Trapped!

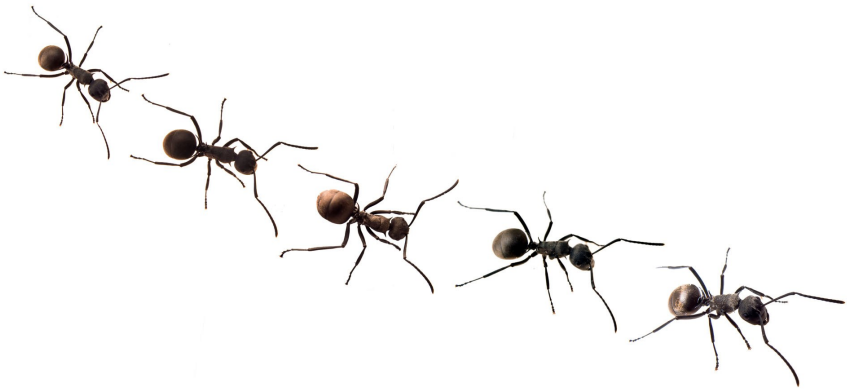
A Stochastic Optimisation of Trap Placement

Hidde Fokkema, Feline Lindeboom

July 13, 2017

Second Year Project Mathematics

Supervisors: Brendan Patch MSc & Nicolaos Starreveld MSc



Korteweg-de Vries Instituut voor Wiskunde
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Universiteit van Amsterdam

Abstract

Trap networks are a component of fundamental practical importance in pest detection, control and eradication programs. Additionally, ecologists also use such networks to identify new species. The goal of this project is to develop a method that optimises placement of traps within a habitat. More specifically we aim to choose the trap locations such that we maximise the quantity of insects caught. We construct a simple stochastic model for the placement of insects. Then, by means of simulation, trap positions are optimised. We use two different optimisation algorithms, both based on the technique of simulated annealing. In the case of a single trap our analysis suggests to place the trap in the centre of the habitat. For two traps our algorithm suggests positions on a circle around the centre are optimal.

Title: Trapped!
Subtitle: A Stochastic Optimisation of Trap Placement
Authors:
Hidde Fokkema, hidde.fokkema@gmail.com, 11010673
Feline Lindeboom, f.l.lindeboom@gmail.com, 10791434
Supervisors: Brendan Patch MSc & Nicolaos Starreveld MSc
Date: July 13, 2017

Korteweg-de Vries Instituut voor Wiskunde
Universiteit van Amsterdam
Science Park 904, 1098 XH Amsterdam
`http://www.science.uva.nl/math`

Contents

Introduction

Trap networks are a component of fundamental practical importance in pest detection, control, delimitation and eradication programs. In addition, insect trapping is a popular method to encounter new species, a topic of great ecological importance. In this project we therefore seek to find optimal trap placement in order to enhance insect trap captures.

Trap placement has been studied before in ecological literature. Branco, Jactel, Franco & Mendel (2006) for example, focused on modelling the response of the scale insect male to increasing doses of its female's sex pheromone, and that of the adult predator to increasing doses of its prey's sex pheromone. Several experiments were conducted to test assumptions on the links between pheromone dosage and trap catches. The results were consistent with the assumption that the relationship between probability of capture and distance from the pheromone attractants fitted a logistic model. Moreover, for a given distance the probability that the insect is attracted by the lure increases with dosage. Manoukis, Hall & Scott (2014) came to a similar, but more general result. A model for trap networks with a variable attractiveness was developed. Using the model, Manoukis et al. found that the relationship between number of traps in a network and capture probability is highly dependent on the attractiveness of the traps. Anguelov, Dufourd & Dumont (2015) developed a trap-insect model which consisted of a chemotaxis model simulating the spread of a chemical attractant released by the traps and an insect spreading model in which the insects' response to the attractant was modelled.

Our focus lies on the development of a novel algorithmic approach to optimal trap placement. To the authors knowledge this has not been studied before. We start by constructing a simple stochastic model for the position of insects. Due to the simplicity of our model it is widely applicable and straight forward to implement. Then, by means of simulation, a finite amount of traps is placed and their location is optimised. This optimisation is performed using two different algorithms. The first algorithm is based on a technique called simulated annealing. The second is a probabilistic adaption of this method.

This project consists of two parts that are independent from one another. Part I concerns the work we did in the second half of the semester. In Chapter 1 and 2 we formally state the problem we are interested in, briefly discussed above. We construct a model for particle placement on the plane and present the algorithms we developed. For the case of one trap we also find the optimal position analytically. Our analytical results are presented in Chapter 3. An implementation of the algorithms and a discussion follow in Chapter 4 and 5. In Part II we discuss stochastic models in general, with an emphasis on queueing models and discrete event simulation. We studied these topics during the first half of the semester in order to develop the skills necessary to address the issues presented in Part I. In Chapter 6 we present some general theory and in Chapter 7

we elaborate more on queueing models. In both Chapters we present some simulation results of various processes. Even though this part does not contribute to solving the problem of trap placement, this part does include various interesting results, so we hope you will keep on reading once the insects are trapped.

Part I.

Ecological Problem of Trap Placement

1. Model

1.1. Potential model

Inspired from a problem in ecology we will try to construct and simulate a probabilistic model describing the movement of insects in a region of the plane: a subset $\mathcal{S} \subset \mathbb{R}^2$. In what follows we will treat insects as particles moving randomly in space and they may interact or not interact with each other. We start with presenting some ideas regarding a model for particle behaviour. We describe the parameters that might be needed in such a model and propose some methods to include them.

1.1.1. How many particles enter $\mathcal{S}$

To model the amount of particles that enter S we use a Poisson process $\{N(|\,\mathcal{S}\,|) : |\,\mathcal{S}\,| \geq 0\}$. The Poisson process will have an intensity λ which represents the rate at which particles arrive. We consider the case where λ is constant but various generalisations are possible. We expect the amount of particles entering $\mathcal{S}$ to depend on the amount of particles already inside $\mathcal{S}$. In this case the parameter λ can be made state dependent. The availability of resources in $\mathcal{S}$, food and water for example, or the time t may also influence the arrival rate of particles. Such cases can be captured in the parameter λ by making it depend on $\mathcal{S}$ or on time.

1.1.2. How particles enter $\mathcal{S}$

Consider the case where $\mathcal{S} = [0,1]^2 \subset \mathbb{R}^2$. This might not be the most realistic case but it simplifies analytic computations. Our algorithms can be extended for the case of a general region $\mathcal{S} \subset \mathbb{R}^2$.

In order to model particle movement we need to clarify how particles enter $\mathcal{S}$. The simplest case to consider is that an arriving particle chooses a point on the boundary of the unit square uniformly at random and then starts moving from that point. This could be in $\mathcal{S}^{\circ}$, on the boundary $\partial\,\mathcal{S}$ or on $\mathbb{R}^2 \setminus \mathcal{S}$.

There are various subtleties that need to be arranged at this point concerning movement on the boundary. It is natural to let particles exit $\mathcal{S}$ but there might be cases where they cannot exit, because of obstacles for example. When particles have left, there is also a probability that they return quickly after this, which will increase the probability a particle arrives. Of course not all insects are the same. For example, ants are social animals and follow traces of their companions, while many other insects are more solitary. This also impacts the potential modelling of particle movement. Such questions are out of the scope of this project and we do not treat them in detail.

1.1.3. The angle of a step

Consider the particles that have arrived and are located in $\mathcal{S}$. Particle movement can be seen as a continuous process, such as a Brownian motion, which is rather complicated to formulate and analyse. Another option is to consider discrete time. At each time step every particle will take a step of some length and with some angle independently of all other particles. We present two methods to choose the angle φ at which each step will be performed.

1. The angle φ is continuously and symmetrically distributed over an interval $[-\pi, \pi)$. This could be a normal distribution with mean 0 and yet unknown standard deviation. In all cases however, we consider a truncated distribution on the interval $[-\pi, \pi)$. We do this because otherwise all angles modulo 2π will lead to the same result.

Modelling the angle in this way, there is a tendency to move straight forward, since the distribution has most weight symmetrically around $\varphi = 0$. Here choice of φ at step i is unrelated to φ at step $i - 1$. In much animal motion, however, direction is persistent. Insects will change their direction gradually from step to step (Kareiva & Shigesada, 1983). This is commonly modelled with a correlated random walk. We can change the way we model the φ according to this.

2. Now we model φ considering a distribution on the increments. If the angle at time $t = 0$ is φ, the angle at $t = 1$ becomes $\varphi + \delta \cdot \varphi$, where δ has some truncated distribution. This way the particle will not walk in a line straight forward as much as in 1, but a direction is chosen based upon previous decisions.

1.1.4. The length of a step

As above we consider discrete time and at each time instance a particle takes a step. The length of a step depends on how fast the insect walks. Let L denote the random variable giving the length of each step. Then L only takes values $l \geq 0$. Also, the values are obviously bounded from above by some constant.

1.1.5. When two particles meet

What particles do when they meet depends on the situation. For instance, particles of the same kind might want to exchange food, or two males may fight over a female. particles of other species might fight or avoid each other. These are all very interesting cases corresponding to real life scenarios, but we avoid an extensive discussion on these topics.

1.2. Simplifications

As discussed above there are many different parameters that can be included in a potential model and there are various different models that can be considered. In this project

we make particle movement as simple as possible and focus on optimal trap placement. Essentially we propose a simple model of particle placement.

First of all the area $\mathcal{S}$ we will consider is the unit square, i.e. $S = [0, 1]^2$. As mentioned above this simplifies analytical computation but does not affect the performance of the algorithms we will develop. Traps in $\mathcal{S}$ are modelled as points representing the centre of a trap. They do not have any attractive properties. We also assume that particles are located in $\mathcal{S}$ according to a Poisson point process $N(|\,\mathcal{S}\,|)$. The process is homogeneous. This means the parameter λ is constant on the whole space $\mathcal{S}$, so no properties of the space and their influences on particle behaviour are taken into account. Particles that have entered $\mathcal{S}$ move around, but their movement is not simulated. In our model the particles are placed uniformly throughout the $\mathcal{S}$. Corresponding to their distance to the centre of the trap, a probability of entering the trap is considered.

2. Methodology

In this section we develop two algorithms that have as input a region in the plane, the number of traps and the parameters of the Poisson point process and outputs the optimal location of the traps. We denote the space that we are interested in by $\mathcal{S} = [0,1]^2 \subset \mathbb{R}^2$. Then, more specifically our algorithm takes as input the function $f : \mathcal{S} \times \mathcal{S} \to [0,1]$ where $f(x,y)$ gives us the probability that a particle at location x will be caught in a trap at location y. We assume particles are located in $\mathcal{S}$ according to the Poisson point process $\mathcal{B}$. That is, for $E_1, \ldots, E_k \in \sigma(\mathcal{S})$ disjoint subsets of the Borel σ-algebra of $\mathcal{S}$, the number of particles found in $E_1, \ldots, E_k$ are independent Poisson random variables with means $\mathcal{B}(E_1), \ldots, \mathcal{B}(E_k)$. A simple example is when $\mathcal{B}(E_i) = |E_i|\lambda$ for all $E_i \in \sigma(\mathcal{S})$. Furthermore, we assume that each of n traps may be placed uniquely at a point in $\mathcal{S}$. Let M_j be a random variable denoting the number of particles caught in trap j and let $\vec{c_i}$ be the position of a trap. We seek to solve the following optimisation problem.

$$\begin{aligned} &\text{Maximise} \quad \mathbb{E}\sum_{j=1}^{n} M_j, \\ &\text{subject to} \quad \vec{c_i} \in \mathcal{S}, \quad \vec{c_i} \neq \vec{c_j}, \quad \forall\, i,j \in \{1,\ldots,n\}. \end{aligned} \tag{2.1}$$

In this chapter we develop two algorithms we used to solve the optimisation problem discussed in (2.1).

2.1. Simulated annealing

The first algorithm makes use of a technique called *simulated annealing*. We briefly explain this method.

To find an optimal solution to our problem we could consider a point in the unit square at random and estimate the expected amount of particles caught by a trap. Afterwards we could consider another point at random in the neighbourhood of the previous one and check whether the expectation in (2.1) increases. This method is also called the *hill climb algorithm* (Skiena, 2008). However, this algorithm runs the risk of getting stuck in a local maximum. There are many ways to overcome this obstacle. We use a technique called simulated annealing to circumvent this issue.

The technique of simulated annealing is based on the physical effect of cooling down hot metals. The basic idea is similar to that of the hill climbing algorithm. The difference is that, while the hill climbing algorithm only allows us to move to points with an improved result, using simulated annealing there is a probability of going to points that do not

improve the performance value function. The rationale is that this should move us away from local maxima. During the simulation one slowly decreases the probability of moving to worse points. This way we slowly converge to the global maximum.

To state it formally, we first define a quantity temperature $T > 0$. Typically $T = 1$ Let $f : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ be a function that assumes a global maximum on the unit square $[0, 1]^2$, then simulated annealing follows the following algorithm.

Algorithm 2.1.1. `Initialise: Set` $T_0 = 1$, $\epsilon > 0$, `choose` $\vec{z}_0 \sim U([0,1]^2)$ `and set` $A_0 = f(\vec{z}_0)$

`(i) Let` $\vec{z} = \vec{z}_{k-1}$ `and` $A = A_{k-1}$

`(ii) Let` $\tilde{\vec{z}} \sim U([0,1]^2)$ `and calculate` $\tilde{A} = f(\tilde{\vec{z}})$.

`(iii) Let` $\alpha = \min\{1, \mathrm{e}^{-(A_{old} - A_{new})/T}\}$.

`(iv)`

$$(\vec{z}_k, A_k, T_k) = \begin{cases} (\tilde{\vec{z}}, \tilde{A}, 0.9T_{k-1}) & \texttt{with probability} \quad \alpha \\ (\vec{z}, A, 0.9T_{k-1}) & \texttt{with probability} \quad 1 - \alpha. \end{cases}$$

`(v) Stop when` $T < \epsilon$, `else return to (i).`

The first step of algorithm 2.1.1 makes sure that the algorithm will follow an iteration scheme. Then in the second step the algorithm samples a new position and estimates what the expected value of captured particles will be. The difference with the hill climbing algorithm expresses itself in steps (iii) & (iv). The parameter α determines the probability the algorithm moves to the new position. We see $\alpha = 1$ when we catch more particles in the new position. If the new position catches fewer particles, $0 < \alpha < 1$, so we might move to this new position anyway. This probability will be small if the new performance is considerably worse. This will ensure that the positions do not move away from candidate positions too much. Finally, by multiplying T_k with 0.9 each step we create a fixed schooling scheme that ensures that the algorithm stops eventually at a position that is close to the global maximum. We generalised this approach to multiple traps by changing the position of only one trap per iteration. In the case of algorithm 2.1.1 the traps were moved in a cycle. So first we potentially move trap 1, then trap 2, then trap 3 and so on.

2.1.1. Simulation

The code we wrote in order to implement this algorithm follows the process as described in the previous section. In our case $\vec{z}_k$ is a point in the plane describing the position of the trap. The function $f(\vec{z})$ describes how many particles enter the trap on average. A small change was made to algorithm 2.1.1 above. Now we do not change the temperature at every step, but let algorithm 2.1.1 run for 100 iterations before changing the temperature. The threshold was $\epsilon = 0.000001$.

Of course this process has to be adapted if we want to move to a model with more than 1 trap in the plane. To do this we implemented a modified version of simulated annealing. The principle is still the same and the temperature is still adjusted in the same manner. The difference is that the new algorithm is initiated not with one point, but with as many as there are traps in the plane. Thus, $\vec{z_i}$ becomes a list of points and $f(\vec{z})$ calculates the total number of particles that enter the traps individually. At each step of this process we will look at the trap that is performing the weakest and look for a better position. This way only one trap is moved each time.

2.2. Probabilistic adaptation

The second algorithm that we developed has two major differences. Firstly, the parameter of the Poisson process may be updated in each step. This change will depend on a probabilistic function g. The second change is that the 'cooling scheme' will no longer be fixed, but will also change according to a probabilistic function h.

Let $\mathcal{B}_\gamma$ be a Poisson process where the number of particles found in $E_1, \dots E_k$ are independent Poisson random variables with means $\gamma\mathcal{B}(E_1), \dots, \gamma\mathcal{B}(E_k)$. Let $g : \mathbb{R} \to [0, 1]$ be a function such that $\lim_{x\to\mathcal{B}(\mathcal{S})} g(x) = 1$ and $g(x_{i+1}) = g(x_i)$ if $x_{i+1} \leq x_i$. Here i tells us from which iteration x was taken. For example

$$g(x) = \max\{-\frac{1}{x_i + 1} + 1, -\frac{1}{x_{i+1} + 1} + 1\}. \tag{2.2}$$

Let $h : \mathbb{R} \to \mathbb{R}$ be a function such that $h(x) \to \infty$ as $x \to \mathcal{B}(\mathcal{S})$ and $h(x) \geq 0$ for all x. For example

$$h(x) = \begin{cases} \alpha/(x - \mathcal{B}(\mathcal{S})) & x > \mathcal{B}(\mathcal{S}) \\ -\alpha/(x - \mathcal{B}(\mathcal{S})) & x \leq \mathcal{B}(\mathcal{S}). \end{cases} \tag{2.3}$$

Our algorithm is as follows.

Algorithm 2.2.1. `Initialise: Set` $k = 1$, $\Gamma_0 = \gamma_0$, $Z_0 = 0$, `choose` $\mathcal{L}_0$ `from` $\mathcal{F}$.

(i) `Let` $\gamma = \Gamma_{k-1}$ `and sample particle locations` $\mathbf{X}$ `according to` $\mathcal{B}_\gamma$.

(ii) `Choose` ℓ `as the trap which previously caught the least particles and then sample` y_ℓ `from unallocated locations (including current location of trap` ℓ`). Let this allocation be` y.

(iii) `Determine number of particles caught` z'.

(iv) `Set` $z = Z_{k-1}$, `then let`

$$(\mathcal{L}_k, \Gamma_k, Z_k) = \begin{cases} (y,\, g(z'),\, z') & \text{with probability} \quad \min\{1, \mathrm{e}^{-h(z')\cdot(z-z')}\}, \\ (\mathcal{L}_{k-1},\, \Gamma_{k-1},\, Z_{k-1}) & \text{otherwise.} \end{cases} \tag{2.4}$$

(v) If *stopping condition* is met, then stop, else set $k = k+1$ and return to (i).

The function g replaces the constant parameter λ in algorithm 2.1.1 and the function h replaces the cooling scheme of T. The reason for the function g is that we no longer estimate the mean number of particles captured . The rationale behind g is that it will increase whenever a set of traps captures many particles. Then, if we catch fewer particles in the next iteration, we know that the previous position captured a lot of particles purely by chance, not because of its position. The function h will tend towards infinity whenever the amount of particles captured is close to the expected value of particles existing in $\mathcal{S}$. The reason for this is that this way we stay at our position if the traps capture almost all particles present in $\mathcal{S}$ in total. In our case we implement algorithm 2.2.1 a finite amount of iterations. A word of caution for using algorithm 2.2.1 has to be given, as it is still a work in progress and a proof of convergence to a maximum is not developed yet.

2.3. Comparing the two algorithms

There are some significant advantages of algorithm 2.2.1 compared to algorithm 2.1.1. The first one is the difference in computing capacity. In algorithm 2.1.1 at every iteration the expected value of captured insects had to be estimated. In our case this means sampling the captured particles a 1000 times and then averaging. In algorithm 2.2.1 we sample only once per iteration. So if algorithm 2.2.1 is proven to converge to a maximum, then it would use significantly less computing power.

Moreover, we have the intuition that it would be simpler to prove convergence for algorithm 2.2.1. If we let the number of iterations tend to infinity, then we want to make predictions about the behaviour of our algorithms. Working with this probabilistic approach it is possible to say that certain things will happen with a non-zero probability. In other words that they must happen at some point, when the number of iterations approach infinity. This was another reason to develop algorithm 2.2.1.

So far we have used both algorithms to optimise the position of one trap. The optimisation of the position of more than one trap was done in the case of two traps. We only used algorithm 2.1.1 for this. Algorithm 2.2.1 was shown not to be reliable enough yet to be used for multiple traps.

3. Explicit Solutions

In this chapter we approach the issues presented in the beginning of Chapter 2 analytically. We treat the optimisation problem (2.1) for a single trap and for multiple traps.

3.1. Optimisation of the position of one trap

For any position of the trap, we want to compute the expected value of the amount of particles that enter the trap. Afterwards we optimise the position of the trap.

We start with a simpler case than the one presented in the beginning of Chapter 2. Let $N(|\,\mathcal{S}\,|)$ be a Poisson Process with rate λ, representing the amount of particles in the area $\mathcal{S} = [0,1]^2 \subset \mathbb{R}^2$. Let A_i, $i = 1, \ldots, N(|\,\mathcal{S}\,|)$ be a collection of random variables indicating whether the i'th particle will enter trap $B \subset \mathcal{S}$ with fixed position. Then

$$A_i = \begin{cases} 1 \text{ with probability } p(X_i, Y_i) \\ 0 \text{ with probability } 1 - p(X_i, Y_i) \end{cases} . \tag{3.1}$$

Here $X_i \sim U(0,1)$ are mutually independent random variables that indicate the x-position of the particle within $\mathcal{S}$, $Y_i \sim U(0,1)$ are mutually independent random variables that indicate the y-position. All A_i are independent. Define $A(|\,\mathcal{S}\,|)$ as the total number of particles that is caught in the trap. Then $A(|\,\mathcal{S}\,|)$ can be written as

$$A(|\,\mathcal{S}\,|) = \sum_{i=1}^{N(|\,\mathcal{S}\,|)} A_i. \tag{3.2}$$

Now, more specifically, we assume the probability that a particle positioned at $\vec{x} = (x, y) \in \mathcal{S}$ will enter a trap with a centre positioned at $\vec{c} = (c_1, c_2)$ is equal to

$$p(\vec{x}, \vec{c}) = \mathrm{e}^{-\gamma \mathrm{d}(\vec{x}, \vec{c})}\,. \tag{3.3}$$

Here $\mathrm{d}(\vec{x}, \vec{c})$ is the Euclidean distance. The constant γ has to be chosen in a suitable way according to how likely it is for particles relatively far from the traps to be caught. Since

this does not change our optimisation, we set $\gamma = 1$ for the sake of simplicity. Then

$$\begin{aligned}\mathbb{E}[A_i] &= \iint_{\mathcal{S}} (1 \cdot p(x,y) + 0 \cdot (1 - p(x,y)))\,\mathrm{d}x\mathrm{d}y \\ &= \int_0^1 \int_0^1 p(x,y)\,\mathrm{d}x\mathrm{d}y \\ &= \int_0^1 \int_0^1 \mathrm{e}^{-\sqrt{(x-c_1)^2+(y-c_2)^2}}\,\mathrm{d}x\mathrm{d}y.\end{aligned} \tag{3.4}$$

Then for $A(|\,\mathcal{S}\,|)$, using properties of conditional expectation, we write

$$\begin{aligned}\mathbb{E}[A(|\,\mathcal{S}\,|)] &= \mathbb{E}[\sum_{i-1}^{N(|\,\mathcal{S}\,|)} A_i] \\ &= \sum_{k=0}^{\infty} \mathbb{E}[\sum_{i=1}^{k} A_i | N(|\,\mathcal{S}\,|) = k] \cdot \mathbb{P}(N(|\,\mathcal{S}\,|) = k) \\ &= \sum_{k=0}^{\infty} k\,\mathbb{E}[A_i] \cdot \mathbb{P}(N(|\,\mathcal{S}\,|) = k) \\ &= \mathbb{E}[N(|\,\mathcal{S}\,|)] \cdot \mathbb{E}[A_i] \\ &= \lambda|\,\mathcal{S}\,| \cdot \int_0^1 \int_0^1 \mathrm{e}^{-\sqrt{(x-c_1)^2+(y-c_2)^2}}\,\mathrm{d}x\mathrm{d}y.\end{aligned} \tag{3.5}$$

Here we use corollary 6.4.6 for the last equality.

We want to determine the point $\vec{c}$ which maximises the expectation given in (3.5). We then find

$$\begin{aligned}\underset{(c_1,c_2)\in[0,1]^2}{\operatorname{argmax}} \mathbb{E}[A(|\,\mathcal{S}\,|)] &= \underset{(c_1,c_2)\in[0,1]^2}{\operatorname{argmax}} \lambda|\,\mathcal{S}\,| \cdot \int_0^1 \int_0^1 \mathrm{e}^{-\sqrt{(x-c_1)^2+(y-c_2)^2}}\,\mathrm{d}x\mathrm{d}y \\ &= \underset{(c_1,c_2)\in[0,1]^2}{\operatorname{argmax}} \int_0^1 \int_0^1 \mathrm{e}^{-(x-c_1)^2-(y-c_2)^2}\,\mathrm{d}x\mathrm{d}y \\ &= \underset{(c_1,c_2)\in[0,1]^2}{\operatorname{argmax}} \int_0^1 \mathrm{e}^{-(x-c_1)^2}\,\mathrm{d}x \cdot \int_0^1 \mathrm{e}^{-(y-c_2)^2}\,\mathrm{d}y \\ &= \underset{(c_1,c_2)\in[0,1]^2}{\operatorname{argmax}} \int_{-c_1}^{1-c_1} \mathrm{e}^{-u^2}\,\mathrm{d}u \cdot \int_{-c_2}^{1-c_2} \mathrm{e}^{-u^2}\,\mathrm{d}u.\end{aligned} \tag{3.6}$$

For the second equality we used the fact that the square root is an increasing function and for the last equality we substituted $x - c_1$ and $y - c_1$ for u. By symmetry around zero of the integrand this is maximised for $c_1 = c_2 = 1/2$. To show this explicitly, let us look at the derivatives of this function. By the fundamental theorem of calculus we

obtain

$$\begin{aligned} D_{c_1} \int_{-c_1}^{1-c_1} \mathrm{e}^{-u^2}\,\mathrm{d}\,u \cdot \int_{-c_2}^{1-c_2} \mathrm{e}^{-u^2}\,\mathrm{d}\,u &= (-\mathrm{e}^{-(1-c_1)^2} + \mathrm{e}^{-(-c_1)^2}) \int_{-c_2}^{1-c_2} \mathrm{e}^{-u^2}\,\mathrm{d}\,u. \\ D_{c_2} \int_{-c_1}^{1-c_1} \mathrm{e}^{-u^2}\,\mathrm{d}\,u \cdot \int_{-c_2}^{1-c_2} \mathrm{e}^{-u^2}\,\mathrm{d}\,u &= (-\mathrm{e}^{-(1-c_2)^2} + \mathrm{e}^{-(-c_2)^2}) \int_{-c_1}^{1-c_1} \mathrm{e}^{-u^2}\,\mathrm{d}\,u. \end{aligned} \tag{3.7}$$

Notice the integrals in these derivatives are never zero, because the integrand is a strictly positive function and the integration interval has length one. Setting this equal to zero we obtain

$$\begin{aligned} \mathrm{e}^{-(1-c_1)^2} &= \mathrm{e}^{-(c_1)^2}\,; \\ (1-c_1)^2 &= c_1^2\,; \\ 1-c_1 &= c_1\,; \\ c_1 &= 1/2, \end{aligned} \tag{3.8}$$

which gives the required result. It could also have been the case that $1 - c_1 = -c_1$, but this equation is not solvable. Analogously we obtain $c_2 = 1/2$.

In the derivation of this result we used our assumption that $\mathcal{S} = [0,1]^2$. Of course this is not always the case and in general the terrain that a researcher would be looking at could be of a more complex form. The integrals that have to be maximised become messy quickly in those cases. For example, if we take a look at the case of a circle, we would like to solve the following problem.

$$\begin{aligned} &\underset{(c_1,c_2)\in C(0,1)}{\operatorname{argmax}} \lambda|\,\mathcal{S}\,| \iint_{C(0,1)} \mathrm{e}^{-\sqrt{x-c_1)^2+(y-c_2)^2}}\,\mathrm{d}x\mathrm{d}y \\ &= \underset{(c_1,c_2)\in C(0,1)}{\operatorname{argmax}} \iint_{C(0,1)} \mathrm{e}^{-(x-c_1)^2-(y-c_2)^2}\,\mathrm{d}x\mathrm{d}y \\ &= \underset{(c_1,c_2)\in C(0,1)}{\operatorname{argmax}} \int_0^{2\pi} \int_0^1 r\,\mathrm{e}^{-r^2+rc_1\cos(\theta)+rc_2\sin(\theta)-(c_1^2+c_2^2)}\,\mathrm{d}r\mathrm{d}\theta. \end{aligned} \tag{3.9}$$

The last expression in the exponent in equation (3.9) is not maximised easily. This gives us even more reason to look at solutions in terms of simulations. In that case we would use the acceptance rejection method to place particles in our area. Practically the shape of the area is not of great importance, because ecologists place their traps somewhere in the natural habitat of insects, so often there is no strictly defined boundary.

3.2. Optimisation of the position of multiple traps

The previous result was as expected. Because of our assumptions all particles are uniformly spread, thus the trap should be placed at the centre. We see the problem is also symmetric for more than one trap, so we expect the solution to have some symmetry for

this case as well.

We now consider the problem of two traps. Again we compute, for all possible positions of the traps, the expected amount of particles captured. According to this we optimise the positions of the traps. Then we once more place particles uniformly in our region $\mathcal{S} \subset \mathbb{R}^2$ according to a Poisson point process. We now want to find the positions of two traps such that the average number of particles caught is maximised. We denote the position of a particle as $\vec{x} = (x, y)$ and the positions of the traps as $\vec{c_1} = (c_{11}, c_{12}), \vec{c_2} = (c_{21}, c_{22})$. We assume that a particle at position $\vec{x}$ is caught in one of two traps at positions $\vec{c_1}$ and $\vec{c_2}$ with probability

$$p(\vec{x}, \vec{c_1}, \vec{c_2}) = \mathrm{e}^{-\gamma \mathrm{d}(\vec{x}, \vec{c_1})} + \mathrm{e}^{-\gamma \mathrm{d}(\vec{x}, \vec{c_2})} - \mathrm{e}^{-\gamma \mathrm{d}(\vec{x}, \vec{c_1})} \cdot \mathrm{e}^{-\gamma \mathrm{d}(\vec{x}, \vec{c_2})}. \tag{3.10}$$

Formula (3.10) can easily be extended in the case of n traps. We assume the trap positions are independent of one another. Again we set $\gamma = 1$. Then, analogous to our previous calculation, the expected number of particles caught in either trap is equal to

$$\mathbb{E}[A(|\,\mathcal{S}\,|)_{\vec{c_1}, \vec{c_2}}] = \frac{1}{|\,\mathcal{S}\,|} \lambda |\,\mathcal{S}\,| \int \int_{\mathcal{S}} p(\vec{x}, \vec{c_1}, \vec{c_2})\, \mathrm{d}\vec{x}. \tag{3.11}$$

Since $\mathcal{S} = [0, 1]^2$, this equals

$$\mathbb{E}[A(|\,\mathcal{S}\,|)_{\vec{c_1}, \vec{c_2}}] = \lambda |\,\mathcal{S}\,| \int \int_{\mathcal{S}} p(\vec{x}, \vec{c_1}, \vec{c_2})\, \mathrm{d}\vec{x}. \tag{3.12}$$

This integral can, unfortunately, not be computed like before. We can numerically optimise the position of two traps, but this is computationally demanding. Here the use of the previously described algorithms comes in extremely useful once again.

4. Implementation

With the algorithms described in the sections above we can search for an optimal position for the traps. We give a visualisation of these results for the case of one trap and two traps.

4.1. A single trap

Figure 4.1 shows results of algorithm 2.1.1 and figure 4.2 shows result of algorithm 2.2.1. We see both algorithms show the centre is the best location for a single trap. Using algorithm 2.2.1 however, the intensity increases slower towards the middle than using algorithm 2.1.1. We suspect this can be explained by the fact that the function h does not tend to infinity fast enough or not at all.

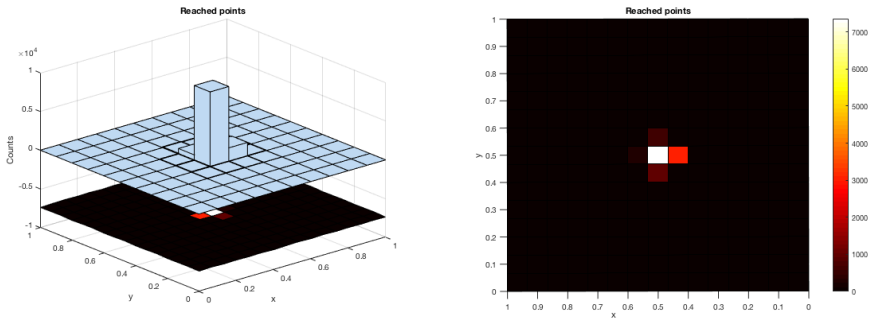


Figure 4.1.: (L): A 3D histogram plot showing where algorithm 2.1.1 allows the trap to be placed. (R): Heatmap of the positions reached by algorithm 2.1.1.

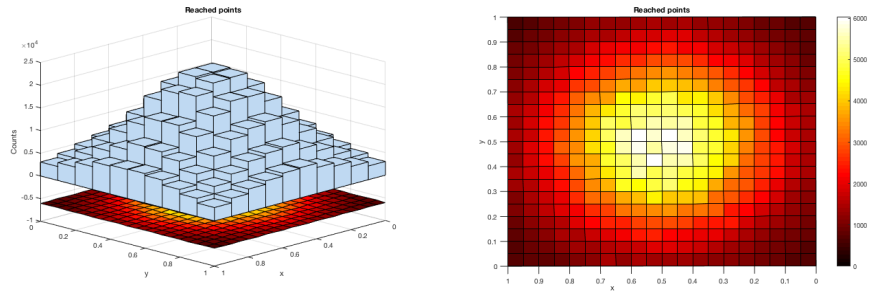


Figure 4.2.: (L): A 3D histogram plot showing where algorithm 2.2.1 allows the trap to be placed. (R): Heatmap of the positions reached by algorithm 2.2.1.

4.2. Two traps

After testing both algorithms on one trap, we moved on towards the case of 2 traps. In this case, the optimal solution is no longer unique. We found different optimal locations that captured the same amount of particles. The first thing we did to gain some insight on our results was to calculate the distance from the traps to the centre of the unit square. A histogram of these results is given in figure 4.3. This data suggested to us that the optimal solution could be lying on a circle.

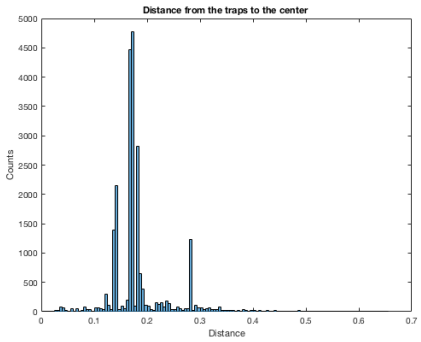


Figure 4.3.: Histogram showing distances to the centre of trap positions reached by algorithm 2.1.1.

To visualise this we ran algorithm 2.1.1 ten times and gathered all the locations the traps landed on. The reason to repeat the same algorithm multiple times instead of running it longer is that once the algorithm finds two good locations, it does not move the traps around anymore. They are 'stuck' in some sense. Since we cannot speak of a unique optimum anymore, running the algorithm multiple times gives more insight in optimal positions for two traps. The result is shown in figure 4.4. We see a circle is formed around the centre. This confirms our presumptions of symmetry pronounced before. The data in figure 4.3 indicates that the circle has a radius of around 0.18.

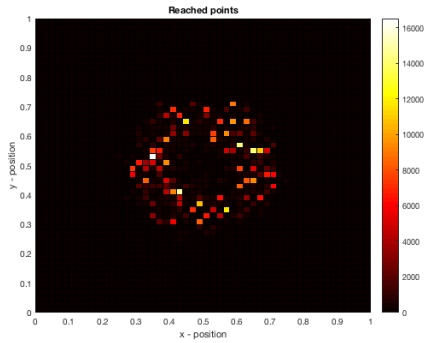


Figure 4.4.: Heatmap of the positions reached by algorithm 2.1.1 in the case of two traps. A circle can be seen centred around the middle.

5. Conclusion

The focus of this project was to develop an efficient and fast algorithm in order to optimise trap placement. Our motivation stems mostly from ecological applications. A simple stochastic model was constructed for the position of the insects within a square subset of the plane. Then, by means of simulation, a finite amount of traps was placed and the position was optimised. This was done for the case of a single trap as well as for two traps. The optimisation was performed using two different algorithms, both based on the technique of simulated annealing. In the case of a single trap, analytical and simulation results agree to place the trap at the centre of the area. For multiple traps the expression we found analytically has to be computed numerically, so our algorithms become especially useful as we may avoid performing numerical integration. For two traps our algorithm suggests a circle around the middle is optimal for placement. There are numerous ways in which the model can be improved in order to simulate a more realistic situation. Among other things this includes implementation of the movement of insects. Now insects are placed according to a Poisson point process and their probability of entering a trap is assumed only to depend on their distance to the centre of the trap. When adaptations like this are made, the model can ultimately be used in ecological research to optimise insect trapping.

Part II.

Stochastic Models

6. Preliminaries

Part 2 of this project focuses on Queueing theory. More specifically, the simulation of queue lengths of different types of queues. Before we can start on this, there are some preliminaries that are of crucial importance for the understanding of the project. They are discussed below.

6.1. Sampling from a distribution function

In order to perform a stochastic simulation, being able to take samples from a given probability distribution is essential. Below a method is presented for taking samples from a given continuous probability distribution. It is called the inverse transformation method. Before we can state this method, let us start with a definition.

Definition 6.1.1. Let X be a random variable with distribution function F. Then the **Quantile function F^{-1} of F** is given by

$$F^{-1}(\alpha) = \inf\{x \in \mathbb{R} : F(x) \geq \alpha\}, \quad \alpha \in (0, 1).$$

In words, $F^{-1}(\alpha)$ equals the smallest x with $F(x) \geq \alpha$.

When F is continuous and strictly increasing, this infimum becomes a minimum and F^{-1} is the inverse of F. This is an important assumption for the following lemma.

Lemma 6.1.2. *For any continuous and strictly increasing cumulative distribution function F, let X be the random variable*

$$X = F^{-1}(U).$$

Then X has distribution function F.

Proof. First observe that

$$\begin{aligned} F_X(a) &= \mathbb{P}(X \leq a) \\ &= \mathbb{P}(F^{-1}(U) \leq a). \end{aligned}$$

Because F is monotonically increasing, we have $F^{-1}(U) \leq a \iff U \leq F(a)$. Thus, $F_X(a) = \mathbb{P}(F^{-1}(U) \leq a) = \mathbb{P}(U \leq F(a)) = F_U(F(a)) = F(a)$. □

This result is useful since it allows us to easily generate samples from many continuous random variables of interest using only the Uniform distribution. An example using the exponential distribution follows.

Example 6.1.3. If $X \sim \text{Exp}(1)$, then $F_X(x) = 1 - \mathrm{e}^{-x}$, so $F^{-1}(u)$ is the value of x such that $1 - \mathrm{e}^x = u$ or, equivalently $x = -\log(1-u)$. Hence, if U is a uniform $(0,1)$ random variable, then $F^{-1}(U) = -\log(1-U)$ is exponentially distributed with mean 1. Since $1 - U$ is also uniformly $(0,1)$ distributed, $-\log(U)$ is exponential with mean 1. When $X \sim \text{Exp}(1)$ then $cX \sim \text{Exp}(c)$, hence $-c\log(U)$ is exponential with mean c.

Above shown is the most simple case. For some random variables it is more difficult to generate random numbers from the distribution. Consider the Normal distribution. It's Quantile function for arbitrary parameters can be derived from a transformation of the Quantile function of the Standard Normal distribution, known as the Probit function. Notation: $\Phi^{-1}(p)$, $p \in [0,1]$. Unfortunately, even though the Probit function is the inverse of the CDF of the Standard Normal distribution, this function has, like the CDF itself, no closed-form. Therefore, approximations are often used. In *Matlab* an 'erfinv' function is available.

Another way to take samples from more complex distribution functions is a type of Monte Carlo method called the *acceptance rejection method*. Here we consider another simpler density function which bounds the function we want to take samples from. Sampling from our simpler distribution, we accept our sample values with a probability proportional to the difference in height of both density functions at the sample value.

6.2. Probability generating function

Definition 6.2.1. Let X be a non-negative discrete random variable with probability mass function $p(n), n \in \mathbb{N}_0$. Then the **probability generating function** $P_X(z)$ of X is defined as

$$P_X(z) = \mathbb{E}(z^X) = \sum_{n=0}^{\infty} p(n) z^n.$$

Now by convergence of the sum for $|z| < 1$, it follows that

$$P'_X(z) = \sum_{n=1}^{\infty} p(n) \cdot n \cdot z^{n-1}.$$

Hence

$$P'_X(1) = \sum_{n=0}^{\infty} p(n) \cdot n = \mathbb{E}[X]. \tag{6.1}$$

Example 6.2.2. We give an example for $X \sim \text{Erlang}_2(\mu)$. In this case $P_L^d(z) = \frac{1-\rho}{1-\rho z - \rho^2 z(1-z)/4}$. We will show a computation for this later on. Now

$$P'_{L^d}(z) = \frac{-(1-\rho)(-\rho - \rho^2/4 + \rho^2 z/2)}{(1 - \rho z - \rho^2 z(1-z)/4)^2},$$

such that

$$P'_{L^d}(1) = \frac{-(1-\rho)(-\rho+\rho^2/4)}{(1-\rho)^2}$$

is the expected value.

6.3. Laplace-Stieltjes transform

Definition 6.3.1. The **Laplace-Stieltjes transform** $\widetilde{X}(s)$ of a non-negative random variable X with probability density function f is defined as

$$\widetilde{X}(s) = \mathbb{E}[\mathrm{e}^{-sX}] = \int_{x=0}^{\infty} \mathrm{e}^{-sx}\, f(x)\,\mathrm{d}x \quad s \geq 0.$$

An example is given below.

Example 6.3.2. When $X \sim \mathrm{Exp}(\mu)$ the Laplace-Stieltjes transform of X equals

$$\begin{aligned}
\mathbb{E}[\mathrm{e}^{-sX}] &= \int_0^{\infty} \mathrm{e}^{-sx}\, \mu\, \mathrm{e}^{-\mu x}\ \mathrm{d}x \\
&= \mu \int_0^{\infty} \mathrm{e}^{-(\mu+s)x}\ \mathrm{d}x \\
&= \mu \Big[-1/(\mu+s)\, \mathrm{e}^{-(\mu+s)x} \Big]_0^{\infty} \\
&= \frac{\mu}{\mu+s}.
\end{aligned}$$

6.4. Stochastic process

Definition 6.4.1. Let $(\Omega, \mathcal{F}, \mathcal{P})$ be a probability space, (S, Σ) a measurable space and $\mathcal{T}$ an index set. A **stochastic process** is a collection $\{X(t) : t \in \mathcal{T}\}$ of $\mathcal{S}$-valued random variables $X(t)$ indexed by $t \in \mathcal{T}$.

The index t is often interpreted as time and therefore we refer to $X(t)$ as the state of the process at time t. When $\mathcal{T}$ is a countable set, (f.e. $\mathcal{T} = \mathbb{N}$), we speak of a discrete-time process. When $\mathcal{T}$ is uncountable (f.e. $\mathcal{T} = \mathbb{R}$), the stochastic process is called a continuous-time process. The set $\mathcal{T}$ can also be multivariate. For example, $\mathcal{T} = \mathbb{R}^2$ in the problem in part one. Here we let the time t be captured in our parameter λ and care about the size of our two-dimensional space. The state space of a stochastic process is defined as the set of all values that the random variable $X(t)$ can assume.

Definition 6.4.2. A continuous-time stochastic process $\{N(t) : t \geq 0\}$ is called a **counting process** if the following holds

1. $N(t) \in \mathbb{N}_0$;
2. If $s < t$ then $N(s) \leq N(t)$.

It is called a counting process because $N(t)$ counts the number of "events" that have taken place up to time t. For $s < t$, $N(t) - N(s)$ is distributed as the number of events that have taken place during the time interval $(s, t]$.

Definition 6.4.3. The **increments** of a continuous time stochastic process are the differences $N(t) - N(s)$ for $t, s \in \mathcal{T}$.

A counting process is said to possess *independent increments* if for $s < t$, $u < v \in \mathcal{T}$, $N(t) - N(s)$ and $N(u) - N(v)$, are independent random variables whenever $[s, t] \cap [u, v] = \emptyset$. In other words the number of events that occur in disjoint time intervals are independent. Note that by induction this generalises to any finite number of increments being mutually independent when the intervals are pairwise non-intersecting.

We speak of *stationary increments* if $N(t) - N(s) \overset{\mathrm{d}}{=} t - s$. So now the number of events that occur in a given time interval is only dependent on the length of the time interval. We will see that this probability distribution partly characterises the type of stochastic process we are talking about.

We now introduce a specific type of counting process which will be of major importance during the rest of this project.

Definition 6.4.4. A counting process $\{N(t) : t \geq 0\}$ is called a **Poisson process** with rate $\lambda > 0$ if it satisfies

1. $N(0) = 0$;
2. The process has stationary and independent increments;
3. $\mathbb{P}\{N(h) = 1\} = \lambda + \mathrm{o}(h)$;
4. $\mathbb{P}\{N(h) \geq 2\} = \mathrm{o}(h)$.

Here we use the Landau-o notation: $\mathrm{o}(h)/h \to 0$ as $h \to \infty$.

The expected value of the Poisson process on $[0, t]$ with rate λ is λt. We first show this analytically. A simulation verifies the result.

Theorem 6.4.5. *Let $\{N(t) : t \geq 0\}$ be a Poisson process with rate λ. Then $N(t) \sim$ Poisson(λt).*

Proof. This is a consequence of the Poisson approximation to the Binomial distribution.

If $X \sim \mathrm{Bin}(n, p)$ let $\lambda = np$ then, for $n \in \mathbb{N}$, $p \in (0, 1)$ and $\lambda \in \mathbb{R}$ we have

$$\begin{aligned}
\mathbb{P}\{X = k\} &= \binom{n}{k} p^k (1-p)^{n-k}, \quad k = 0, 1, \ldots, n \\
&= \frac{n!}{(n-k)!k!} \Big(\frac{\lambda}{n}\Big)^k \frac{(1 - \frac{\lambda}{n})^n}{(1 - \frac{\lambda}{n})^k} \\
&= \frac{n(n-1)\cdots(n-k+1)}{n^k} \frac{\lambda^k}{k!} \frac{(1 - \frac{\lambda}{n})^n}{(1 - \frac{\lambda}{n})^k}.
\end{aligned}$$

Now we see $\lim_{n\to\infty}(1-\frac{\lambda}{n})^k = 1$, $\lim_{n\to\infty}\frac{n(n-1)\cdots(n-k+1)}{n^k} = 1$ and $\lim_{n\to\infty}(1-\frac{\lambda}{n})^n = \mathrm{e}^{-\lambda}$. The last equality follows from $(1-\frac{\lambda}{n})^n = \mathrm{e}^{n\log(1-\frac{\lambda}{n})}$ and

$$\begin{aligned}\lim_{n\to\infty} n\log\Big(1-\frac{\lambda}{n}\Big) &= \lim_{n\to\infty}\frac{(1-\frac{\lambda}{n})^{-1}\cdot\frac{\lambda}{n^2}}{-n^{-2}}\\ &= \lim_{n\to\infty}\frac{-\lambda}{1-\frac{\lambda}{n}}\\ &= -\lambda,\end{aligned}$$

where the first equality is obtained from l' Hospitals rule because $1/n$ and $\log(1-\frac{\lambda}{n})$ are both differentiable.

Now, because $\lim_{n\to\infty} n\log(1-\frac{\lambda}{n})$ exists and is finite and $\mathrm{e}^{-\lambda}$ is continuous we have $\lim_{n\to\infty}(1-\frac{\lambda}{n})^n = \lim_{n\to\infty}\mathrm{e}^{n\log(1-\frac{\lambda}{n})} = \mathrm{e}^{\lim_{n\to\infty} n\log(1-\frac{\lambda}{n})} = \mathrm{e}^{-\lambda}$.

Thus for n large and p small enough such that the desired convergence holds, we have $\mathbb{P}\{X=k\} \approx \mathrm{e}^{-\lambda}\frac{\lambda^k}{k!}$. We conclude that for $\lambda = np$ we can approximate the binomial distribution with parameters n and p by the Poisson distribution with parameter λ.

Next, we subdivide the interval $[0,t]$ into m equal parts where m is very large. Now using definition 6.4.4 we see $\mathbb{P}\{N(h)\geq 2\} = \mathrm{o}(h)$. In other words, when $h=\frac{t}{m}$, letting $m\to\infty$ and thus $h\to 0$ the probability of having two or more events in any of the subintervals approaches zero. Hence $N(t)$ will (in the limit of $m\to\infty$) equal the number of subintervals in which an event occurs. Because the Poisson process has stationary and independent increments this will be binomially distributed with parameters m and $p = \lambda\cdot\frac{t}{m}+\mathrm{o}(h)$ by (iii). Hence, by the Poisson approximation to the binomial distribution, we see $N(t)$ will be Poisson distributed with parameter $\lim_{m\to\infty} m[\lambda\frac{t}{m}+\mathrm{o}(\frac{t}{m})] = \lambda t$, because $\mathrm{o}(\frac{t}{m})\to 0$ if $m\to\infty$. Thus $N(t)\sim\text{Poisson}(\lambda t)$.

□

Corollary 6.4.6. *Let $\{N(t): t\geq 0\}$ be a Poisson process with rate λ. Then $\mathbb{E}[N(t)] = \lambda t$.*

Proof. From theorem 6.4.6 it follows that $N(t) \sim \text{Poisson}(\lambda t)$. Now

$$\begin{aligned}
\mathbb{E}[N(t)] &= \sum_{i=0}^{\infty} i \, \mathrm{e}^{-\lambda t} \frac{(\lambda t)^i}{i!} \\
&= \lambda t \sum_{i=1}^{\infty} \mathrm{e}^{-\lambda t} \frac{(\lambda t)^{i-1}}{(i-1)!} \\
&= \lambda t \cdot \mathrm{e}^{-\lambda t} \sum_{j=0}^{\infty} \frac{(\lambda t)^j}{j!} \\
&= \lambda t,
\end{aligned}$$

where we have substituted $j = i - 1$ for the third equality. □

The above shows the process has stationary increments and the number of events in any interval of length t is Poisson distributed with mean λt. This means for $s, t \geq 0$ we have

$$\mathbb{P}\{N(t+s) - N(s) = n\} = \mathrm{e}^{-\lambda t} \frac{(\lambda t)^n}{n!} \quad n \in \{0, 1, 2, \ldots\}.$$

Making use of theorem 6.4.5 the following theorem shows in a Poisson process with rate λ the inter-arrival times are exponentially distributed with mean $1/\lambda$. The proof is based on Ross (2000).

Theorem 6.4.7. *In a Poisson process with rate λ the inter-arrival times $(T_n)_{n\in\mathbb{N}}$ are independent exponential random variables with parameter λ.*

Proof. Consider a Poisson process and for $n \in \mathbb{N}$ let T_n denote the time between event $(n-1)$ and event n. Then the sequence $(T_n)_{n\in\mathbb{N}}$ is a sequence of inter-arrival times. To determine the distribution of T_n, note that $\{T_1 > t\}$ occurs if and only if no events of the Poisson process occur in the interval $[0, t]$ and hence,

$$\mathbb{P}(T_1 > t) = \mathbb{P}(N(t) = 0) = \mathrm{e}^{-\lambda t},$$

by theorem 6.4.5. Thus

$$\mathbb{P}(T_1 \leq t) = 1 - \mathbb{P}(T_1 > 1) = 1 - \mathrm{e}^{-\lambda t},$$

so $T_1 \sim \text{Exp}(\lambda)$. Now

$$\mathbb{P}(T_2 \leq t) = \mathbb{E}[\mathbb{P}(T_2 > t \mid T_1)],$$

and

$$\mathbb{P}(T_2 > t \mid T_1 = s) = \mathbb{P}(0 \text{ events in } (s, s+t] \mid T_1 = s).$$

Now by the independence of the increments this equals

$$\mathbb{P}(0 \text{ events in } (s, s+t]),$$

and by stationarity this results again in $e^{-\lambda}$. So T_2 is independent of T_1 and also exponentially distributed with parameter λ. Repeating this argument proves the theorem. □

The result is not surprising. The exponential distribution is the only continuous distribution with the *lack-of-memory property*. The inter-arrival times possess this property by stationarity and independency of the increments of the Poisson process. As promised we show a simulation of a Poisson process.

6.4.1. Simulation of a Poisson Process

We simulated the mean of the Poisson process result using two functions. The function *poisson_fixed_time* generates a sample path of a Poisson process with rate λ. This is done by counting the number of events that occurred in a time interval of length *time* according to a Poisson process with rate *lambda*. Sample times and event times are calculated. If the Next Sample Time (NST) is greater than the Next Event Time (NET) we count one event and a new NET is calculated according to the exponential distribution. We performed the simulation for *lambda* = 3 and *time* = 20. This results in a stepfunction, which can be seen on the left in figure 6.1. Next we took the sample mean of a number of sample paths to approximate the expected value. For different amount of sample paths, we plotted the difference between the output of the function *expected_value_poisson* together with the actual mean λt. This can be seen on the right in figure 6.1.

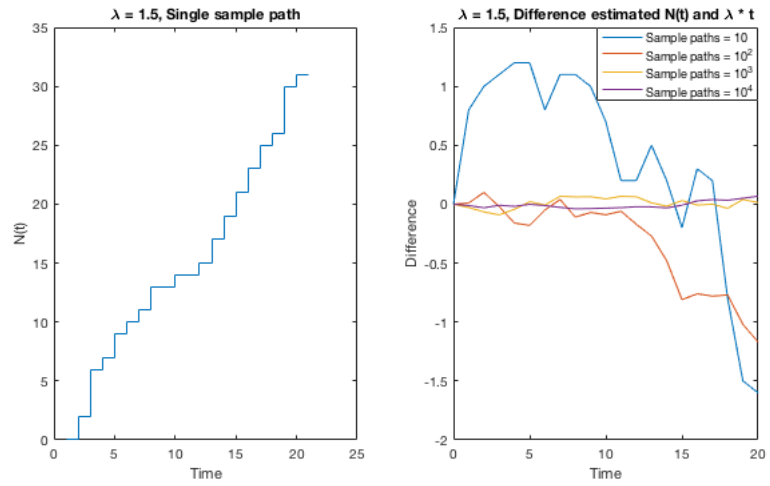


Figure 6.1.: A single sample path and the difference between the estimated and the expected value of the Poisson process.

We see an increasing number of sample paths causes less deviation from the by corollary 6.4.6 expected line 'Difference' = 0.

7. Queueing Models

Now we almost have enough knowledge to prove and simulate some basic queueing results. Before we start, prior knowledge on queueing theory comes in useful.

7.1. Theory

The discipline of Queueing Theory was developed to study the costs of investment versus the gain of a shorter waiting time in queues. Queues are often notated using *Kendall's Notation*. Hereby different types of queues are represented using three letters: A/B/c.

1. A denotes the distribution of the time between arrivals;
2. B denotes the the distribution of the time between departures;
3. c denotes the quantity of serving nodes.

Examples are the M/M/1 queue and the M/G/1 queue. The M stands for Markovian or Memorylessness. Here arrivals (and in the case of M/M/1 also departures) occur according to the exponential distribution. The G stands for any general distribution. A schematic representation of a queue is given in figure 7.1. Costumers arrive at the left, wait in line and get served in the circle, then leave at the right.

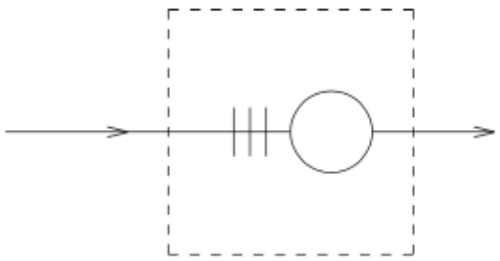


Figure 7.1.: Schematic representation of a single server queue.

We can study queues in terms of their equilibrium or limiting distribution. There is a subtle difference between these two. Consider a continuous time Markov chain $\{X(t)\colon t \geq 0\}$ with probability function $P(t)$. Then a row vector π, with the same dimension as the state space, is said to be the *equilibrium distribution* of the Markov chain

if $\pi = \pi P(t) \quad \forall t \geq 0$. In general the equilibrium distribution does not have to be the same as the behaviour of the Markov chain in the long term. However, for a continuous time Markov chain we can observe the *limiting probabilities*, given by

$$\lim_{t\to\infty} p_{ij}(t),$$

where $p_{ij}(t) = \mathbb{P}(X(t) = j \mid X(0) = i)$. The equilibrium distribution and limiting probabilities are connected by the following theorem.

Theorem 7.1.1. *In a continuous time Markov chain, if an equilibrium distribution $\pi = [\pi_1 \dots \pi_j \dots \pi_n]$ exists, then it is unique and*

$$\lim_{t\to\infty} p_{ij}(t) = \pi_j \quad \forall i.$$

A proof of this theorem can be found in (Norris, 1997). In later sections we will calculate the probability generating function of the equilibrium queue length distribution of several types of queues. This shows the equilibrium distribution exists and thus that it can be seen as the limiting distribution. The simulations in this part of the project are focused on verifying the equilibrium/limiting distributions of queue lengths of several types of queues mentioned above.

7.2. Single server queues

We start by stating a theorem that will be of great use in the rest of the project.

Theorem 7.2.1. Pollaczek-Khinchin formula
Consider an M/G/1 queue with serving time distribution $B, \tilde{B}$ the Laplace Transform of B, λ the arrival rate and $\rho = \lambda\,\mathbb{E}(B)$. Let L_k^d denote the number of costumers left behind by the k-th departing costumer. We write L^d for the limiting random variable of L_k^d. Then the probability generating function $P_{L^d}(z)$ of the equilibrium distribution of the queue length at departure moments is given by

$$P_{L^d}(z) = \frac{(1-\rho)\tilde{B}(\lambda - \lambda z)(1-z)}{\tilde{B}(\lambda - \lambda z) - z}.$$

The formula gives a relationship between the queue length and the service time distribution Laplace transform for any M/G/1 queue. Proving this result is beyond the scope of this project, but the interested reader may find satisfaction in the proof of Adan & Resing (2015). The formula is however a useful tool to derive the equilibrium distribution of the queue length of any M/G/1 queue. We start with the M/M/1 queue.

7.2.1. M/M/1 Queue

Like mentioned above, in the M/M/1 queue both inter-arrival times and serving times follow an exponential distribution and there is one server. In this case it is possible to

use the Pollaczek-Khinchin formula to calculate the limiting distribution explicitly. This distribution is Geometric(λ/μ). A calculation follows.

Consider an M/M/1 queue with $B \sim \text{Exp}(\mu)$. Then the Laplace transform $\tilde{B}(s)$ is equal to $\frac{\mu}{\mu+s}$, as seen in example 6.3.2. Hence

$$\begin{aligned} P_{L^d}(z) &= \frac{(1-\rho)\frac{\mu}{\mu+\lambda-\lambda z}(1-z)}{\frac{\mu}{\mu+\lambda-\lambda z} - z} \\ &= \frac{(1-\rho)\mu(1-z)}{(\mu-\lambda z)(1-z)} \\ &= \frac{1-\rho}{1-\rho z}, \end{aligned} \tag{7.1}$$

where we used $\rho = \lambda\,\mathbb{E}(B)$ for the last equality. For $|\rho z| < 1$ we see

$$\frac{1-\rho}{1-\rho z} = (1-\rho)\sum_{n=0}^{\infty}\rho^n z^n = \sum_{n=0}^{\infty}(1-\rho)\rho^n z^n, \tag{7.2}$$

so $p_n = (1-\rho)\rho^n$ for $n \in \mathbb{N}_0$ since $P_{L^d}(z) = \sum_{n=0}^{\infty} p_n z^n$. This is the probability mass function of a geometrically distributed random variable with parameter $\rho = \lambda/\mu$. In order to simulate this, we use the following two results.

Lemma 7.2.2. *Consider two independent exponentially distributed random variables $X \sim \text{Exp}(\lambda)$ and $Y \sim \text{Exp}(\mu)$ then $Z := \min\{X, Y\} \sim \text{Exp}(\mu+\lambda)$.*

Proof.

$$\begin{aligned} F_Z(z) &= \mathbb{P}(Z \le z) \\ &= \mathbb{P}(X \le z \text{ or } Y \le z) \\ &= 1 - \mathbb{P}(X \ge z)\,\mathbb{P}(Y \ge z) \\ &= 1 - (1-F_X(z))(1-F_Y(z)) \\ &= 1 - \mathrm{e}^{-(\lambda+\mu)\cdot z}. \end{aligned}$$

This is the CDF of an Exponentially distributed random variable with parameter $\lambda + \mu$. □

Lemma 7.2.3. *Let $X \sim \text{Exp}(\lambda)$ and $Y \sim \text{Exp}(\mu)$ be independent random variables. Then $\mathbb{P}(X = \min(X,Y)) = \frac{\lambda}{\lambda+\mu}$.*

Proof. $\mathbb{P}(X = \min(X,Y)) = \mathbb{P}(Y > X)$. We calculate this probability using the joint probability density function. By the independence of X and Y we have $f_{X,Y}(x,y) = f_X(x)\cdot f_Y(y) = \lambda\,\mathrm{e}^{-\lambda x}\cdot\mu\,\mathrm{e}^{-\mu y}$ for $x, y > 0$. So

$$\begin{aligned}\mathbb{P}(Y > X) &= \int_0^\infty \int_0^y \lambda \, \mathrm{e}^{-\lambda x} \cdot \mu \, \mathrm{e}^{-\mu y} \, \mathrm{d}x \mathrm{d}y \\ &= \int_0^\infty \mu \, \mathrm{e}^{-\mu y} \cdot (1 - \mathrm{e}^{-\lambda y}) \, \mathrm{d}y \\ &= \frac{\lambda}{\mu + \lambda}.\end{aligned}$$

□

Simulation

We have simulated the M/M/1 queue process and estimated the mean queue length and its distribution with the data obtained. This can be done with two different methods. *Method 1* uses the properties of the exponential distribution shown in lemma 7.2.2 and lemma 7.2.3, *method 2* does not. The density of the equilibrium queue length at fixed time t was also simulated using *method 1*.

Simulating the M/M/1 queue is only slightly different from simulating the number of events that have occurred in the Poisson counting process shown before. Now for each event we specify whether it is an arrival or a departure and this determines whether the queue length increases or decreases with 1. We again compute a vector of queue lengths at different times. We do this multiple times to get a mean queue length.

For the M/M/1 queue we may now use lemma 7.2.2 to determine the next event time and lemma 7.2.3 to decide whether this is an arrival or departure. We call this *method 1*. Using this method we generated 10^5 sample paths on the interval $[0, 20]$ with $\lambda = 1$ and $\mu = 3$.

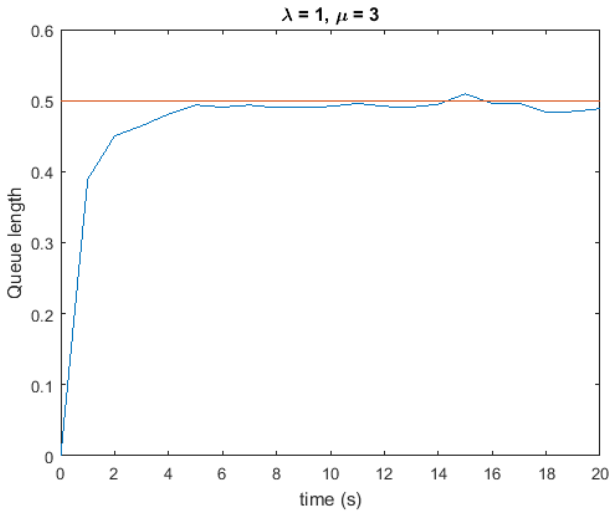


Figure 7.2.: The mean length of an M/M/1 queue.

For $X \sim \text{Geometric}(\rho)$ we have $\mathbb{E}[X] = \frac{\rho}{1-\rho}$ so for $\lambda = 1, \mu = 3$, the mean queue length is 0.5. We see the mean queue length increases exponentially to 0.5, so we are glad to see the simulation confirms the theory.

The same graph can be obtained using *method 2*. Now we simulate the M/M/1 queue process and estimate the mean queue length without the help of the lemma 7.2.2 and lemma 7.2.3. This is done by first calculating a new arrival time with rate λ. If this time is reached, the simulation checks if the queue is empty or not. If the queue is empty a departure for the arriving costumer is scheduled immediately. If the queue is not empty, the costumer is added to the queue. At every time step the simulation also checks if a departure time has been reached to decrease the queue length with one.

To simulate the probability mass function of the equilibrium queue length we assumed that the queue was simulated after sufficiently large time and that the probability of the queue in equilibrium state having length k is given by

$$\mathbb{P}(M = k) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\{N_i(t) = k\}, \tag{7.3}$$

where t is sufficiently large such that M is the queue length in equilibrium state. Here N is the sample size. We consider a sample of size $N = 1000$ and a time horizon equal to $t = 2000$ time units. Figure 7.3 shows the equilibrium distribution of the M/M/1 queue with $\rho = 1/3$ and a plot of Geometric(1/3). Here the dots are connected by lines. We see there is a difference but the lines follow a similar curve. The graphs may differ because t and N are large, but not nearly infinity.

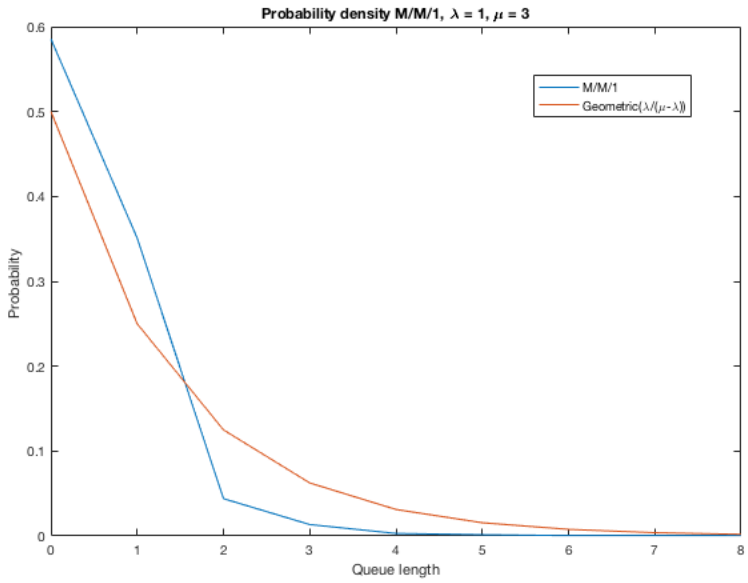


Figure 7.3.: Probability mass function of the M/M/1 queue length in equilibrium and the geometric distribution with mean $\frac{\lambda}{\mu-\lambda}$.

Discussion on simulation methods

Both methods discussed above can be used to simulate the mean queue length of the M/M/1 queue. Both have their (dis)advantages. *Method 1* is faster. Here we compute a next event time and then decide whether it is an arrival (with probability $\frac{\lambda}{\lambda+\mu}$) or a departure. For the second method we compute an arrival time and a departure time due to their individual distributions and then compare which one occurs first. this is slower because it has to sort a list every time a new costumer enters the queue. It is know that this takes $\mathcal{O}(n\log(n))$ computation time. The first method, however, cannot be generalised to other serving distributions, as the properties of the exponential distributions are explicitly used. We want to be able to compute equilibrium queue lengths of queues with other serving times distributions, as in reality serving times are definitely not always exponentially distributed. When we are interested in the equilibrium queue length of any other M/G/1 queue, we must use *method 2*.

7.2.2. M/G/1 Queues

The M/M/1 queue is a specific type of M/G/1 queue. For other M/G/1 queues it can be difficult to derive the probability density or -mass function of the equilibrium distribution of the queue length, like we have done previously. We can, however, compute the mean of the equilibrium distribution, since $P'_{L^d}(1) = \mathbb{E}[L^d]$, as shown in (6.1), with L^d as before. As an example, we consider serving time distributions $B \sim \text{Erlang}_2(\mu)$. Then $\widetilde{B}(s) = (\frac{\mu}{\mu+s})^2$. Now we use the Pollaczek-Khinchin formula again to derive the

probability generating function.

$$\begin{aligned} P_{L^d}(z) &= \frac{(1-\rho)(\frac{\mu}{\mu+\lambda-\lambda z})^2(1-z)}{(\frac{\mu}{\mu+\lambda-\lambda z})^2 - z} \\ &= \frac{(1-\rho)\mu^2(1-z)}{\mu^2 - z(\mu+\lambda-\lambda z)^2} \\ &= \frac{(1-\rho)(1-z)}{1-z(1+\rho(1-z)/2)^2} \\ &= \frac{1-\rho}{1-\rho z - \rho^2 z(1-z)/4}. \end{aligned} \tag{7.4}$$

Here we used $\rho = \lambda \cdot \mathbb{E}[B] = \frac{2\lambda}{\mu}$ for the third equality. Now from example 6.2.2 we see $\mathbb{E}[L^d] = P'_{L^d}(1) = \frac{-(1-\rho)(-\rho+\rho^2/4)}{(1-\rho)^2}$ so now since $\rho = 2/3$ we obtain an expected value of $5/3$.

Simulation

Simulating the M/G/1 process and estimating the density and mean of the equilibrium distribution of queue length can be done by adapting the *method 2* used for the M/M/1 queue. The simulations of especially the density are interesting since, like mentioned before, analytically deriving the equilibrium queue length distribution can be very challenging. Figure 7.4 shows the mean queue length for Erlang_2 distributed serving times. Other parameters are left the same.

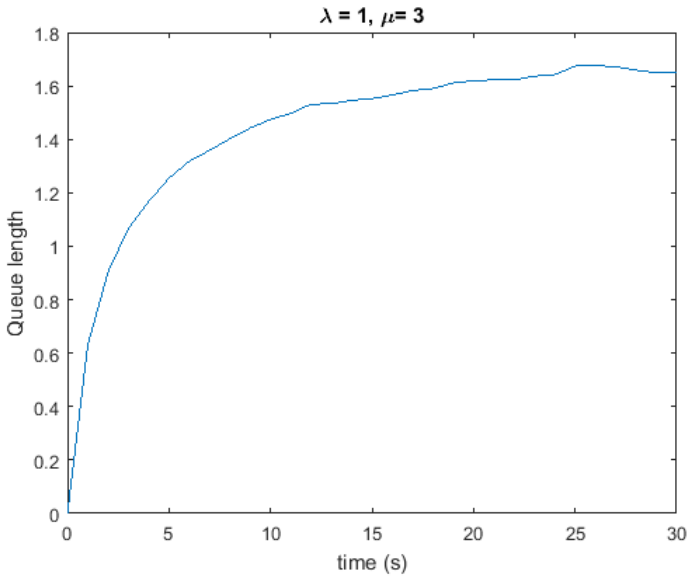


Figure 7.4.: Mean queue length of the $\text{M/E}_2\text{/1}$ queue.

We see the mean queue length stabilizes at a higher number of costumers. It seems that the queue length stabilizes around 5/3, so this confirms the theory.

7.3. Insensitive queueing systems

We have seen the equilibrium distributions of the queue lengths for the M/M/1 and M/G/1 queues. Here there is only one server, so costumers may have to wait before being served. Now we take a look at queues where there is no waiting time and theoretically infinitely many people can be served at the same time. They are called M/G/∞ queues. In general the expected equilibrium queue length depends on the service time distribution. In M/G/∞ queues, however, it only depends on the mean service time. This property is called insensitivity. The equilibrium distribution of all M/G/∞ queues is Poisson(ρ), where $\rho = \lambda \cdot \mathbb{E}[B]$ (Adan & Resing, 2015). We will not prove this. A simulation of the probability mass function and mean follow.

Simulation

Both simulation methods can be adjusted to simulate the mean queue length of an insensitive queueing system.

The mean queue length of the M/M/∞ queue is shown for $\lambda = 1, \mu = 3$. Here we have $B \sim \text{Exp}(\mu)$. In figure 7.5 we indeed see the M/M/∞ queue length stabilises around $1/3 = \lambda/\mu = \lambda\,\mathbb{E}(B) = \rho$, which is the expected value of a Poisson(ρ) distributed random variable. We used *method 1* for this simulation.

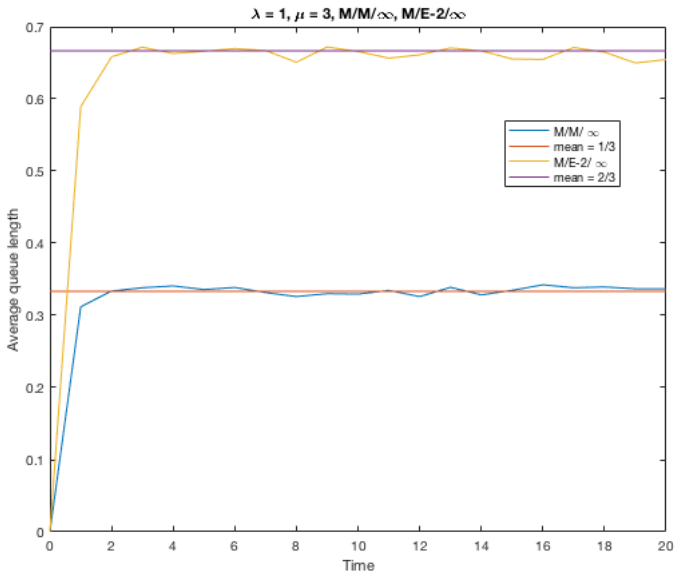


Figure 7.5.: Mean queue length of the M/M/∞ and M/E$_2$/∞ queues.

Using *method 2* the mean queue length of the $M/E_2/\infty$ queue is shown for the same parameter values, also in figure 7.5. Now $B \sim \text{Erlang}_2(\mu)$, so $\rho = \lambda \mathbb{E}(B) = 2\lambda/\mu = 2/3$ and this result is also verified by the simulation.

Lastly in figure 7.6, a connected plot of the probability mass function of the equilibrium distribution of the $M/M/\infty$ queue for $\lambda = 1, \mu = 3$ is shown together with a connected graph of the Poisson(1/3) distribution.

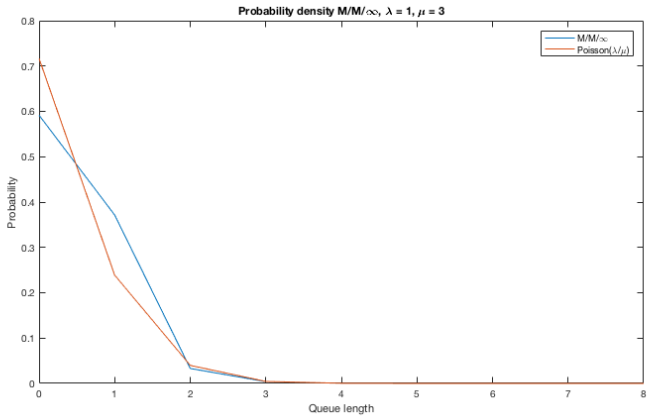


Figure 7.6.: Probability mass function of the $M/M/\infty$ queue length in equilibrium and the Poisson distribution with mean $\frac{\lambda}{\mu}$. Number of sample paths was 1000 and $t = 2000$.

Bibliography

Adan, I. & Resing, J. (2015). *Queueing systems.* Eindhoven, The Netherlands: Department of Mathematics and Computing Science, Eindhoven University of Technology.

Anguelov, R., Dufourd, C. & Dumont, Y. (2015). Simulations and parameter estimation of a trap-insect model using a finite element approach. *Mathematics and Computers in Simulation, 133*, 47-75.

Branco, M., Jactel, H., Franco, J. & Mendel, Z. (2006). Modelling response of insect trap captures to pheromone dose. *Ecological Modelling, 197* (1-2), 247-257.

Manoukis, N.C., Hall, B. & Scott, M. G. (2014). A computer Model of Insect Traps in a Landscape. *Scientific Reports, 7015* (4), 1-8.

Norris, J. R. (1997). *Markov chains.* Cambridge, UK: Press syndicate of the university of Cambridge

Ross, S. M. (2000). *Introduction to Probability Models.* Berkeley, California: Harcourt Academic Press.

Kareiva, P. M. & Shigesada, N. (1983). Analyzing Insect Movement as a Correlated Random Walk. *Oecologia, 56,* 234-238.

Skiena, S.S. (2008). *The Algorithm Design Manual.* New York, USA: Department of Computer Science, State University of New York.

A. Popular Summary

There are many real life situations in which we try to catch moving insects. For example, in pest detection, control, delimitation and eradication programs. We could also be motivated from ecology, where researchers seek to encounter new species of insects. In these situations we are keen to know what is the best location for our traps, such that our catch of insects is maximised.

In order to approach this problem properly, we first need to model the situation. We start with a flat area we call our field. In this area we place a certain amount of insects uniformly throughout the field. Next we wonder where the best spot for our trap would be, if we only possess one trap. Let us place the trap somewhere in the field at random. Then we ask ourselves: which of the insects existing in the field, will enter the trap? In our model, the insects that were placed in the field do not move. We do, however, anticipate upon their movement. So to decide upon which of the insects will enter the trap, we give every insect a certain probability of entering the trap. This probability depends on the distance of the insect to the centre of the trap. We give insects we placed further away from the trap a smaller probability of entering. With this probability we assign the number 1 to the insect and otherwise 0. Now we decide insects that were assigned a 1 will enter the trap. Insects that were assigned a 0 do not enter. The sum of all these (zeros and) ones equals the total number of insects caught by our trap at the given position. Our goal is now to maximise the average total number of particles caught by our trap, with respect to the location of the trap. In this project we focused on doing this using both explicit computation as simulation.

Our first attempt is to solve this using calculus. We can compute the average amount of insects that will enter a trap at an unknown location by means of integration. Maximising the integral with respect to the variable location of the trap gives us an optimal location right in the middle of our field.

This result however, is seemingly trivial. We are more curious for the optimal position of two traps in the field. Unfortunately we cannot optimise this analytically like we have done before, because the integral will contain a cross-term which makes the necessary substitution impossible. This makes it extremely desirable to be able to simulate this whole situation such that hopefully we can learn where to place our traps when we have more than one.

We used two different algorithms to simulate the optimisation of trap placement. They are both based on a technique called *simulated annealing*, which we will describe briefly. Consider the case with one trap. To find an optimal position for this trap, we could start with a random location for the trap and estimate the average amount of insects trapped. Then we could consider another random location in the neighbourhood and again estimate the average amount of particles caught by the trap, to see if this amount

increases. If it does, we would prefer our new location over the old one. This method is called the *hill climbing algorithm*. However, using this algorithm we run the risk of getting stuck in a local maximum. In other words, there is a chance the algorithm will not give us the ultimate position of the trap. There are many ways to overcome this obstacle. The technique of simulated annealing is one of them. The basic idea of this algorithm is similar to that of the hill climbing algorithm. The difference is that, while the hill climbing algorithm only allows us to move to trap locations with an improved result, using simulated annealing there is a probability of going to locations that do not increase the estimated average amount of insects caught. This way we can move out of local maxima and reach the ultimate position of the trap within our field. One of the algorithms we used for the optimisation of trap location is simulated annealing itself. The other one is an adaptation of this algorithm, where more probabilistic features are included.

Making use of these algorithms, we are glad to see the simulation results verify our calculus and show the centre of the field is the optimal location to place a single trap. However, we can now do more! Making use of the simulated annealing algorithm, we see placement of two traps is optimal on a circle around the middle of the area.

B. Matlab Code for Simulations

We provide *Matlab* code for some simulations discussed in the report.

B.1. poisson_fixed_time

```
function [state, n] = poisson_fixed_time ( lambda, time )
  state = zeros(1 , time);
  n = 0;
  NST = 0.0;
  NET = -log(rand) / lambda;
  i = 1;

  while(NST ≤ time)
    while ((NET > NST) && (time ≥ NST))
        state(i) = n;
        NST = NST + 1;
        i = i + 1;
    end
    dt = - log ( rand ) / lambda;
    NET = NET + dt;
    n = n + 1;
  end

  stairs(state)
  return
end
end
```

B.2. expected_value_poisson

```
function i = expected_value_poisson(lambda, time, N)
summed_state = zeros(1 , time*10+1);
i = 0;

while i < N
    [state, ] = poisson_fixed_time(lambda, time);
    summed_state = summed_state + state;
    i = i + 1;
end
```

```
10
11 summed_state = summed_state./N;
12
13 plot(summed_state)
14
15 end
```

B.3. MM1_mean (Method 1)

```
1  %function MM1_mean(lambda, mu, Horizon, StepSize, NumberSamplePaths)
2  clear all; clc;
3  lambda = 1;
4  mu = 3;
5  Horizon = 20;
6  StepSize = 1;
7  NumberSamplePaths = 10000;
8
9  TotalValues = zeros(1, Horizon/StepSize+1);
10 SamplePath = zeros(1, Horizon/StepSize+1);
11
12
13 for SamplePathNumber = 1:NumberSamplePaths
14
15     NextSampleTime = StepSize;
16     NextEventTime = -log(rand)/lambda;
17     SamplePathIndex = 2;
18     CurrentState = 0;
19
20     while (NextSampleTime ≤ Horizon)
21
22         while (NextSampleTime < NextEventTime) && (NextSampleTime ≤ ...
               Horizon)
23             SamplePath(1, SamplePathIndex) = CurrentState;
24             NextSampleTime = NextSampleTime + StepSize;
25             SamplePathIndex = SamplePathIndex + 1;
26         end
27
28         r = rand;
29         if r < lambda/(lambda+mu*(CurrentState>0));
30             CurrentState = CurrentState + 1;
31         else
32             CurrentState = CurrentState - 1;
33         end
34
35         NextEventTime = ...
               NextEventTime-log(rand)/(lambda+mu*(CurrentState>0));
36
37     end
38
39     TotalValues = TotalValues + SamplePath;
```

```
40      SamplePath(1,1) = 0;
41
42  end
43
44
45  plot(0:StepSize:Horizon, ...
        TotalValues./NumberSamplePaths,0:StepSize:Horizon,
46  ones(1, length(0:StepSize:Horizon)).*(lambda/(mu-lambda)))
47
48  end
```

B.4. MM1_mean2 (Method 2)

```
1   %function MM1_mean2(lambda, mu, Horizon, StepSize, NumberSamplePaths)
2   clear all; clc;
3   lambda = 1;
4   mu = 3;
5   Horizon = 20;
6   StepSize = 1;
7   NumberSamplePaths = 10000;
8
9   TotalValues = zeros(1, Horizon/StepSize+1);
10  SamplePath = zeros(1, Horizon/StepSize+1);
11
12
13  for SamplePathNumber = 1:NumberSamplePaths
14
15      NextSampleTime = StepSize;
16      NextEventTime = -log(rand)/lambda;
17      EventList = [NextEventTime, 1];
18      SamplePathIndex = 2;
19      CurrentState = 0;
20
21      while (NextSampleTime ≤ Horizon)
22
23          while (NextSampleTime < NextEventTime) && (NextSampleTime ≤ ...
                Horizon)
24              SamplePath(1, SamplePathIndex) = CurrentState;
25              NextSampleTime = NextSampleTime + StepSize;
26              SamplePathIndex = SamplePathIndex + 1;
27          end
28
29          EventType = EventList(1,2);
30          switch EventType
31              case 1
32                  CurrentState = CurrentState + 1;
33                  EventList = [EventList(2:end,:);...
34                               NextEventTime-log(rand)/lambda,1];
35                      if CurrentState == 1
```

```
                                EventList = [EventList; ...
                                    NextEventTime-log(rand)/mu, 2];
                            end
                case 2
                    CurrentState = CurrentState - 1;
                    EventList = EventList(2:end,:);
                    if CurrentState > 0
                        EventList = [EventList; NextEventTime - ...
                            log(rand)/mu, 2];
                    end
            end

            EventList = sortrows(EventList);
            NextEventTime = EventList(1,1);


        end

        TotalValues = TotalValues + SamplePath;
        SamplePath(1,1) = 0;
        SamplePathNumber/NumberSamplePaths

    end


    plot(0:StepSize:Horizon, ...
        TotalValues./NumberSamplePaths,0:StepSize:Horizon,
    ones(1, length(0:StepSize:Horizon)).*(lambda/(mu-lambda)))

    end
```

B.5. MMinfty_mean (Method 1)

```
%function MMinfty_mean(lambda, mu, Horizon, StepSize, NumberSamplePaths)
clear all; clc;
lambda = 1;
mu = 3;
Horizon = 20;
StepSize = 1;
NumberSamplePaths = 10000;

TotalValues = zeros(1, Horizon/StepSize+1);
SamplePath = zeros(1, Horizon/StepSize+1);


for SamplePathNumber = 1:NumberSamplePaths

    NextSampleTime = StepSize;
    NextEventTime = -log(rand)/lambda;
    SamplePathIndex = 2;
```

```
18      CurrentState = 0;
19
20      while (NextSampleTime ≤ Horizon)
21
22          while (NextSampleTime < NextEventTime) && (NextSampleTime ≤ ...
                Horizon)
23              SamplePath(1, SamplePathIndex) = CurrentState;
24              NextSampleTime = NextSampleTime + StepSize;
25              SamplePathIndex = SamplePathIndex + 1;
26          end
27
28          r = rand;
29          if r < lambda/(lambda+mu*CurrentState*(CurrentState>0));
30              CurrentState = CurrentState + 1;
31          else
32              CurrentState = CurrentState - 1;
33          end
34
35          NextEventTime = NextEventTime-log(rand)/
36          (lambda+CurrentState*mu*(CurrentState>0));
37
38      end
39
40      TotalValues = TotalValues + SamplePath;
41      SamplePath(1,1) = 0;
42
43  end
44
45
46  plot(0:StepSize:Horizon, ...
        TotalValues./NumberSamplePaths,0:StepSize:Horizon,
47  ones(1, length(0:StepSize:Horizon)).*(lambda/mu))
48
49  %end
```

B.6. MMinfty_mean2 (Method 2)

```
1   %function MMinfty_mean2(lambda, mu, Horizon, StepSize, NumberSamplePaths)
2   clear all; clc;
3   lambda = 1;
4   mu = 3;
5   Horizon = 20;
6   StepSize = 1;
7   NumberSamplePaths = 10000;
8
9   TotalValues = zeros(1, Horizon/StepSize+1);
10  SamplePath = zeros(1, Horizon/StepSize+1);
11
12
13  for SamplePathNumber = 1:NumberSamplePaths
```

```
14
15      NextSampleTime = StepSize;
16      NextEventTime = -log(rand)/lambda;
17      EventList = [NextEventTime, 1];
18      SamplePathIndex = 2;
19      CurrentState = 0;
20
21      while (NextSampleTime ≤ Horizon)
22
23          while (NextSampleTime < NextEventTime) && (NextSampleTime ≤ ...
                Horizon)
24              SamplePath(1, SamplePathIndex) = CurrentState;
25              NextSampleTime = NextSampleTime + StepSize;
26              SamplePathIndex = SamplePathIndex + 1;
27          end
28
29          EventType = EventList(1,2);
30          switch EventType
31              case 1 % arrival
32                  CurrentState = CurrentState + 1; % Update state
33                  EventList = [EventList(2:end,:);... % Remove arrival ...
                        from event list
34                                  NextEventTime-log(rand)/lambda,1];% ...
                                      Schedule next arrival
35                                  EventList = [EventList; ...
                                      NextEventTime-log(rand)/mu, 2];
36                                  % If item is only one present, then ...
                                      schedule its departure
37              case 2 % departure
38                  CurrentState = CurrentState - 1;
39                  EventList = EventList(2:end,:); % Remove departure ...
                        from event list
40          end
41
42          EventList = sortrows(EventList);
43          NextEventTime = EventList(1,1);
44
45
46      end
47
48      TotalValues = TotalValues + SamplePath;
49      SamplePath(1,1) = 0;
50      SamplePathNumber/NumberSamplePaths;
51
52  end
53
54
55  figure = plot(0:StepSize:Horizon, ...
        TotalValues./NumberSamplePaths,0:StepSize:Horizon,
56  ones(1, length(0:StepSize:Horizon)).*(lambda/mu))
57
58  %end
```

B.7. positie_optimalisatie_2 (Algorithm 2.1.1)

```
pos_x = rand;
pos_y = rand;

A = Gebied_simulaties(pos_x, pos_y);
T = 1;
T_min = 0.000001;

x_geweest = [pos_x];
y_geweest = [pos_y];


while T > T_min
    i = 1;
    while i ≤ 100
        pos_x_2 = rand;
        pos_y_2 = rand;
        A_2 = Gebied_simulaties(pos_x_2, pos_y_2);
        if min([1 exp(-(A-A_2)/T)]) ≥ rand
            pos_x = pos_x_2;
            pos_y = pos_y_2;
            A = A_2;
        else
            pos_x = pos_x;
            pos_y = pos_y;
            A = A;
        end
        i = i + 1;
        x_geweest = [x_geweest; pos_x];
        y_geweest = [y_geweest; pos_y];
    end
    T = T*0.9;
end
Data = [0, 0; 1, 1; x_geweest, y_geweest];

hold on
hist3(Data)


n = hist3(Data, [15 15]); % default is to 10x10 bins
n1 = n';
n1(size(n,1) + 1, size(n,2) + 1) = 0;

xb = linspace(min(Data(:,1)),max(Data(:,1)),size(n,1)+1);
yb = linspace(min(Data(:,2)),max(Data(:,2)),size(n,1)+1);

h = pcolor(xb,yb,n1);

h.ZData = ones(size(n1)) * -max(max(n));
colormap(hot) % heat map
title('Reached points');
```

```
grid on
view(3)


optimum = [pos_x pos_y]
A = A
```

B.8. particle_sim_grid (Algorithm 2.2.1)

```
lambda_0 = 10;%poisson parameter
gamma_0 = 0.1;
lambda = gamma_0*lambda_0;
beta = 1;%probability in box parameter
alpha = 5;% SA parameter
t = 20;
n = 1;  %Number of boxes
x_grid = linspace(0,1);
y_grid = linspace(0,1); %grid for the boxes

x_positions = randsample(x_grid, n);
y_positions = randsample(y_grid, n);%positions of the boxes

Totaal_particles = poissrnd(lambda*t,1,1);
x_part_pos = rand(Totaal_particles, 1);
y_part_pos = rand(Totaal_particles, 1);%Random poisson process amount ...
    of particles and location
Dist_to_boxes = zeros(Totaal_particles, n); %matrix keeps track of ...
    distance of particles to boxes

 for j=1:n
    Dist_to_boxes(:,j) = sqrt((x_part_pos - ...
        x_positions(j).*ones(Totaal_particles, 1)).^2+(y_part_pos - ...
        y_positions(j).*ones(Totaal_particles, 1)).^2);
 end

A_boxes = zeros(1, n); %will keep track of how many particles are in ...
    the boxes
prob_particle_in_box = rand(Totaal_particles, n);

for i=1:Totaal_particles
   [B, Dist_indices_ascending] = sort(Dist_to_boxes(i,:));
   for k=1:n
       if prob_particle_in_box(i, Dist_indices_ascending(k)) < ...
           exp(-beta*Dist_to_boxes(i, Dist_indices_ascending(k)))
           A_boxes(Dist_indices_ascending(k)) = ...
               A_boxes(Dist_indices_ascending(k)) + 1;
           break
       end
   end
end
```

```
35
36 x_geweest = x_positions;
37 y_geweest = y_positions;
38 A_geweest = A_boxes;
39
40 for l=1:100000
41     [C, I] = sort(A_boxes);
42     new_x = randsample(x_grid, 1);
43     new_y = randsample(y_grid, 1);
44     x_positions_new = x_positions; x_positions_new(I(1)) = new_x;
45     y_positions_new = y_positions; y_positions_new(I(1)) = new_y;
46
47     new_Totaal_particles = poissrnd(lambda*t,1,1);
48     new_x_part_pos = rand(new_Totaal_particles, 1);
49     new_y_part_pos = rand(new_Totaal_particles, 1);%Random poisson ...
           process amount of particles and location
50     new_dist_to_boxes = zeros(new_Totaal_particles, n); %matrix keeps ...
           track of distance of particles to boxes
51
52     for j=1:n
53         new_dist_to_boxes(:,j) = sqrt((new_x_part_pos - ...
               x_positions_new(j).*ones(new_Totaal_particles, ...
               1)).^2+(new_y_part_pos - ...
               y_positions_new(j).*ones(new_Totaal_particles, 1)).^2);
54     end
55
56     new_A_boxes = zeros(1, n);
57     prob_particle_in_box_2 = rand(new_Totaal_particles, n);
58
59     for i=1:new_Totaal_particles
60         [B, new_Dist_indices_ascending] = sort(new_dist_to_boxes(i,:));
61         for k=1:n
62             if prob_particle_in_box_2(i, ...
                   new_Dist_indices_ascending(k)) < ...
                   exp(-beta*new_dist_to_boxes(i, ...
                   new_Dist_indices_ascending(k)))
63                 new_A_boxes(new_Dist_indices_ascending(k)) = ...
                       new_A_boxes(new_Dist_indices_ascending(k)) + 1;
64                 break
65             end
66         end
67     end
68     p = rand;
69     if sum(new_A_boxes) ≤ lambda_0*t
70         if  p < exp(alpha*(sum(A_boxes)-sum(new_A_boxes))/
71         (sum(new_A_boxes)-lambda_0*t))
72             x_positions = x_positions_new;
73             y_positions = y_positions_new;
74             lambda = max([-1/(sum(A_boxes)+1)+1, ...
                   -1/(sum(new_A_boxes)+1)+1])*lambda_0;
75             %lambda = max([sum(A_boxes)/Totaal_particles, ...
                   sum(new_A_boxes)/new_Totaal_particles])
76             Totaal_particles = new_Totaal_particles;
77             A_boxes = new_A_boxes;
```

```
78          end
79      else
80          if p < exp(-alpha*(sum(A_boxes)-sum(new_A_boxes))/
81          (sum(new_A_boxes)-lambda_0*t))
82             x_positions = x_positions_new;
83             y_positions = y_positions_new;
84             lambda = max([-1/(sum(A_boxes)+1)+1, ...
                   -1/(sum(new_A_boxes)+1)+1])*lambda_0;
85             %lambda = max([sum(A_boxes)/Totaal_particles, ...
                   sum(new_A_boxes)/new_Totaal_particles])
86             Totaal_particles = new_Totaal_particles;
87             A_boxes = new_A_boxes;
88          end
89      end
90      x_geweest = [x_geweest; x_positions];
91      y_geweest = [y_geweest; y_positions];
92      A_geweest = [A_geweest; A_boxes];
93  end
94
95  lambda
96  A_boxes
97  x_positions
98  y_positions
99
100 [x_geweest , y_geweest];
101 A_geweest;
102
103 Data = [0, 0; 1, 1];
104
105 for o=1:n
106     Data = [Data; x_geweest(:,n), y_geweest(:,n)];
107 end
108
109 hold on
110 hist3(Data);
111
112
113 n = hist3(Data, [20 20]);
114 n1 = n';
115 n1(size(n,1) + 1, size(n,2) + 1) = 0;
116
117 xb = linspace(min(Data(:,1)),max(Data(:,1)),size(n,1)+1);
118 yb = linspace(min(Data(:,2)),max(Data(:,2)),size(n,1)+1);
119
120 h = pcolor(xb,yb,n1);
121
122 h.ZData = ones(size(n1)) * -max(max(n));
123 colormap(hot) % heat map
124 title('Reached points');
125 grid on
126 view(3)
127
128 hold off
```

B.9. Positie_optimalisatie_2dozen (Algorithm 2.1.1 for two traps)

```
pos_x = rand;
pos_y = rand;
pos_x_2 = rand;
pos_y_2 = rand;

A = Gebied_simulaties_2dozen(pos_x, pos_y, pos_x_2, pos_y_2);
T = 1;
T_min = 0.00001;

x_geweest = [pos_x];
y_geweest = [pos_y];
x_geweest_2 = [pos_x_2];
y_geweest_2 = [pos_x_2];
dist_geweest = [sqrt((0.5 - pos_x)^2 + (0.5 - pos_y)^2);sqrt((0.5 - ...
    pos_x_2)^2 + (0.5 - pos_y_2)^2)];



while T > T_min
    i = 1;
    while i ≤ 100
        if mod(i,2) == 0
            pos_x_3 = rand;
            pos_y_3 = rand;
            A_2 = Gebied_simulaties_2dozen(pos_x_3, pos_y_3, pos_x_2, ...
                pos_y_2);
            if min([1 exp(-(A-A_2)/T)]) ≥ rand
                pos_x = pos_x_3;
                pos_y = pos_y_3;
                A = A_2;
            else
                pos_x = pos_x;
                pos_y = pos_y;
                pos_x_2 = pos_x_2;
                pos_y_2 = pos_y_2;
                A = A;
            end
        else
            pos_x_3 = rand;
            pos_y_3 = rand;
            A_2 = Gebied_simulaties_2dozen(pos_x, pos_y, pos_x_3, ...
                pos_y_3);
            if min([1 exp(-(A-A_2)/T)]) ≥ rand
                pos_x_2 = pos_x_3;
                pos_y_2 = pos_y_3;
                A = A_2;
            else
                pos_x = pos_x;
```

```
                pos_y = pos_y;
                pos_x_2 = pos_x_2;
                pos_y_2 = pos_y_2;
                A = A;
            end
        end
        x_geweest = [x_geweest; pos_x];
        y_geweest = [y_geweest; pos_y];
        x_geweest_2 = [x_geweest_2; pos_x_2];
        y_geweest_2 = [y_geweest_2; pos_y_2];
        dist_geweest = [dist_geweest; sqrt((0.5 - pos_x)^2 + (0.5 - ...
            pos_y)^2);sqrt((0.5 - pos_x_2)^2 + (0.5 - pos_y_2)^2)];
        i = i + 1;
    end
    T = T*0.9;
end



optimum = [pos_x pos_y, pos_x_2, pos_y_2]
Dist_to_center_1 = sqrt((0.5 - pos_x)^2 + (0.5 - pos_y)^2)
Dist_to_center_2 = sqrt((0.5 - pos_x_2)^2 + (0.5 - pos_y_2)^2)
Dist_to_eachother = sqrt((pos_x - pos_x_2)^2 + (pos_y - pos_y_2)^2)
A = A

Data_1 = [0, 0; 1, 1; x_geweest, y_geweest; x_geweest_2, y_geweest_2];
Data_2 = [0; dist_geweest];
```