnd(xi_f,sqrt(T),sim,1);
    vvar(1,f) = var(ar(:,f));
    end
vvar
plot(vvar, 'Color', 'yellow')
xlabel('Number of flashes','FontSize',15);
ylabel('Variance','FontSize',15);
hold on
```

firehouse1.m

```
%COUNTBOUNCES - Count the number of times a process bounces
%               on the walls (with target fixation drift)
%
%countBounces(Sim,T,N) returns a Sim-by-1 vector with the number of times
%each process has bounced in a field that is N units wide and T units long.
%
%by Anders Österling, University of Stockholm, 2006.


function output = firehouse1(Simulations, T,flashes)
% clear;
% Simulations = 100;
% T = 1000;
```

```
% N = 10;%twice the width of the field!


    P = 0.5*ones(Simulations,1);
    upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
    Pos = zeros(Simulations,1)+upDown; %setting the position at T=0. Since the lowest ve
    countBounce = zeros(Simulations,1);
    drift = P;%.5 is means no drift.
    saving=zeros(Simulations,1);

    flashCount = 1;

    flashTimes=round([(T/(flashes+1)):(T/(flashes+1)):((flashes*T)/(flashes+1))]);

    %var(T)~=sqrt(T)
%    for stepping=0:round(T-sqrt(T));
    for stepping=0:T;

        one = ones(Simulations,1);%for application speed, define ones.
        bouncing = zeros(Simulations,1);%reset the bounce.

        upDown = 2*binornd(1,drift,Simulations,1) -1;
        Pos = Pos + upDown;
        if(stepping==flashTimes(flashCount))
            %calculate new drift...
            %if the angle is less than one, use it for drift
            drifting = abs(Pos/(2*(T-stepping)))<1;
            %if the angle is greater than one, use 1 for drift.
            driftingLate = .5*(one-drifting);

            drift = .5*one - drifting.*(Pos/(2*(T-stepping)))-driftingLate;
            disp([num2str(stepping) ' var' num2str(var(drift))]);

            if(flashCount<length(flashTimes))
                flashCount = flashCount +1;
            end

        end

        upDown = 2*binornd(1,drift,Simulations,1) -1;
        Pos = Pos + upDown;

    end
    output = Pos;
```

plotFirehouse1.m

```
clear, clf;

Simulations = 20000; %number of simulations to run for every timestep
times = 400;%[100:200:2000];%vector with the different timesteps to try
harbour = [-10 10];%the lower/upper bound to check C.I for.
means=zeros(length(times),3);
tic
flashes=[1:10:81];%number of firehouse flashes;

data=[];

for loop = 1:length(flashes)
    ciCount=0;
    arrivals = firehouse1(Simulations,times,flashes(loop));
    outside = sum(arrivals<=harbour(1)) + sum(arrivals>=harbour(2));
    inside=Simulations-outside;
    data = [data inside/Simulations];
end

toc
figure(1)
plot(flashes,data,'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','Mark
hold on
line([0:10/flashes(end):flashes(end)],0.95,'LineWidth',2);
%plot(times,means(:,2),'--rs','LineWidth',2,'MarkerEdgeColor','blue','MarkerFaceColor','
%plot(times,means(:,3),'--rs','LineWidth',2,'MarkerEdgeColor','green','MarkerFaceColor',
xlabel(['Number of flashes']);
ylabel(['Percent ariving within (' num2str(harbour(1)) ',' num2str(harbour(2)) ').']);
title(['Plotting harbour-arrivals. Calculation time  ' num2str(toc) 'sec for ' num2str(S
%hist(data)
%nästa steg är att kör för fixade grunkor. fett dude!
```

F.2 Harbour at the end of a Norwegian Fjord

firehouseFjord1.m

```
%COUNTBOUNCES - Count the number of times a process bounces
%               on the walls (with target fixation drift)
%
%countBounces(Sim,T,N) returns a Sim-by-1 vector with the number of times
%each process has bounced in a field that is N units wide and T units long.
%
%by Anders Österling, University of Stockholm, 2006.
```

```
function output = firehouse1(Simulations, T,N,flashes)

    P = 0.5*ones(Simulations,1);
    upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
    Pos = zeros(Simulations,1)+upDown; %setting the position at T=0. Since the lowest ve
    drift = P;%.5 is means no drift.

    flashCount = 1;

    flashTimes=round([(T/(flashes+1)):(T/(flashes+1)):((flashes*T)/(flashes+1))]);

    for stepping=0:T;

        one = ones(length(Pos),1);%for application speed, define ones.

        if(stepping==flashTimes(flashCount))
            %calculate new drift...
            %if the angle is less than one, use it for drift
            drifting = abs(Pos/(2*(T-stepping)))<1;
            %if the angle is greater than one, use 1 for drift.
            driftingLate = .5*(one-drifting);

            drift = .5*one - drifting.*(Pos/(2*(T-stepping)))-driftingLate;

            if(flashCount<length(flashTimes))
                flashCount = flashCount +1;
            end

        end
        %terminating at the boundries.
        u=(Pos<N);%1's for all Pos <N
        u=1./u/Inf;%0's forall Pos<N, NaN 4A Pos>=N.
        l=(Pos>-N);
        l=1./l/Inf;
        Pos = Pos + u + l;%NaN 4A pos>N,pos<-N.
        Pos=Pos(finite(Pos));
        drift=drift+u+l;
        drift=drift(finite(drift));

       % disp(['drift ' num2str(size(drift)) ', pos ' num2str(size(Pos)) ',upDown ' num2
        upDown = 2*binornd(1,drift,length(Pos),1) -1;
        Pos = Pos + upDown;%+lower+upper;%add the NaN's

    end
```

```
    output = Pos;
```

plotFirehouseFjord1.m

```
clear, clf;

Simulations = 10000; %number of simulations to run for every timestep
times = 1000;%[100:200:2000];%vector with the different timesteps to try
N=45;
harbour = [-10 10];%the lower/upper bound to check C.I for.
tic
%NOTE. One needs to think about before deciding how many flashes to use
%and where to put them Putting too many in will mean they are at the
%same place and not all of them are actually used, thus getting less
%actually working lighthouses!
flashes=[1:20:101];%number of firehouse flashes;

data=[];

for loop = 1:length(flashes)
    ciCount=0;
    arrivals = firehouseFjord1(Simulations,times,N,flashes(loop));
    outside = sum(arrivals<=harbour(1)) + sum(arrivals>=harbour(2));
    length(arrivals)
    inside=length(arrivals)-outside;
    data = [data inside/Simulations];
end

toc
figure(1)
plot(flashes,data,'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','Mark
hold on
line([0:10/flashes(end):flashes(end)],0.95,'LineWidth',2);
%plot(times,means(:,2),'--rs','LineWidth',2,'MarkerEdgeColor','blue','MarkerFaceColor','
%plot(times,means(:,3),'--rs','LineWidth',2,'MarkerEdgeColor','green','MarkerFaceColor',
xlabel(['Number of flashes']);
ylabel(['Percent ariving within (' num2str(harbour(1)) ',' num2str(harbour(2)) ').']);
title(['Plotting fjord-arrivals, the fjord is 2*' num2str(N) ' wide,calc time ' num2str(
%hist(data)
%nästa steg är att kör för fixade grunkor. fett dude!
```

F.3 A wider fjord is easier to sail through

firehouseFjordWidthCI.m

```
%COUNTBOUNCES - Count the number of times a process bounces
%               on the walls (with target fixation drift)
%
%countBounces(Sim,T,N) returns a Sim-by-1 vector with the number of times
%each process has bounced in a field that is N units wide and T units long.
%
%by Anders Österling, University of Stockholm, 2006.
function output = firehouse1(Simulations, T,N,flashes)

P = 0.5*ones(Simulations,1);
upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
Pos = zeros(Simulations,1)+upDown; %setting the position at T=0. Since the lowest vector
drift = P;%.5 is means no drift.
flashCount = 1;
flashTimes=round([(T/(flashes+1)):(T/(flashes+1)):((flashes*T)/(flashes+1))]);

for stepping=0:T;
    one = ones(length(Pos),1);%for application speed, define ones.
    if(stepping==flashTimes(flashCount))
        %calculate new drift...
        %if the angle is less than one, use it for drift
        drifting = abs(Pos/(2*(T-stepping)))<1;
        %if the angle is greater than one, use 1 for drift.
        driftingLate = .5*(one-drifting);

        drift = .5*one - drifting.*(Pos/(2*(T-stepping)))-driftingLate;

        if(flashCount<length(flashTimes))
            flashCount = flashCount +1;
        end

    end
    %terminating at the boundries.
    u=(Pos<N);%1's for all Pos <N
    u=1./u/Inf;%0's forall Pos<N, NaN 4A Pos>=N.
    l=(Pos>-N);
    l=1./l/Inf;
    Pos = Pos + u + l;%NaN 4A pos>N,pos<-N.
    Pos=Pos(finite(Pos));
    drift=drift+u+l;
    drift=drift(finite(drift));
    if(length(Pos)==0)%if all process are terminated, quit.
        break;
    end
    upDown = 2*binornd(1,drift,length(Pos),1) -1;
    Pos = Pos + upDown;%+lower+upper;%add the NaN's
```

```
end
output = Pos;
```

plotFirehouseFjordWidthCI.m

```
tic
clear, clf;

Simulations = 10000; %number of simulations to run for every timestep
times = 1000;%[100:200:2000];%vector with the different timesteps to try
%N=45;
widths=[45:5:85];
CI=0.95;%how many need to pass for beeing OK.
harbour = [-10 10];%the lower/upper bound to check C.I for.
flashes=[30:2:60 61:5:86];%number of firehouse flashes;

optimalFlashes=zeros(length(widths),1);


for Nloop=1:length(widths)
    for loop = 1:length(flashes)
        ciCount=0;
        arrivals = firehouseFjord1(Simulations,times,widths(Nloop),flashes(loop));
        outside = sum(arrivals<=harbour(1)) + sum(arrivals>=harbour(2));
        inside=length(arrivals)-outside;
        if((inside/Simulations)>CI)
            optimalFlashes(Nloop)=flashes(loop);
            break;
        end
    end
end

figure(1)
plot(2*widths,optimalFlashes,'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor
xlabel(['Fjord witdh'],'FontSize',15);
ylabel(['Number of flashes'],'FontSize',15);
toc
```

F.4 The boundaries are linear but at an angle

plotSmallerGoalRW.m

```
%This, smallerGoalRW.m plots the number of simulations that arrive within
%different sizes of the goal. There is no boundarie what so ever.
clear, clf;
```

```
Simulations = 10000; %number of simulations to run for every timestep
times = 1000;%[100:200:2000];%vector with the different timesteps to try
N=50;
M=[5:10:45];

tic
%NOTE. One needs to think about before deciding how many flashes to use
%and where to put them Putting too many in will mean they are at the
%same place and not all of them are actually used, thus getting less
%actually working lighthouses!
%flashes=[1:20:101];%number of firehouse flashes;

data=[];

for loop = 1:length(M)
    ciCount=0;
    arrivals = smallerGoalRW(Simulations,times,N,0);
    harbour = [-M(loop) M(loop)];%the lower/upper bound to check C.I for.
    outside = sum(arrivals<=harbour(1)) + sum(arrivals>=harbour(2));
    inside=length(arrivals)-outside;
    data = [data inside/Simulations];
end

toc
figure(1)
plot(M./N,data,'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','MarkerS
hold on
%line([0:10/M(end):M(end)],0.95,'LineWidth',2);
%plot(times,means(:,2),'--rs','LineWidth',2,'MarkerEdgeColor','blue','MarkerFaceColor','
%plot(times,means(:,3),'--rs','LineWidth',2,'MarkerEdgeColor','green','MarkerFaceColor',
xlabel(['Width of goal (in percent of the start-field)']);
ylabel(['Percent ariving in the goal']);
title(['Plotting non-barrier goals, start field is 2*' num2str(N) ' wide,calc time ' num
%hist(data)
%nästa steg är att kör för fixade grunkor. fett dude!
```

smallerGoalRW.m

```
%COUNTBOUNCES - Count the number of times a process bounces
%               on the walls (with target fixation drift)
%
%countBounces(Sim,T,N) returns a Sim-by-1 vector with the number of times
%each process has bounced in a field that is N units wide and T units long.
%
%by Anders Österling, University of Stockholm, 2006.
```

```
function output = smallerGoalRW(Simulations, T,N,flashes)

    P = 0.5*ones(Simulations,1);
    upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
    Pos = zeros(Simulations,1)+upDown; %setting the position at T=0. Since the lowest ve
    drift = P;%.5 is means no drift.

% %  dubbel comment for lighthouse signals.
% %     flashCount = 1;
% %
% %     flashTimes=round([(T/(flashes+1)):(T/(flashes+1)):((flashes*T)/(flashes+1))]);
% %
    for stepping=0:T;

        one = ones(length(Pos),1);%for application speed, define ones.

% %         if(stepping==flashTimes(flashCount))
% %             %calculate new drift...
% %             %if the angle is less than one, use it for drift
% %             drifting = abs(Pos/(2*(T-stepping)))<1;
% %             %if the angle is greater than one, use 1 for drift.
% %             driftingLate = .5*(one-drifting);
% %
% %             drift = .5*one - drifting.*(Pos/(2*(T-stepping)))-driftingLate;
% %
% %             if(flashCount<length(flashTimes))
% %                 flashCount = flashCount +1;
% %             end
% %
% %         end
        %terminating at the boundries.
%         u=(Pos<N);%1's for all Pos <N
%         u=1./u/Inf;%0's forall Pos<N, NaN 4A Pos>=N.
%         l=(Pos>-N);
%         l=1./l/Inf;
%         Pos = Pos + u + l;%NaN 4A pos>N,pos<-N.
%         Pos=Pos(finite(Pos));
% %         drift=drift+u+l;
% %         drift=drift(finite(drift));
drift =.5*one;

       % disp(['drift ' num2str(size(drift)) ', pos ' num2str(size(Pos)) ',upDown ' num2

        upDown = 2*binornd(1,drift,length(Pos),1) -1;
        Pos = Pos + upDown;%+lower+upper;%add the NaN's
```

```
    end
    output = Pos;
```

angleBarrier.m

```
%COUNTBOUNCES - Count the number of times a process bounces
%               on the walls (with target fixation drift)
%
%countBounces(Sim,T,N) returns a Sim-by-1 vector with the number of times
%each process has bounced in a field that is N units wide and T units long.
%
%by Anders Österling, University of Stockholm, 2006.


function output = angleBarrier(Simulations, T,N,flashes,barrier)

    P = 0.5*ones(Simulations,1);
    upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
    Pos = zeros(Simulations,1)+upDown; %setting the position at T=0. Since the lowest ve
    drift = P;%.5 is means no drift.

% %  dubbel comment for lighthouse signals.
% %     flashCount = 1;
% %
% %     flashTimes=round([(T/(flashes+1)):(T/(flashes+1)):((flashes*T)/(flashes+1))]);
% %
    for stepping=0:T;

        one = ones(length(Pos),1);%for application speed, define ones.

% %         if(stepping==flashTimes(flashCount))
% %             %calculate new drift...
% %             %if the angle is less than one, use it for drift
% %             drifting = abs(Pos/(2*(T-stepping)))<1;
% %             %if the angle is greater than one, use 1 for drift.
% %             driftingLate = .5*(one-drifting);
% %
% %             drift = .5*one - drifting.*(Pos/(2*(T-stepping)))-driftingLate;
% %
% %             if(flashCount<length(flashTimes))
% %                 flashCount = flashCount +1;
% %             end
% %
% %         end
```

```
        %terminating at the boundries.
        u=(Pos<barrier(stepping+1));%1's for all Pos <N
        u=1./u/Inf;%0's forall Pos<N, NaN 4A Pos>=N.
        l=(Pos>-barrier(stepping+1));
        l=1./l/Inf;
        Pos = Pos + u + l;%NaN 4A pos>N,pos<-N.
        Pos=Pos(finite(Pos));
        drift=drift+u+l;
        drift=drift(finite(drift));
%drift =.5*one;

       % disp(['drift ' num2str(size(drift)) ', pos ' num2str(size(Pos)) ',upDown ' num2
       length(Pos),length(drift)
        upDown = 2*binornd(1,drift,length(Pos),1) -1;
        Pos = Pos + upDown;%+lower+upper;%add the NaN's

    end
    output = Pos;
```

plotAngleBarrier.m

```
%This, smallerGoalRW.m plots the number of simulations that arrive within
%different sizes of the goal. There is no boundarie what so ever.
clear, clf;

Simulations = 10000; %number of simulations to run for every timestep
times = 1000;%[100:200:2000];%vector with the different timesteps to try
N=50;
M=[5:10:45];

tic
%NOTE. One needs to think about before deciding how many flashes to use
%and where to put them Putting too many in will mean they are at the
%same place and not all of them are actually used, thus getting less
%actually working lighthouses!
%flashes=[1:20:101];%number of firehouse flashes;

data=[];

for loop = 1:length(M)
    ciCount=0;
    barrier=N-round([M(loop):(N-M(loop))/times:N])+M(loop);
    arrivals = angleBarrier(Simulations,times,N,0,barrier);
    harbour = [-M(loop) M(loop)];%the lower/upper bound to check C.I for.
```

```
    outside = sum(arrivals<=harbour(1)) + sum(arrivals>=harbour(2));
    inside=length(arrivals)-outside;
    data = [data inside/Simulations];
end

toc
figure(1)
plot(M./N,data,'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','MarkerS
hold on
%line([0:10/M(end):M(end)],0.95,'LineWidth',2);
%plot(times,means(:,2),'--rs','LineWidth',2,'MarkerEdgeColor','blue','MarkerFaceColor','
%plot(times,means(:,3),'--rs','LineWidth',2,'MarkerEdgeColor','green','MarkerFaceColor',
xlabel(['Width of goal (in percent of the start-field)']);
ylabel(['Percent ariving in goal']);
title(['Linear barrier to the goal, start field is 2*' num2str(N) ' wide,calc time ' num
%hist(data)
%nästa steg är att kör för fixade grunkor. fett dude!
```

G Source Code for chapter 6

G.1 Unbounded Theoretical solutions

plottaMatlabIntegral.m

```
clear;%clf;
tic
syms y;
a=[-10:.1:10]
intEnd=20;
b=a;
c=a;

for x=1:length(a)
    %This is a slow way of integrating the tails, but it was quick to
    %implement and it doesnt really mater that it takes a few seconds.
    b(x)=eval(int(1/(2*pi)*exp(-(a(x)^2)/2)*exp((a(x)*y-y^2)),1,intEnd));
    c(x)=eval(int(1/(2*pi)*exp(-(a(x)^2)/2)*exp((a(x)*y-y^2)),-intEnd,-1));
end
plot(a,c,'color','blue');
hold on
hold on
plot(a,b,'color','green');
hold on
plot(a,b+c,'color','red');

axis([ -6 6 0 .07])
grid on
legend('Left tail','Right tail','Sum of tails')
title(['Arrivals at t=T'],'FontSize',15)
xlabel(['x values'],'FontSize',15);
ylabel(['arival probability'],'FontSize',15);
toc
```

quasiMCRightAndLeftOfTheWater1.m

```
clf;
tic
simulations=250000;
simulations2=1;
atOne=randn(simulations,1);
left=atOne<-1;
right=atOne>1;
noLeftOf=sum(left);
noRightOf=sum(right);
```

```
atTwo=[];


leftTail=findnz(1*left);
a=1;
%leftAtTwo=[];
leftAtTwo=zeros(noLeftOf,1);
while(a<=length(leftTail(:,1)))
 %  leftAtTwo=[leftAtTwo; atOne(leftTail(a))+randn(simulations2,1)];
   leftAtTwo(a)=atOne(leftTail(a))+randn(simulations2,1);
   a=a+1;
end
rightTail=findnz(1*right);
a=1;
rightAtTwo=zeros(noRightOf,1);
while(a<=length(rightTail(:,1)))
%    rightAtTwo=[rightAtTwo; atOne(rightTail(a))+randn(simulations2,1)];
   rightAtTwo(a)=atOne(rightTail(a))+randn(simulations2,1);
    a=a+1;
end

size(atTwo)
noLeftOf+noRightOf

%figure(1)
normaliseradL=histc(leftAtTwo,[-5:1:5])/(1*simulations);
plot([-5:.1:5],normaliseradL,'color','blue');
hold on
normaliseradR=histc(rightAtTwo,[-5:1:5])/(1*simulations);
plot([-5:.1:5],normaliseradR,'color','green');
hold on
%normaliserad=histc(atTwo,[-5:.1:5])/(simulations);
%plot([-5:.1:5],normaliserad,'color','red');
%normaliserad=histc(atTwo,[-5:.1:5])/(simulations);
plot([-5:.1:5],normaliseradL+normaliseradR,'color','red');

axis([ -6 6 0 .07])
grid on
legend('Left tail','Right tail','Sum of tails')
title(['Arrivals at t=T'],'FontSize',15)
xlabel(['x values'],'FontSize',15);
ylabel(['arival percentage'],'FontSize',15);


toc
```

```
% subplot(2,2,1)
% plot(a,(b))
% grid on
% title(['Right side tail'],'FontSize',15);
% axis([ -5 5 0 .4])
% subplot(2,2,2)
% plot(a,(c))
% grid on
% title(['Left side tail'],'FontSize',15);
% axis([ -5 5 0 .4])
% subplot(2,2,[3 4])
% plot(a,(1./(sqrt(2*pi).*exp((a.^2)./2+1)).*(exp(a)+exp(-a))))
% grid on
% title(['Both tails together'],'FontSize',15);
% axis([ -5 5 0 .4])
```

G.1.1 Proper slalom, with more than one obstacle

noEdgesSlalomObstacles.m

```
%TERMINATEEDGESSLALOM - Finds the percent of simulations that survives a
%                       slalom path where there are cold barriers and
%                       terminating walls.
%
%terminateEdgesSlalom(Sim,T,N) returns a percentage of the Simulations that
%       survived until the end.
%
%by Anders Österling, University of Stockholm, 2006.


function output = terminateEdgesSlalomGates(Simulations, T, N, Gamma, delta, alfa)
% clear;
% Simulations = 1000;
% T = 100;
% N = 10;%twice the width of the field!
% Gamma = 8;
gateTimes = round([(T/(Gamma+1)):(T/(Gamma+1)):((Gamma*T)/(Gamma+1))]);%places for the w
% stepCount = 0;
% delta = N/5;%witdh of the wall
%alfa=(2*N-delta)*rand(Gamma,T)-N;%could have PI instead of N...

P = 0.5*ones(Simulations,1);
upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
Pos = upDown; %setting the position at T=0. Since the lowest vector

gates=1;%counting gates passed through
```

```
for stepping=3:T;
    one = ones(length(Pos),1);%for application speed, define ones.

    upDown = 2*binornd(1,P(1:length(Pos)),length(Pos),1) -1;
    %If Pos=[] the UpDown is a 1-by-0 matrix and dimensions do not agree.
    %thus we have to introduce this test.
    if(~isempty(Pos))
        Pos = Pos + upDown;

        %the code below uses the MATLAB feature that if you do a divide
        %by zero you get a 'Inf'. You can then delete these 'Inf' as we
        %do on the last line in the code. This way, everytime a process
        %is terminated it's taken out of the loop and we get less and less
        %data to process.
   %     Pos = Pos.*(one-(Pos>N))+1./(one-(Pos>N));%TERMINATING(bouncing down)
   %     Pos = Pos.*(one-(Pos<-N))-1./(one-(Pos<-N));%TERMINATING(bouncing up)

        Pos=Pos(finite(Pos));
        %checking the walls
        %if the process is inside of the gate, terminate it.
        if(stepping==gateTimes(gates))
            for process = 1:length(Pos)
                if(Pos(process,1)<=alfa(gates,process))
                    %                    Pos(process,1) = NaN;
                elseif(Pos(process,1)>=alfa(gates,process)+delta)
                    %                   Pos(process,1) = NaN;
                else
                    Pos(process,1) = NaN;
                end

        end
        if(gates<length(gateTimes))
            gates = gates +1;
        end
    end



    end
end
output = length(Pos)/Simulations;
```

slalomObstaclesTimeNoEdge.m

```
clear, clf;
tic
% slalomGates.m - program to find how the survivalrates
```

```
% depend on the length of the timefield, in a process where
% we have cold barriers and also walls that we might hit.

%terminateEdgesSlalomGates(Simulations, T, N, Gamma, delta)

Simulations = 10000;%number of simulations for each run.
times = [500:1000:6500];%how many and which times should we check.
%times=1000;
N=50;%the width of the fiels is 2*N;
Gamma = 10; %the number of small walls to use;
delta = 10;%round([(N/(Gamma)):(N/Gamma):(N)]); %the width of small gates;
data = [];

for loop = 1:length(times)
%loop=1;
    %alfa is a matrix numberofGates*numberOfSimulations and contains the
    %random gate-widths.
    alfa=(2*N-delta)*rand(Gamma,Simulations)-N;%could have PI instead of N...
    data = [data terminateEdgesSlalomObstacles(Simulations, times(loop), N, Gamma, delta
end
%    data
figure(1)
plot(times,data,'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','Marker
xlabel(['Length (T) of the field (discretee timesteps)'],'FontSize',14);
ylabel(['Survival-percent'],'FontSize',14);
title(['Plot of survival-function \rho(T) .'],'FontSize',14);
hold off
data
toc
```

G.1.2 Bigger holes implies less drunken men

slalomObstaclesNoEdgeDifferentSizeRocks.m

```
clear, clf;
tic
% slalomGates.m - program to find how the survivalrates
% depend on the length of the timefield, in a process where
% we have cold barriers and also walls that we might hit.

%terminateEdgesSlalomGates(Simulations, T, N, Gamma, delta)

Simulations = 10000;%number of simulations for each run.
times = 1000;% [100:100:800];%how many and which times should we check.
N=50;%the width of the fiels is 2*N;
```

```
Gamma = 10; %the number of small walls to use;
delta = round([(N/(Gamma)):(N/Gamma):(N)]); %the width of small gates;
%delta=10;
data = [];

for loop = 1:length(delta)
%loop=1;
    alfa=(2*N-delta(loop))*rand(Gamma,Simulations)-N;%could have PI instead of N...
    data = [data noEdgesSlalomObstacles(Simulations, times, N, Gamma, delta(loop),alfa)]
end
%    data
figure(1)
plot(delta,data,'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','Marker
xlabel(['Width (\delta) of the obstacles'],'FontSize',14);
ylabel(['Survival-percent'],'FontSize',14);
title(['Plot of survival-function \rho(\delta) .'],'FontSize',14);
hold off
data
toc
```

G.1.3 Gates rather than holes

noEdgesSlalomGates.m

```
% tic
%TERMINATEEDGESSLALOM - Finds the percent of simulations that survives a
%                       slalom path where there are cold barriers and
%                       terminating walls.
%
%terminateEdgesSlalom(Sim,T,N) returns a percentage of the Simulations that
%       survived until the end.
%
%by Anders Österling, University of Stockholm, 2006.


function output = terminateEdgesSlalomGates(Simulations, T, N, Gamma, delta, alfa)
% clear;
% Simulations = 1000;
% T = 100;
% N = 10;%twice the width of the field!
% Gamma = 8;
gateTimes = round([(T/(Gamma+1)):(T/(Gamma+1)):((Gamma*T)/(Gamma+1))]);%places for the w
% stepCount = 0;
% delta = N/5;%witdh of the wall
%alfa=(2*N-delta)*rand(Gamma,T)-N;%could have PI instead of N...
```

```
P = 0.5*ones(Simulations,1);
upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
Pos = upDown; %setting the position at T=0. Since the lowest vector

gates=1;%counting gates passed through
for stepping=3:T;
    one = ones(length(Pos),1);%for application speed, define ones.

    upDown = 2*binornd(1,P(1:length(Pos)),length(Pos),1) -1;
    %If Pos=[] the UpDown is a 1-by-0 matrix and dimensions do not agree.
    %thus we have to introduce this test.
    if(~isempty(Pos))
        Pos = Pos + upDown;

        %the code below uses the MATLAB feature that if you do a divide
        %by zero you get a 'Inf'. You can then delete these 'Inf' as we
        %do on the last line in the code. This way, everytime a process
        %is terminated it's taken out of the loop and we get less and less
        %data to process.
        %Pos = Pos.*(one-(Pos>N))+1./(one-(Pos>N));%bouncing down
        %Pos = Pos.*(one-(Pos<-N))-1./(one-(Pos<-N));%bouncing up

        Pos=Pos(finite(Pos));
        %checking the walls
        %if the process is outside of the gate, terminate it.
        if(stepping==gateTimes(gates))
            for process = 1:length(Pos)
                if(Pos(process,1)<=alfa(gates,process))
                    Pos(process,1) = NaN;
                elseif(Pos(process,1)>=alfa(gates,process)+delta)
                    Pos(process,1) = NaN;
                end
            end

        end
        if(gates<length(gateTimes))
            gates = gates +1;
        end
    end



    end
end
output = length(Pos)/Simulations;
```

```
% disp(['' num2str(length(Pos)/Simulations) '% surviving']);
% toc
```

slalomGatesNoEdges.m

```
clear, clf;
tic
% slalomGates.m - program to find how the survivalrates
% depend on the length of the timefield, in a process where
% we have cold barriers and also walls that we might hit.

%terminateEdgesSlalomGates(Simulations, T, N, Gamma, delta)

Simulations = 10000;%number of simulations for each run.
times = 1000;% [100:100:800];%how many and which times should we check.
N=50;%the width of the fiels is 2*N;
Gamma = 10; %the number of small walls to use;
delta = [(2*N/Gamma):(2*N/Gamma):(2*N)]; %the width of small gates;
data = [];

for loop = 1:length(delta)
%loop=1;
    alfa=(2*N-delta(loop))*rand(Gamma,Simulations)-N;%could have PI instead of N...
    data = [data noEdgesSlalomGates(Simulations, times, N, Gamma, delta(loop),alfa)];
end
%    data
figure(1)
plot(delta,data,'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','Marker
xlabel(['Width (\delta) of the gates'],'FontSize',14);
ylabel(['Survival-percent'],'FontSize',14);
title(['Plot of survival-function \rho(\delta) .'],'FontSize',14);
hold off

toc
```

G.1.4 Nice slalom is easy, hard slalom is hard

noEdgesSlalomGatesNiceness.m

```
% tic
%TERMINATEEDGESSLALOM - Finds the percent of simulations that survives a
%                       slalom path where there are cold barriers and
%                       terminating walls.
%
```

```
%terminateEdgesSlalom(Sim,T,N) returns a percentage of the Simulations that
%       survived until the end.
%
%by Anders Österling, University of Stockholm, 2006.


function output = noEdgesSlalomGates(Simulations, T, N, Gamma, delta, alfa)
% clear;
% Simulations = 1000;
% T = 100;
% N = 10;%twice the width of the field!
% Gamma = 8;
gateTimes = round([(T/(Gamma+1)):(T/(Gamma+1)):((Gamma*T)/(Gamma+1))]);%places for the w
% stepCount = 0;
% delta = N/5;%witdh of the wall
%alfa=(2*N-delta)*rand(Gamma,T)-N;%could have PI instead of N...

P = 0.5*ones(Simulations,1);
upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
Pos = upDown; %setting the position at T=0. Since the lowest vector

gates=1;%counting gates passed through
for stepping=3:T;
    one = ones(length(Pos),1);%for application speed, define ones.

    upDown = 2*binornd(1,P(1:length(Pos)),length(Pos),1) -1;
    %If Pos=[] the UpDown is a 1-by-0 matrix and dimensions do not agree.
    %thus we have to introduce this test.
    if(~isempty(Pos))
        Pos = Pos + upDown;

        %the code below uses the MATLAB feature that if you do a divide
        %by zero you get a 'Inf'. You can then delete these 'Inf' as we
        %do on the last line in the code. This way, everytime a process
        %is terminated it's taken out of the loop and we get less and less
        %data to process.
        %Pos = Pos.*(one-(Pos>N))+1./(one-(Pos>N));%bouncing down
        %Pos = Pos.*(one-(Pos<-N))-1./(one-(Pos<-N));%bouncing up

        Pos=Pos(finite(Pos));
        %checking the walls
        %if the process is outside of the gate, terminate it.
        if(stepping==gateTimes(gates))
            for process = 1:length(Pos)
                if(Pos(process,1)<=(alfa(gates,1)-delta/2))
                    Pos(process,1) = NaN;
```

```
                elseif(Pos(process,1)>=(alfa(gates,1)+delta/2))
                    Pos(process,1) = NaN;
                end
            end

        end
        if(gates<length(gateTimes))
            gates = gates +1;
            %        disp(['t
        end
    end



    end
end
output = length(Pos)/Simulations;
% disp(['' num2str(length(Pos)/Simulations) '% surviving']);
% toc
```

slalomGatesNicenessNoEdges.m

```
clear, clf;
tic
% slalomGates.m - program to find how the survivalrates
% depend on the length of the timefield, in a process where
% we have cold barriers and also walls that we might hit.

%terminateEdgesSlalomGates(Simulations, T, N, Gamma, delta)

Simulations = 10000;%number of simulations for each run.
times = 1000;% [100:100:800];%how many and which times should we check.
N=50;%the width of the fiels is 2*N;
Gamma = 10; %the number of small walls to use;
delta = 10;%[(2*N/Gamma):(2*N/Gamma):(2*N)]; %the width of small gates;
data = [];
niceness=5;%number of different sigmas to use...
niceGates=[0:N/niceness:N-N/niceness];

%skapar en \pm 1 vektor...
inGates=zeros(Gamma,1);
for pm = 1:Gamma;
    if(~rem(pm,2))
        %even
        inGates(pm,1)=1;
```

```
    else
        inGates(pm,1)=-1;
    end
end
alfa=zeros(length(niceGates),1);
for gate = 1:length(niceGates)
    alfa=niceGates(gate).*inGates;%+(delta/2).*inGates%-delta/2

    data = [data noEdgesSlalomGatesNiceness(Simulations, times, N, Gamma, delta,alfa)];
end


figure(1)
plot(niceGates,data,'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','Ma
xlabel(['Gate offset from centre-line.'],'FontSize',14);
ylabel(['Survival-percent'],'FontSize',14);
title(['Plot of survival-function \rho(\delta) .'],'FontSize',14);
hold off

toc
```

G.2 Bounded Monte Carlo solutions

G.2.1 Hitting a rock will terminate you like a termite

slalomObstaclesTime.m terminateEdgesSlalomObstacles.m

```
%TERMINATEEDGESSLALOM - Finds the percent of simulations that survives a
%                       slalom path where there are cold barriers and
%                       terminating walls.
%
%terminateEdgesSlalom(Sim,T,N) returns a percentage of the Simulations that
%       survived until the end.
%
%by Anders Österling, University of Stockholm, 2006.


function output = terminateEdgesSlalomGates(Simulations, T, N, Gamma, delta, alfa)
% clear;
% Simulations = 1000;
% T = 100;
% N = 10;%twice the width of the field!
% Gamma = 8;
gateTimes = round([(T/(Gamma+1)):(T/(Gamma+1)):((Gamma*T)/(Gamma+1))]);%places for the w
% stepCount = 0;
% delta = N/5;%witdh of the wall
```

```
%alfa=(2*N-delta)*rand(Gamma,T)-N;%could have PI instead of N...

P = 0.5*ones(Simulations,1);
upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
Pos = upDown; %setting the position at T=0. Since the lowest vector

gates=1;%counting gates passed through
for stepping=3:T;
    one = ones(length(Pos),1);%for application speed, define ones.

    upDown = 2*binornd(1,P(1:length(Pos)),length(Pos),1) -1;
    %If Pos=[] the UpDown is a 1-by-0 matrix and dimensions do not agree.
    %thus we have to introduce this test.
    if(~isempty(Pos))
        Pos = Pos + upDown;

        %the code below uses the MATLAB feature that if you do a divide
        %by zero you get a 'Inf'. You can then delete these 'Inf' as we
        %do on the last line in the code. This way, everytime a process
        %is terminated it's taken out of the loop and we get less and less
        %data to process.
        Pos = Pos.*(one-(Pos>N))+1./(one-(Pos>N));%TERMINATING(bouncing down)
        Pos = Pos.*(one-(Pos<-N))-1./(one-(Pos<-N));%TERMINATING(bouncing up)

        Pos=Pos(finite(Pos));
        %checking the walls
        %if the process is inside of the gate, terminate it.
        if(stepping==gateTimes(gates))
            for process = 1:length(Pos)
                if(Pos(process,1)<=alfa(gates,process))
                    %                    Pos(process,1) = NaN;
                elseif(Pos(process,1)>=alfa(gates,process)+delta)
                    %                   Pos(process,1) = NaN;
                else
                    Pos(process,1) = NaN;
                end

        end
        if(gates<length(gateTimes))
            gates = gates +1;
        end
    end



    end
```

```
end
output = length(Pos)/Simulations;
```

G.2.2 The bigger they get, the harder it gets

slalomObstacles.m

```
clear, clf;
tic
% slalomGates.m - program to find how the survivalrates
% depend on the length of the timefield, in a process where
% we have cold barriers and also walls that we might hit.

%terminateEdgesSlalomGates(Simulations, T, N, Gamma, delta)

Simulations = 10000;%number of simulations for each run.
times = 1000;% [100:100:800];%how many and which times should we check.
N=50;%the width of the fiels is 2*N;
Gamma = 10; %the number of small walls to use;
delta = round([(N/(Gamma)):(N/Gamma):(N)]); %the width of small gates;
data = [];

for loop = 1:length(delta)
%loop=1;
    alfa=(2*N-delta(loop))*rand(Gamma,Simulations)-N;%could have PI instead of N...
    data = [data terminateEdgesSlalomObstacles(Simulations, times, N, Gamma, delta(loop)
end
%    data
figure(1)
plot(delta,data,'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','Marker
xlabel(['Width (\delta) of the obstacles'],'FontSize',14);
ylabel(['Survival-percent'],'FontSize',14);
title(['Plot of survival-function \rho(\delta) .'],'FontSize',14);
hold off
data
toc
```

G.2.3 More on stochastic slalom with gates to pass through

slalomGates.m

```
clear, clf;
tic
% slalomGates.m - program to find how the survivalrates
% depend on the length of the timefield, in a process where
% we have cold barriers and also walls that we might hit.
```

```
%terminateEdgesSlalomGates(Simulations, T, N, Gamma, delta)

Simulations = 10000;%number of simulations for each run.
times = 1000;% [100:100:800];%how many and which times should we check.
N=50;%the width of the fiels is 2*N;
Gamma = 10; %the number of small walls to use;
delta = [(2*N/Gamma):(2*N/Gamma):(2*N)]; %the width of small gates;
data = [];

for loop = 1:length(delta)
%loop=1;
    alfa=(2*N-delta(loop))*rand(Gamma,Simulations)-N;%could have PI instead of N...
    data = [data terminateEdgesSlalomGates(Simulations, times, N, Gamma, delta(loop),alf
end
%    data
figure(1)
plot(delta,data,'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','Marker
xlabel(['Width (\delta) of the gates'],'FontSize',14);
ylabel(['Survival-percent'],'FontSize',14);
title(['Plot of survival-function \rho(\delta) .'],'FontSize',14);
hold off

toc
```

terminateEdgesSlalomGates.m

```
% tic
%TERMINATEEDGESSLALOM - Finds the percent of simulations that survives a
%                       slalom path where there are cold barriers and
%                       terminating walls.
%
%terminateEdgesSlalom(Sim,T,N) returns a percentage of the Simulations that
%       survived until the end.
%
%by Anders Österling, University of Stockholm, 2006.


function output = terminateEdgesSlalomGates(Simulations, T, N, Gamma, delta, alfa)
gateTimes = round([(T/(Gamma+1)):(T/(Gamma+1)):((Gamma*T)/(Gamma+1))]);%places for the w

P = 0.5*ones(Simulations,1);
upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
Pos = upDown; %setting the position at T=0. Since the lowest vector

gates=1;%counting gates passed through
```

```
for stepping=3:T;
    one = ones(length(Pos),1);%for application speed, define ones.

    upDown = 2*binornd(1,P(1:length(Pos)),length(Pos),1) -1;
    %If Pos=[] the UpDown is a 1-by-0 matrix and dimensions do not agree.
    %thus we have to introduce this test.
    if(~isempty(Pos))
        Pos = Pos + upDown;

        %the code below uses the MATLAB feature that if you do a divide
        %by zero you get a 'Inf'. You can then delete these 'Inf' as we
        %do on the last line in the code. This way, everytime a process
        %is terminated it's taken out of the loop and we get less and less
        %data to process.
        Pos = Pos.*(one-(Pos>N))+1./(one-(Pos>N));%bouncing down
        Pos = Pos.*(one-(Pos<-N))-1./(one-(Pos<-N));%bouncing up

        Pos=Pos(finite(Pos));
        %checking the walls
        %if the process is outside of the gate, terminate it.
        if(stepping==gateTimes(gates))
            for process = 1:length(Pos)
                if(Pos(process,1)<=alfa(gates,process))
                    Pos(process,1) = NaN;
                elseif(Pos(process,1)>=alfa(gates,process)+delta)
                    Pos(process,1) = NaN;
                end
            end

        end
        if(gates<length(gateTimes))
            gates = gates +1;
            %        disp(['t
        end
    end
    end
end
output = length(Pos)/Simulations;
```

G.2.4 Hard vs. Easy slalom

slalomGatesNiceness.m

```
clear, clf;
tic
% slalomGates.m - program to find how the survivalrates
```

```
% depend on the length of the timefield, in a process where
% we have cold barriers and also walls that we might hit.

Simulations = 10000;%number of simulations for each run.
times = 1000;% [100:100:800];%how many and which times should we check.
N=50;%the width of the fiels is 2*N;
Gamma = 10; %the number of small walls to use;
delta = 10;%[(2*N/Gamma):(2*N/Gamma):(2*N)]; %the width of small gates;
data = [];
niceness=5;%number of different Deltas to use...
niceGates=[0:N/niceness:N-N/niceness];

%skapar en \pm 1 vektor...
inGates=zeros(Gamma,1);
for pm = 1:Gamma;
    if(~rem(pm,2))
        %even
        inGates(pm,1)=1;
    else
        inGates(pm,1)=-1;
    end
end
alfa=zeros(length(niceGates),1);
for gate = 1:length(niceGates)
    alfa=niceGates(gate).*inGates;%+(delta/2).*inGates%-delta/2

    data = [data terminateEdgesSlalomGatesNiceness(Simulations, times, N, Gamma, delta,a
end


figure(1)
plot(niceGates,data,'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','Ma
xlabel(['Gate offset from centre-line.'],'FontSize',14);
ylabel(['Survival-percent'],'FontSize',14);
title(['Plot of survival-function \rho(\Delta) .'],'FontSize',14);
toc
```

terminateEdgesSlalomGatesNiceness.m

```
% tic
%TERMINATEEDGESSLALOM - Finds the percent of simulations that survives a
%                       slalom path where there are cold barriers and
%                       terminating walls.
%
%terminateEdgesSlalom(Sim,T,N) returns a percentage of the Simulations that
%       survived until the end.
```

```
%
%by Anders Österling, University of Stockholm, 2006.


function output = terminateEdgesSlalomGates(Simulations, T, N, Gamma, delta, alfa)
gateTimes = round([(T/(Gamma+1)):(T/(Gamma+1)):((Gamma*T)/(Gamma+1))]);%places for the w
P = 0.5*ones(Simulations,1);
upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
Pos = upDown; %setting the position at T=0. Since the lowest vector

gates=1;%counting gates passed through
for stepping=3:T;
    one = ones(length(Pos),1);%for application speed, define ones.

    upDown = 2*binornd(1,P(1:length(Pos)),length(Pos),1) -1;
    %If Pos=[] the UpDown is a 1-by-0 matrix and dimensions do not agree.
    %thus we have to introduce this test.
    if(~isempty(Pos))
        Pos = Pos + upDown;

        %the code below uses the MATLAB feature that if you do a divide
        %by zero you get a 'Inf'. You can then delete these 'Inf' as we
        %do on the last line in the code. This way, everytime a process
        %is terminated it's taken out of the loop and we get less and less
        %data to process.
        Pos = Pos.*(one-(Pos>N))+1./(one-(Pos>N));%bouncing down
        Pos = Pos.*(one-(Pos<-N))-1./(one-(Pos<-N));%bouncing up

        Pos=Pos(finite(Pos));
        %checking the walls
        %if the process is outside of the gate, terminate it.
        if(stepping==gateTimes(gates))
            for process = 1:length(Pos)
                if(Pos(process,1)<=(alfa(gates,1)-delta/2))
                    Pos(process,1) = NaN;
                elseif(Pos(process,1)>=(alfa(gates,1)+delta/2))
                    Pos(process,1) = NaN;
                end
            end

        end
        if(gates<length(gateTimes))
            gates = gates +1;
            %        disp(['t
        end
    end
```

```
end
output = length(Pos)/Simulations;
```

G.3 How easy is easy?

slalomGatesVSwalls.m

```
clear, clf;
tic
% slalomGates.m - program to find how the survivalrates
% depend on the length of the timefield, in a process where
% we have cold barriers and also walls that we might hit.
Simulations = 12000;%number of simulations for each run.
times = 1000;% [100:100:800];%how many and which times should we check.
N=50;%the width of the fiels is 2*N;
Gamma = 10; %the number of small walls to use;
delta = [(2*N/Gamma):(2*N/Gamma):(2*N)]; %the width of small gates;
data1 = [];
data2=[];

for loop = 1:length(delta)
    alfa=-(delta(loop)/2)*ones(Gamma,Simulations);
    data1 = [data1 terminateEdgesSlalomGates(Simulations, times, N, Gamma, delta(loop),a
    data2=[data2 terminateEdgesSlalomWOGates(Simulations, times, delta(loop)/2, Gamma, a
end
%    data
figure(1)
plot(delta,data1,'--rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','Marke
hold on;
plot(delta,data2,'-.bs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','Marke
xlabel(['\delta'],'FontSize',15);
ylabel(['Survival-percent'],'FontSize',15);
title(['Plot of survival-function \rho(\delta) .'],'FontSize',14);
hold off
grid on
toc
```

terminateEdgesSlalomWOGates.m

```
% tic
%TERMINATEEDGESSLALOM - Finds the percent of simulations that survives a
%                       slalom path where there are cold barriers and
%                       terminating walls.
%
```

```
%terminateEdgesSlalom(Sim,T,N) returns a percentage of the Simulations that
%       survived until the end.
%
%by Anders Österling, University of Stockholm, 2006.
function output = terminateEdgesSlalomWOGates(Simulations, T, N, Gamma, alfa)
gateTimes = round([(T/(Gamma+1)):(T/(Gamma+1)):((Gamma*T)/(Gamma+1))]);%places for the w

P = 0.5*ones(Simulations,1);
upDown = 2*binornd(1,P,Simulations,1) -1; %Only -1 and 1 in UpDown.
Pos = upDown; %setting the position at T=0. Since the lowest vector

gates=1;%counting gates passed through
for stepping=3:T;
    one = ones(length(Pos),1);%for application speed, define ones.

    upDown = 2*binornd(1,P(1:length(Pos)),length(Pos),1) -1;
    %If Pos=[] the UpDown is a 1-by-0 matrix and dimensions do not agree.
    %thus we have to introduce this test.
    if(~isempty(Pos))
        Pos = Pos + upDown;

        %the code below uses the MATLAB feature that if you do a divide
        %by zero you get a 'Inf'. You can then delete these 'Inf' as we
        %do on the last line in the code. This way, everytime a process
        %is terminated it's taken out of the loop and we get less and less
        %data to process.
        Pos = Pos.*(one-(Pos>N))+1./(one-(Pos>N));%terminated
        Pos = Pos.*(one-(Pos<-N))-1./(one-(Pos<-N));%terminated

        Pos=Pos(finite(Pos));
        %checking the walls
        %if the process is outside of the gate, terminate it.
%         if(stepping==gateTimes(gates))
%             for process = 1:length(Pos)
%                 if(Pos(process,1)<=alfa(gates,process))
%                     Pos(process,1) = NaN;
%                 elseif(Pos(process,1)>=alfa(gates,process)+delta)
%                     Pos(process,1) = NaN;
%                 end
%             end
%
%         end
        if(gates<length(gateTimes))
            gates = gates +1;
            %        disp(['t
        end
```

```
    end



    end
end
output = length(Pos)/Simulations;
% disp(['' num2str(length(Pos)/Simulations) '% surviving']);
% toc
```

H Source Code for Appendix A

fiveTransProb1.m

```
%\begin{verbatim}
%This file contains the sourcecode for generating the convergence-figure of the
%transition probability matrix.


%NOTE I USE the statistical trans prob mx. in dynamics one would transpose
%the matrix.


%
%This file calculates the transition probabilities for a randomwalk.
%
%Variables;
%halfSize: This is how wide the mx will be, from the centre. so it'll be
%2*howWide+1 broad.
%howManySteps: The number of steps to take before starting the loop.
%incrSize: how big the increments should be (i.e how many steps longer the road should b
%incrMany: how many times one should increase the length of the road.


clear;
t0 = clock;
halfSize = 19;
% howManySteps is the variable where we strat.
howManySteps = 2;
% the increments for the 3d stuff
incrSize = 20;
incrMany = 10;
%howBroad need be an even number, so
howBroad = 2*halfSize;

A = zeros(howBroad+1 ,howBroad+1);
for row=0:howBroad
    for column=0:howBroad
        if abs(row-column)==1
            A(row+1,column+1)=1;
        end
    end
end

%Now. the indexing of A is a bit screwed. since we are using the both
%positive and negative numers, we need see the centre of the matrix as the
```

```
%centre, and then the tings on the left are the negative parts.
%
%also. the first and last entry in the mx needs be shifted. this is because
%when at the limit, it is not a 50/50 prob wether to exceed the limit or
%not.
%its a 100/0 probability that we will not exceed the wall.

%note. A is the adjacency matrix. We modify the edges to make it impossible
%for the process to go outside of A
%and divide by 2 to get the tranition probability matrix B.
% C is the matrix of the added trans prob mx's. I.e the prb that the process
%has sometime entered a certain level.
A(2,1)=2;
B = A/2;
B(column, row+1)=1;


H = zeros(howBroad+1,1)
binsW = -(halfSize):1:(halfSize);
for eskil=1:incrMany
    D = B^(howManySteps + eskil*incrSize);
    F = D(:,halfSize+1);
    H = [H F];
end
I = H(2:2:end,2:end)
subplot(1,2,1)
plot(I)
subplot(1,2,2)
surface(I)
view(-35,45)

etime(clock,t0)

%\end{verbatim}
```