University of Southern Queensland

Faculty of Engineering & Surveying

Simulation of Monolayer Distribution for Use in Evaporation Reduction

A dissertation submitted by

Matthew du Preez

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Mechanical Engineering

Submitted: October, 2012

Abstract

The use of monolayers to reduce evaporation was originally developed in 1925 (Frenkiel 1965). The major problem with using monolayers is the difficulty associated with the prediction and control of the distribution of the monolayer to the water surface. Full scale experiments are not very successful due to difficulties quantitatively measuring the evaporation resistance the monolayer provides to different areas of the water surface.

A simulation of the monolayer as it disperses and degrades on the water surface can be used to predict the long term performance as well as a real time control system. The model was created in MATLAB and simulates the behaviour of the monolayer based on experimental results. The simulation selects the optimal application rate for each individual applicator based on maximising the amount of money saved by comparing the money gained from all possible permutations of applicator rates and applicator positions. The money saved has two parts, the cost associated with distributing the monolayer and the value of the water saved from evaporation. This comparison is performed for each small time step. After the optimal permutation of rates has been found, the optimal permutation of rates for the following time step is calculated.

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Prof F Bullen

Dean

Faculty of Engineering and Surveying

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Matthew du Preez

0050101486

_____

Signature

_____

Date

Acknowledgments

I would like to thank Dr Andrew Wandel, my supervisor, for his assistance and guidance throughout the duration of this project. I wish to thank my family for their support and encouragement.

Matthew du Preez

University of Southern Queensland
October 2012

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Water evaporation in large dams is a significant problem, especially with unpredictable weather patterns which make it challenging to forecast rainfall and wind. Monolayers have proven effective at reducing evaporation during field tests for many years but their long term behaviour has been difficult to predict (Schmidt & Scobie 2012 p. 25). This has limited their use for large dams, where they are most effective.

Schmidt and Scobie (2012) reviewed several different methods for reducing evaporation in farm storage dams. They noted for monolayers that "frequent product application is required given the effects of wind, waves, UV radiation, algae and bacteria on product distribution and longevity" (p. 22). The main advantages of monolayers over competing evaporation reduction products is "...the low initial setup cost. Additionally, the product need be applied only when it is required, for example when the dam is full and during periods of high evaporation" (p. 22).

Another problem with the use of monolayers, especially during field trials is "monolayers generally cannot be seen clearly by eye on the surface of a dam limiting confidence in the technology. Increased water surface tension does allow detection under light wind conditions through smoothing of surface wavelets. Various methods for monolayer detection have been researched (Coop, Lamb, Fellows & Bradbury 2011) however no commercially viable approach is available. Accurate quantification of water savings is also a challenge" (Schmidt & Scobie 2012, p. 24). It is also worth noting the long

term performance of monolayers may be different from short trials. "McJannet et al. (2008) highlight that long term trials of monolayers have not been conducted and that suppression of evaporation will raise water temperature thereby limiting further evaporation savings" (Schmidt & Scobie 2012 p. 24).

The aim of this project was to develop a simulation of the behaviour of a monolayer film on the surface of a large dam. The model was then used to select optimal applicator rates. This model should make it easier to predict the long term performance of the monolayers for evaporation reduction.

The objectives of the project are:

- Research experiments to obtain data and equations to describe behaviour of the monolayer.
- Create matrices to represent domain, boundaries and applicator positions for simulation.
- Simulate monolayer movement and degradation over time, using historical weather data as input.
- Use an objective function to control application of monolayer.
- Validate model by comparing results of simulation to real experiments.

If time permits:

- Use model to optimise location of applicators.

1.1 Design Requirements

The model needed to meet several critical design criteria. The most important aspect of the model was that it needed to be written in a generic style so that it could easily accommodate different monolayers, as well as make it easy to input the different forms of degradation that a particular monolayer is susceptible to.

The model needed to achieve the following performance requirements:

- Generic model
- Accept a variety of different parameters to analyse performance with sufficient detail
- Be easily expanded for different forms of degradation
- Facilitate plotting to show results of simulation graphically

The model was designed to simulate the movement of monolayer on the surface of the water when subjected to varying wind speeds and directions. The major constraint placed on the model is due to performance issues arising from simulating a large number of combinations of application rates for each time step. The behaviour of the monolayer was separated into different sections, which could be described by experiments.

1.2 Design Methodology

The model was developed in MATLAB because it is a high level programming language that also has excellent graphing functions. The simulation optimises the application rate for each individual applicator by testing all the permutations for each time step and selecting the optimal rate based on the amount of money saved. The amount of money saved is the value of water saved from evaporation minus the cost of applying the monolayer to the water. This approach is better than making a judgement on how much monolayer can be used to cover a given area of water for each time step because it removes the possibility of different people making different decisions. This means that given the same situation, the simulation will always make the same decision.

1.3 Analysis and Performance

The model is capable of selecting an optimal application rate but is currently limited in its use to real world applications because it still requires some types of degradation

of the monolayer to be quantified before they can be incorporated into the model.

1.4 Overview of the Dissertation

This dissertation is organised as follows:

Chapter 2 reviews the literature

Chapter 3 discusses the methodology behind the model

Chapter 4 discusses the detailed design of the model and why it has been written in a particular manner. This chapter explains the major aspects of the important scripts and functions used in the model.

Chapter 5 explains the purpose and background for each section of code. The chapter shows the order and explains how the various scripts and functions are linked together.

Chapter 6 discusses the results obtained from running the model.

Chapter 7 summarises the achievements of this research project and suggests further work in the area of evaporation reduction using monolayers.

Chapter 2

Literature Review

2.1 Introduction

Monolayers were first used for evaporation reduction in 1925 (Frenkiel 1965). Monolayers are chemical films that are one molecule thick and form a phase boundary between the water and the air. There are two properties that make them useful for evaporation reduction. Firstly, monolayers are insoluble in water which ensures that the monolayer and water will remain as separate chemicals. This occurs because the head of the molecule is attracted to water (hydrophilic) while the tail repels water (hydrophobic). The second important property is that the molecules of the monolayer are anchored to the water and are therefore unable to pile up on top of each other (Brink 2011).

The surface pressure, Π is defined as the reduction in surface tension due to the addition of the monolayer to the water surface. It is the difference in surface tension of pure water, γ_w, on one side of the barrier and the film on the water on the other side, γ_f (Barnes & Gentle 2011, Rastogi 2003).

$$\Pi = \gamma_w - \gamma_f \tag{2.1}$$

If the 2D film behaves as an ideal gas, then the perfect gas law is applicable, $\Pi A = kT$, where Π is 2D pressure, A is the area occupied by one molecule, k is the Boltzmann constant and T is the temperature. Rastogi states that when Π is low and the area occupied is large, increasing Π will compress the film. This means that ideal gas be-

haviour is no longer applicable and real gas behaviour should be used. A 2D monolayer obeys a law similar to the van der Waals equation, $\left(\Pi + \alpha/A^2\right) + (A - \beta) = kT$ where α is a constant that describes the mutual attraction forces between the hydrocarbon tails and β is the force of repulsion that acts in the cross sectional area of the molecule in the adsorbed monolayer.

2.2 Surfactants

Surfactants are compounds that lower the surface tension of a liquid or the interfacial tension between two liquids or between a liquid and a solid. Surfactants have many uses including detergents, wetting agents, emulsifiers, foaming agents and dispersants. The behaviour of surfactants can be explained by looking at the behaviour of the molecules. Some molecules in surfactants prefer one phase and other molecules prefer another phase. These materials concentrate at the interface and adsorb. An example is the use of a detergent to disperse an oil in water, when usually water and oil are insoluble. The resulting adsorbed layers are one molecule thick (1-3 nm) and can be described as nanofilms. This leads to methods of creating films that are one molecule thick using the Langmuir-Blodgett technique for air-water interfaces (Barnes & Gentle 2011, p. 6).

Myer (2006, p. 95) describes the two aspects of the performance regarding surfactants that are needed in order to lower the surface tension of a solution: sufficient concentration of surfactant to produce a given surface tension, and maximum tension reduction possible regardless of concentration of surfactant present.

Rastogi (2003) says that since the thickness of a monolayer is 10^{-7} cm, the system is 2D and the surface pressure Π, can be expressed as dynes/cm as opposed to dynes/cm^2 for a 3D system. The surface pressure of the monolayer is related to the area as shown by Barnes and Gentle (2011, p. 112). There are four major monolayer phases: gaseous, liquid expanded, liquid condensed and a solid phase. This agrees with the simplified figure Rastogi (2003, p. 163) presented. Due to the narrow range of temperatures and pressures used in practice, not all phases will be seen (Barnes & Gentle 2011).

2.3 Evaporation resistance

Henry et al. (2010) describe the accessible area theory of evaporation resistance with the following equation

$$r = \frac{1}{\alpha}\sqrt{\frac{2\pi M}{RT}}\left(\frac{A}{a} - 1\right) \tag{2.2}$$

where A is the total surface area; a is the accessible area; α is the evaporation coefficient; M is the molar mass of water; R is the ideal gas constant; and T is the temperature. The theory states that evaporation occurs at the same rate through holes in the monolayer as a clean surface. This theory doesn't account for the alkyl chain length or impurities, which has a large effect, therefore this theory is not valid for use in large scale.

There are several reasons why it is difficult to correlate current theory and experimental data (Langmuir 1998 cited in Barnes 2001). One reason that Langmuir proposes is that equilibrium is not established when the area is small or the surface pressure is high, and these conditions are not explained by surface pressure–area curves when they were measured. The experimental results Barnes obtained supported his hypothesis and explained the difference between the theory based on equilibrium and experiments which do not have enough time to reach equilibrium. Detailed information regarding the structure of Langmuir–Blodgett films have been described by Peng et al. (2001).

2.4 Monolayer Behaviours

The behaviours of the monolayer include:

- describing the movement under variety of wind speeds
- monolayer losses due to shoreline interaction
- evaporation of the monolayer itself
- monolayer submergence – monolayer sinks below surface and is lost
- degradation due to biological attack – bacteria breaks down the monolayer, reducing effectiveness

The main behaviours that have been simulated are the movement and degradation of the monolayer due to biological attack and evaporation of the monolayer.

2.4.1 Movement

Brink (2011) observed two main shapes that the monolayer formed at different wind speeds during experiments he conducted in a 6 m diameter tank. At low wind speeds, the monolayer spreads in a circular manner, relatively unaffected by the wind. At higher speeds the monolayer forms a sector of a circle (a portion of a circle that is enclosed by two radii and an arc). Brink (2011) reported that Vines (1960), McArthur (1962) and Crow and Mitchell (1975) agreed that wind-induced drift started to occur at wind speeds of approximately 3.2 km/h. At wind speeds below this, the spreading rate can be considered to be purely a function of the surface tension gradient between the monolayer and the water.

Brink (2011, p. 64) reported that all previous work had been conducted on very small surfaces ($< 3\,m^2$), and for extremely short periods of time ($< 5\,s$). The monolayer movement can be described using equations obtained by Brink (2011). The major variable was the wind speed which controlled the drift speed and angle of spread of the monolayer. Brink conducted experiments in a 6 m diameter tank and over 55 s. The eccentricity of the assumed shape of a circle was on average 2.0% (Brink 2011, p. 74).

Brink performed experiments with tank diameters 0.3, 2 and 6 m and with three different theoretical monolayer thicknesses: 1, 2 and 6 times the minimum theoretical amount required to form a monolayer. The different thicknesses represent different application rates that can be used. Brink (p. 76 , 2011) found that for short periods of time, the spreading rate was a weak function of the application rate, for 1-6 x minimum theoretical amount (see Fig. 2.1 on page 10). The error in ignoring the effect of the application rate on the drift speed will be small, especially for short periods of time. The variation between different application rates is very small across the range of application rates for time less than 15 s, which corresponds to a radius of approximately 1.0 m. The average result from three experiments showed only 2.48 % variation for the 6 m tank. The equation of the curve for the 6 m tank with a 6x application rate was

used for the model since it is the most applicable for large scale use.

Brink reported Vines (1960), McArthur (1962) and Crow & Mitchel (1975) discovered that at a critical wind speed of 3.2 km/h the shape of the monolayer changed from a circle to a sector of a circle (wedge). The two equations which describe these two cases are shown below. Both equations are for a 6 m tank with six times the minimum theoretical application rate for a valid monolayer to form.

The equations below are for $C_{18}OH$ and Brij78 in a water-emulsion. Herzig et al. (2011) described an emulsion as a liquid dispersed in liquid with the long-chain alcohols solid at room temperature, but both Herzig and Brink melted the alcohols before mixing with water. Emulsions are used because a liquid will have a greater spreading rate compared to solid powder.

The following equation shows the drift speed of the monolayer as a function of time:

$$u_{drift,circle} = k_D t^n \tag{2.3}$$

where $k_D = 0.1436$; t is the elapsed time (s); and $n = 0.7351$. The equation Brink found matches well with the generally accepted value of n = 0.75 (Brink 2011, p. 84).

The drift speed of the monolayer above the critical wind speed of 3.2 km/h is

$$u_{drift,wedge} = 0.0459\, u_{wind} - 0.0661 \tag{2.4}$$

where u_{wind} is the wind speed (m/s). Brink (2011, p. 104) found that the drift speed of the monolayer is a weak function of the application rate in the range of 17-53 mL/min (see Fig. 2.2 on page 11). Ignoring the application rate will therefore not introduce large errors into the model.

Herzig et al. (2011) described an experiment to determine how the spreading rates are related to the evaporation rates for emulsified monolayers. Emulsified monolayers aim to increase the spreading rates of monolayers at the cost of reduced evaporation resistance. The final surface pressure is also dependant on the initial surface pressure (Fig. 2.3a). At initial pressures greater than 15 mN m^{-1} any additional monolayer added to the surface has no effect on the evaporation resistance as there is no increase in surface pressure. The rate of application affects the surface pressure (Fig. 2.3b).

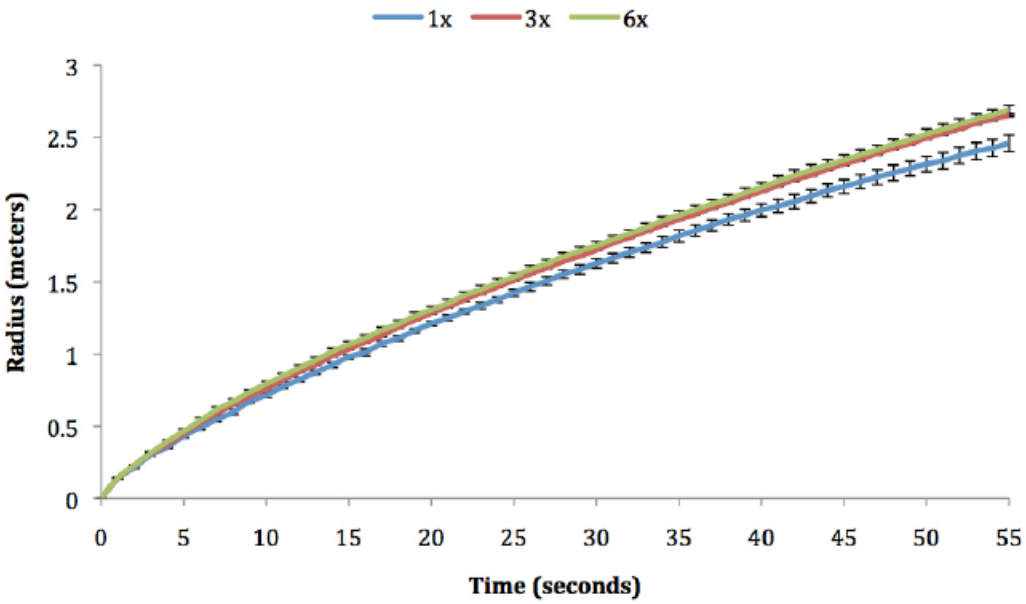


Figure 2.1: The leading edge radius for different application rates with no wind. Reproduced from Brink (2011).

The resistance is a function of the surface pressure and time for this experiment, but Herzig et al. did not consider the effect of wind (see Fig. 2.3c).

2.4.2 Relating evaporation resistance to windspeed

McJannet, Knight, Cook and Burns (2008) used a linear model to describe the relationship between evaporation resistance and wind speed because they were unable to find better data. This data showed a decrease in evaporation resistance with wind speed. This places three constraints on the model:

- Wind speed is less than 6.71 m/s
- Concentration of monolayer is greater than two times the minimum theoretical amount to create a valid monolayer. This is to ensure that a monolayer has definitely formed.
- Equation of the straight line to describe reduction in monolayer resistance at higher wind speeds.

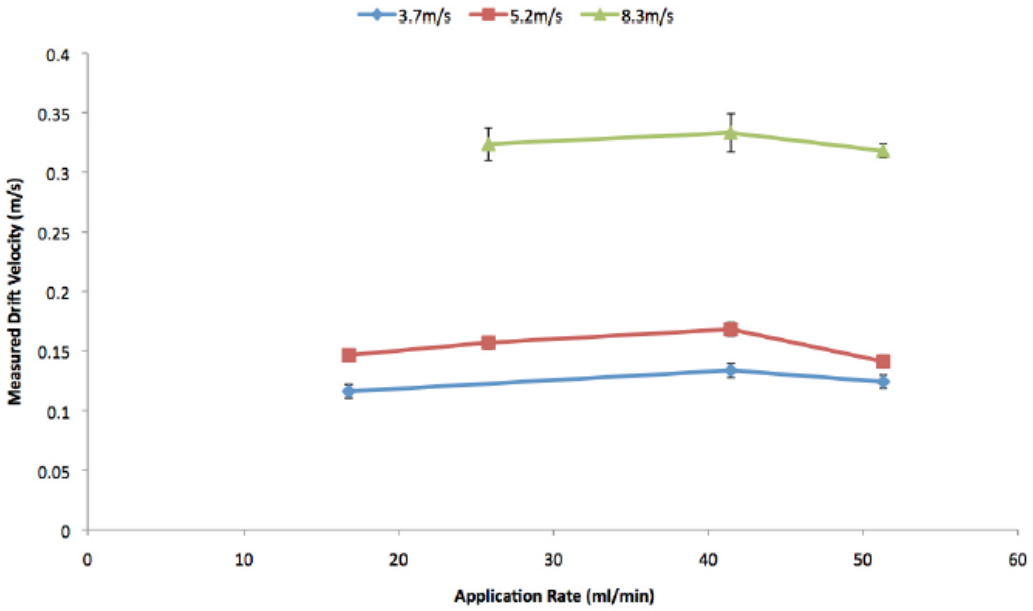
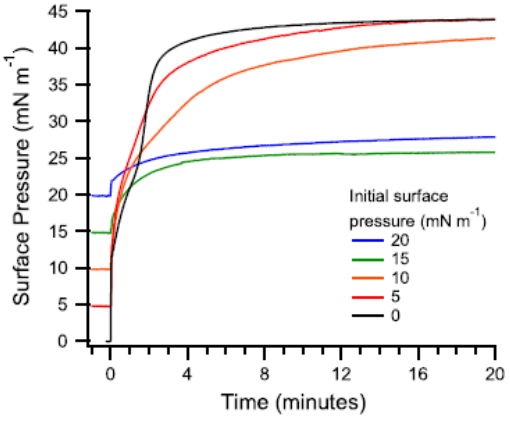


Figure 2.2: Drift speed of monolayer for different application rates at various wind speeds. Reproduced from Brink (2011).

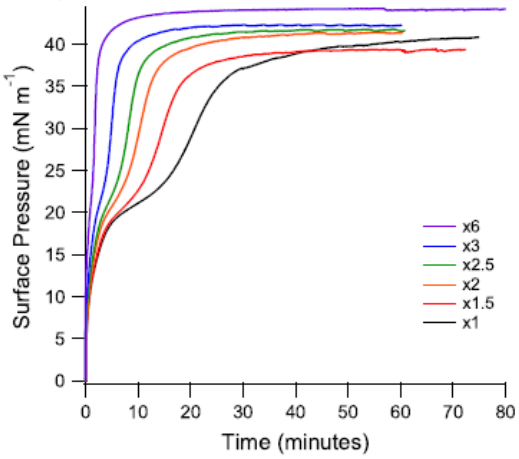
Brink (2011) states that the consensus reached by many researchers is that at wind speeds above 26.4 (± 5) km/h it becomes impractical to apply monolayer. This is because the evaporation resistance at higher wind speeds approaches zero (McJannet et al. 2008, p. 6). It is however necessary to have a wind speed higher than zero and not too high humidity in order to spread the monolayer across the surface of the dam in a reasonable period of time (Gladyshev 2002 cited in Brink 2011, p. 46).

2.4.3 Monolayer losses due to shoreline interaction

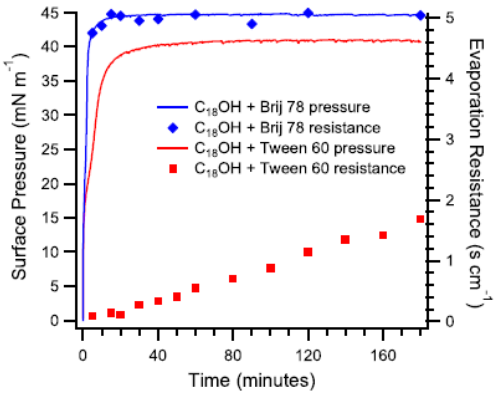
The effect of the shoreline is variable depending on the slope and material of the shoreline. The slope of the shoreline will determine how easily the monolayer is deposited on the shoreline and what proportion is reintroduced when the wind changes direction. The type of shoreline (e.g. sand, soil, rock, vegetation) will also have an effect on the rate and proportion of monolayer removed and returned to the surface of the water.



(a)



(b)



(c)

Figure 2.3: Herzig et al. (2011) experimental results for $C_{18}OH$ and Brij78 in a water-emulsion. (a) Effect of initial surface pressure on final surface pressure, (b) Effect of application rate on surface pressure, (c) Evaporation resistance vs surface pressure and time.

Table 2.1: Loss of monolayer material due to evaporation. Reproduced from Barnes (2008).

Monolayer	Surface pressure (Π/mN m^{-1})	Half life (20°, $t_{\frac{1}{2}}$/h)	Half life (40°, $t_{\frac{1}{2}}$/h)
Cetyl alcohol	35	48	1.3
Stearyl alcohol	35	> 200	9.6
Cetyl/stearyl mix, 33/67	35	-	3.2
Cetyl/stearyl mix, 67/33	35	-	1.9

Table 2.2: Loss of monolayer material due to evaporation. Reproduced from Brink (2011).

Monolayer	Fractional loss ($\times 10^{-6}$ s^{-1})		
	$5°C$	$20°C$	$40°C$
Myristyl $C_{14}OH$	20	58	1900
Cetyl $C_{16}OHl$	1	4	150
Stearyl $C_{18}OH$	0	0	20

2.4.4 Volitalisation

Volitalisation describes the evaporation of the monolayer. Barnes (2008) gave a reason for the significant losses of monolayer material of hexadecanol over several days. Barnes found an experiment conducted by Brooks and Alexander (1960) that proves that evaporation of the monolayer into the atmosphere was the cause for these losses. The results are reproduced in Table 2.1.

Brink (2011) also adapted a table from Brooks and Alexander (1960) which is shown in Table B.1 on page 75. It shows that longer chained alcohols have a much lower fractional loss rate. Mansfield (1959) cited in Brink (2011, p.23-4), developed Equation 2.5 to describe the losses due to volitalisation.

$$dF_e = \frac{D_a c_a v_2}{40 c_s} \tag{2.5}$$

where dF_e is the fractional loss by volatilisation ($\frac{1}{s}$), D_a is the coefficient of diffusion of the monolayer (cm^2 s^{-1}), c_a is the concentration of vapour in equilibrium with the

monolayer (g cm^{-3}), v_2 is the wind speed (cm s^{-1}), and c_s is the surface concentration of the monolayer (g cm^{-2}). Mansfield applied empirical values to equation 2.5 for a cetyl alcohol monolayer at 40 mN/m and introduced a correction for low wind speeds.

$$dF_e = 6.8 \times 10^{-7} v_2 \tag{2.6}$$

2.4.5 Monolayer submergence

The submergence of the monolayer occurs at higher wind speeds and during rainfall (Brink 2011, p. 45). Brink (2011, p.26) was only able to find two studies which investigated the effect of rain on the performance of monolayers. A study conducted by Green and Houk (1979) found the effect to be significant and increased considerably with drop size and rainfall intensity. A study by Bair (1972) found the opposite, but Brink concludes that the likely cause for the discrepancy is the low concentration of monolayer used by Green and Houk (1 mg/m^2), as well as the inadequate size of the the tank used. Brink was unable to find research to quantify these results.

Brink (2011, p. 59) states that since emulsions are particles dispersed in water, submergence will need to be considered. The purified $C_{18}OH$ and Brij78 in water emulsion showed "... no evidence of submergence, however with unpurified $C_{18}OH$ (which was used for all the experimental work in [Brink's] thesis), 33 to 50 % of the $C_{18}OH$ was lost to submergence." Despite this significant loss, Brink and Herzig maintain that $C_{18}OH$ and Brij78 in water emulsion is still a practical means of applying monolayer to reduce evaporation. Herzig et al. (2011, cited in Brink 2011) says the losses can be reduced by "... ensuring the dispersed particles within the emulsion are kept as small as possible and is gently applied to the water surface." This may mean significant losses will occur if the monolayer is applied above certain wind speeds, or if the emulsion separates over time before it is applied to the water surface.

In order to investigate the use of monolayer products with large losses due to submergence, it would be worth investing time to determine the losses as a function of time and wind speed so it can be added to the model. This is a possibility for future work. The effects of submergence will not be considered further for the current model due to lack of appropriate experimental results.

2.4.6 Degradation due to biological attack

Barnes (2008, p. 349) cites laboratory experiments conducted by Chang et al. (1962) who discovered that "Hexadecanol and octadecanol mixed with inorganic agar were able to support the growth of colonies of *Pseudomonas* and *Flavobacterium sp.*, but not of other bacterial species commonly found in water storages. However some of these other species were able to grow in association with *Pseudomonas* or *Flavobacterium sp.* suggesting that they were able to feed on the breakdown products." Barnes concluded that after 3-4 days there was a "serious" decline in the evaporation resistance of the monolayer.

Brink (2011) reported on a laboratory experiment conducted by Dr Pittaway at the University of Southern Queensland as shown in Fig. 2.4. It appears that the decrease in concentration of the monolayer can be approximated as linear – the use of another function may imply a relationship which does not exist considering the limited number of data points.

The following characteristics identified by Brink (2011) indicate a higher rate of biological activity and therefore a lower useful life for monolayer material: regular algal blooms, high UV absorbance, dark brown water colour and a relatively high area ($< 1\,\text{ha}$).

The concentration for 0, 2, 3 and 4 days was read from the graph and plotted on linear axes. A linear line of best fit was plotted that was within the error bars (see Appendix B for further details).

2.5 Calculating Evaporation

The rate of evaporation can be found using a modified Penman-Monteith (1965) equation which was presented by McJannet, Knight, Cook and Burn (2008). The equation was originally designed to calculate the evaporation of water from plants. The equation is:

$$E = \frac{1}{\lambda}\left(\frac{\Delta_w\left(Q^* - N\right) + 86400\rho_a C_a\left(e_w^* - e_a\right)/\left(r_a + r_m\right)}{\Delta_w + \gamma}\right) \tag{2.7}$$

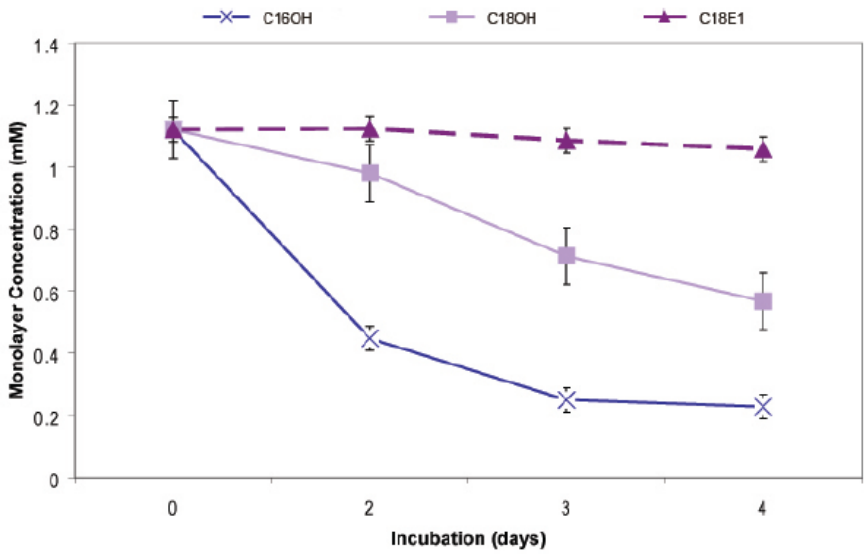


Figure 2.4: Microbial degradation of $C_{16}OH$, $C_{18}OH$ and C18E1 (Pittaway 2008).

where

λ is the latent heat of vaporisation (MJ kg^{-1})

Δ_w is the slope of the temperature saturation water vapour curve at water temperature (kPa/°C)

Q^* is net radiation (MJ m^{-2} d^{-1})

N is change in heat storage in the water body (MJ m^{-2} d^{-1})

ρ_a is density of air (kg m^{-3})

C_a is specific heat of air (MJ kg^{-1} K^{-1})

e_w^* is saturated vapour pressure at water temperature (kPa)

e_a is vapour pressure at air temperature (kPa)

r_a is aerodynamic resistance (s m^{-1})

r_m is monolayer resistance (s m^{-1})

γ is the psychometric constant (kPa $°C^{-1}$)

This equation requires many different inputs which are satisfied with several seconday equations. The full details have been presented by McJannet et al. (2008) and have

been used in the MATLAB code to evaluate the evaporation. A comparison for a small area $dx.dy$ can be made for the current monolayer resistance and no resistance. This can be used to determine the volume of water saved from evaporation and ultimately the profit gained for each small area $dx.dy$. This information has been used to select the most profitable combination of application rates for each of the individual

2.6 Conclusion

There have been several experiments in the field of monolayers but unfortunately very few of these experiments have produced results which can be used to quantify the effects of the various behaviours that monolayers exhibit. Brink has been successful in finding relationships to describe the movement of the monolayer. Pittaway has found a relationship to describe the degradation of the monolayer due to biological attack as a function of time.The losses associated with shoreline interaction and submergence have not yet been quantified experimentally.

The evaporation can be found by applying the modified Penman-Montheith equation. A comparison of the evaporation rate with and without the monolayer present can be used to calculate the volume of water saved from evaporation.

Chapter 3

Methodology

3.1 Introduction

There have been many good reasons why monolayers have not been adopted for use in large scale tests. The main reason is that the evaporation reduction arising from the use of monolayers can be very difficult to predict. A computer model can be used to achieve many of the results that experimental trials can achieve, but at a fraction of the cost and risk.

The model has several objectives. Firstly, it must not be limited to a specific monolayer since there are many different types of monolayers and there is yet to be a definitive monolayer due to the complex combination of advantages and disadvantages that each poses. The model must also be capable of running for long periods of time in order to simulate a real life test. The major difficulty with monolayers is that the long term performance is very unpredictable and this makes it difficult to perform large scale experimental trials. A computer model can be used to gain an insight into the behaviour of monolayers that cannot be gained with traditional experimental trials.

The model must also deal with dynamic weather conditions consisting of changing wind speeds and directions. It should be able to accept data in the same form as it is collected by weather stations. This will make it easy to assess long term effects for

different locations. The model should also adequately represent the behaviour of the monolayer by incorporating all the major effects that affect the performance of the monolayer. Some of the effects that affect the monolayer have not yet be quantified by experiment or computer model and therefore cannot be included in this current model, but the program must be able to include these effects when they have been quantified with minimal effort.

In summary, the main requirements of the model are:

- Not limited to specific monolayer
- Capable of modelling long periods of time
- Incorporate dynamic wind conditions
- Represent behaviour of monolayer
- Behaviour that has not yet been quantified can be easily added in the future.

3.2 Methodology

3.2.1 Overview of model

The model has been written in MATLAB which has excellent graphing facilities. This will make it easy to visualise the results of the simulation. The model will attempt to optimise the application rates of the applicators that are located on the dam.

At the beginning of each timestep, the wind conditions will determine the drift speed and angle of spread of the monolayer. Particles will then be created at the applicator locations to represent the continuous application of monolayer during the timestep. The dam will be divided into small rectangles, $dx \times dy$. If the number of particles is sufficiently large enough for the size of this rectangle, then a continuous film can be modelled as a large number of discrete particles. Each particle will be created at a random fraction of the timestep, as well as at a random angle of spread.

After several iterations of creating particles from the applicators, it will be necessary to calculate average values for several important variables. These variables include: mass, concentration, monolayer resistance and age of particles. The critical variable will be the monolayer resistance which will be used to calculate the evaporation for each area $dx \times dy$ in order to determine how effective the monolayer is on the dam. The difference in evaporation rates with and without monolayer present will then be calculated. By assigning a value to the water, a dollar saving can be found. The profit gained by applying the monolayer will be calculated by subtracting the cost of applying the monolayer to the dam from the value of the water saved from evaporation.

The highest profit for different combinations of application rates will be the optimal combination of applicator rates. These above steps are repeated as the wind changes. The pre-existing monolayer on the surface of the dam will also be modelled using experimental data to describe the the different types of behaviour of the monolayer.

3.2.2 Sections of the program

The program has been broken into several functions and scripts. Each of these functions and scripts has a specific purpose. Each function and script will be explained in greater detail.

3.3 Equations and essential features of the model

3.3.1 Movement of monolayer

The monolayer has been modelled as particles. There are two terms that describe their behaviour: a stochastic and a deterministic term. The stochastic term represents the random nature of the particles. When the number of particles in a given area is high enough, the effect of using random particles ceases to impact the results and the simulation (with a finite number of particles) becomes a good approximation of the continuous monolayer film. The deterministic term is from the experimental results which describe the drift speed and the angle of attack of the monolayer.

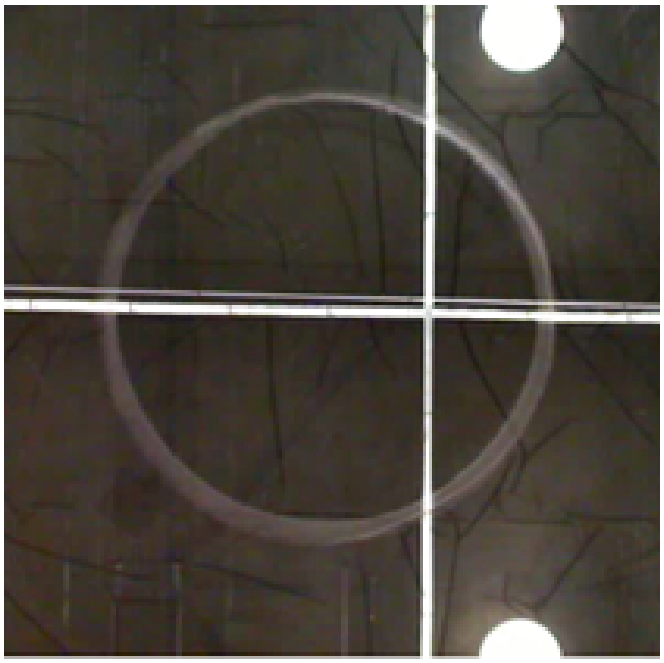


Figure 3.1: The shape of the monolayer with no wind. Talcum power is used to show the edge of the monolayer as it spread from the centre where it was initially applied. Reproduced from Brink (2011, p. 69).

The movement of the monolayer is characterised based on the angle of spread and the drift speed. The shape of the monolayer is dependent of the wind speed. At wind speeds lower than 3.2 km/h the monolayer spreads in a circle (see Figure 3.1) and at higher wind speeds is a sector of a circle (see Figure 3.2).

3.3.2 Evaporation resistance of monolayer as a function of wind speed

The resistance of the monolayer depends not only on the concentration of the monolayer (mg/m^2), but also on the wind speed. Figure 3.3 shows the relationship between wind speed and evaporation resistance can be treated as linear due to lack of available data. It can be argued that the linear model is an average, since some parts of the monolayer collapse and others are exposed due to the effect of the waves at higher wind speeds (McJannet, Knight, Cook and Burn 2008). The model represented in the figure places three constraints on the model:

- Speed $< U_{max}$

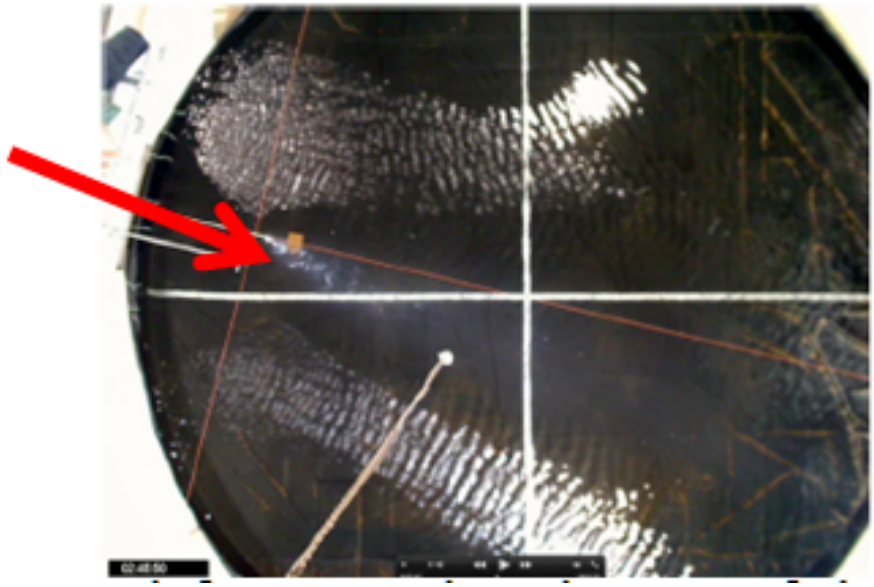


Figure 3.2: The shape of the monolayer with a wind speed of 4.5 m/s. The monolayer is continuously introduced from the left and forms a wedge shape. The monolayer is present where there are no light reflections, which indicate waves. Reproduced from Brink (2011, p. 100).

- Concentration < 2 x minimum theoretical amount to form a valid monolayer
- Equation of the straight line

The concentration is limited to at least 2 x the minimum theoretical amount to ensure sufficient monolayer product is present (within each cell dx dy) to guarantee the monolayer will inhibit evaporation.

3.3.3 Volatilisation

This is the evaporation of the monolayer. The fractional loss (degradation of mass due to evaporation) is calculated using the

$$\text{dme} = \text{dt} \times 0.073957 \times e^{0.14\text{Temp}} \times 10^{-6} \tag{3.1}$$

where dt is the time step [s] and Temp is the water temperature [°C]. Appendix B has further details.

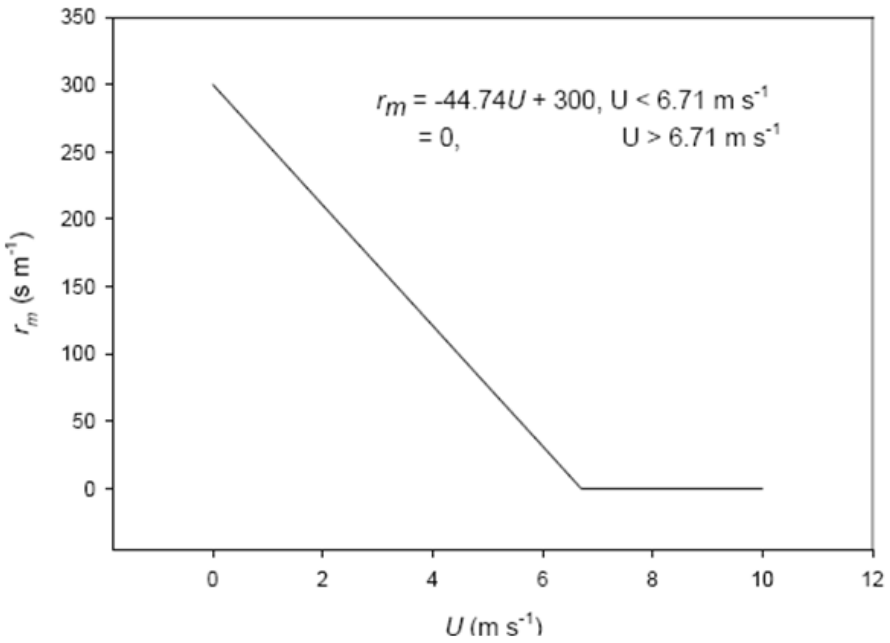


Figure 3.3: The effect of wind speed on the evaporation resistance. Reproduced from McJannet et al. (2008).

3.3.4 Degradation due to biological attack

The monolayer degrades due to algae and bacteria on the water surface. This is modelled by reducing the amount of effective mass of monolayer that is on the water surface as the monolayer ages. The monolayer has a typical life of 8 days before it has completely degraded. See Appendix B for complete calculations.

3.3.5 Shoreline interaction

The effect of the shoreline will need experimental or CFD modelling to account for the losses. See Chapter 7: Further Work.

3.3.6 Monolayer submergence

A proportion of the monolayer will be lost due to the action of the waves at high wind speeds. This effect can be modelled by considering that a proportion of the effective

mass is lost (i. e. the model only keeps track of the effective mass of monolayer). There has been no experimental results to date so this component will need to added to the model in the future in order to account for this effect. See Chapter 7: Further Work.

3.4 Resource Requirements

The model will create particles from the applicators. As time progresses, the number of particles could cause the computer to use all the available Random-access memory (RAM) and will use the hard drive instead. This will cause significant slowdowns. There are two ways to solve this problem:

- Use a computer with more RAM
- Write code to reduce the number of particles

The first solution will work up to a point. There will however be a maximum number of particles that can fit into the RAM. There are several factors that influence the number of particles in the simulation after a long period of time: the current time; the size of the domain compared to dx and dy; and the number of particles created at each time step. After long periods of time it may be difficult to ensure there is sufficient RAM available while maintaining an acceptable level of accuracy.

An additional function could be written to reduce the density of the particles if they exceed a set limit by either removing particles with very low mass compared to the other particles, or combining particles that are very close together. This may be necessary if the domain is very large and/or the period of time is very high. This is preferable to simply using a computer which has more RAM as it reduces the number of computers that can run the model and will have a limit on the complexity of the problem that can be successfully modelled.

3.5 Consequential Effects

The model can be used to further develop control mechanisms for the application of monolayer to large dams with complex shorelines. The two aspects that the model can help optimise are the application rates of the individual applicators and the location of the applicators. It is important that the model can be used for large lams and for long periods of time. Once the model has sufficient detail to represent the real situation it can be validated by comparing the results to real experiments.

3.5.1 Current Model

The current model can handle multiple applicators with different application rates within a rectangular boundary. The model needs to be expanded to include as many of the different monolayer behaviours with sufficient depth to ensure the model represents real experiments. It also needs to cope with non-rectangular boundaries.

3.6 Conclusion

This chapter introduced the requirements for the model and some the behaviours that have been modelled to get a performance indicator of the monolayer for use in evaporation reduction. The fundamental goals and the methodology to achieve these goals was also discussed.

Chapter 4

Detailed Design

4.1 Introduction

This chapter explains the each function and script used in the program as well as methodology to use to extend the model.

4.2 Overall Plan

The model will use an explicit solution. This means the model will need to satisfy the Courant-Friedrichs–Lewy condition which ensures that particles do not move more than one grid square per time step in order to ensure stability of the system. It specifies the maximum time step that can be used given a speed and grid size. The equation for the one dimensional case is:

$$\frac{U_L \Delta t}{\Delta L} \leq C_{max}$$

where U_L is the speed in the L direction, Δt is the time step and C_{max} is 1 for the explicit method. The CFL condition must be satisfied for each dimension since the grid is two dimensional. The limiting case will be either in the x or the y direction. This is because the limit for the CFL condition requires the use of the larger of either $\frac{U_x}{\Delta L_x}$ or

$\frac{U_y}{\Delta L_y}$. Rearranging for Δt yields:

$$\Delta t \leq \frac{C_{max}\Delta L}{U_L}$$

4.2.1 Movement of particles

The monolayer is a continuous one molecule thick film. One approach to represent the film in a computer program is through the use of discrete particles. The movement of the monolayer is stochastic, that is it has a deterministic and random element. The deterministic element is the equation which describe the speed and angle of spread. The random element is the use of random numbers to generate particle locations which are required to approximate a continuous film using discrete particles without bias.

There are two different ways that the particles move in the model: creation of particles from the applicators, and movement thereafter. The particles originating from the applicators are specified by a random radial position and a random angle. The radial position is between zero and the smaller of dx or dy, while the angle is a combination of the wind direction and a random angle within the angle of spread, which is based on the wind speed.

4.3 Script and function design

4.3.1 Representing monolayer as particles

The monolayer film is represented as a large number of discrete particles. The number of particles needs to be large enough so that the individual position of each particle is not important and only the overall properties of a group of particles. The particles are created in a rectangle at the applicator with dimensions dx by dy.

Iteration	Applicator		
	1	2	3
1	1:50	51:100	101:150
2	151:200	201:250	251:300
3	301:350	351:400	401:450

Table 4.1: Values for indexing, where np_cell = 50

4.3.2 Indexing matrices

It is necessary to specify which values of the matrix will be modified to prevent dynamic memory allocation occurring. Dynamic memory allocation is significantly (10 to 100 times) slower than using preallocated matrices. This time penalty is unacceptable when dealing with large arrays that are modified often as is the case with the majority of the variables that the model calculates. The index has two components: a start and a final value. There are two different indices required to move the particles.

The old particles start a one and finish at the last particle that has been created. The value of this last particle is given by the following expression, which was found by considering the pattern in Table 4.2:

$$\text{im3} = (cc - 1) \times n\_app \times np\_cell + k \times np\_cell + tc \times n\_app \times np\_cell \tag{4.1}$$

where cc is the current counter (number of iterations), tc is the total counter (sum of all previous current counters/iterations), n_app is the number of applicators and np_cell is the number of particles in a cell (dx by dy). The expression for indices for the random angles (theta old) in the cosine and sine functions are:

$$to(1 : im3) = windDirection(windloop) + (0.5 - rand(1, im3)) \times angleSpread \tag{4.2}$$

The new particles indices were also found by considering a similar pattern. The following two expressions are for the start and final index respectively:

$$lhs = 1 + (cc - 1) \times n_app \times np\_cell + (k - 1) \times np\_cell + tc \times n\_app \times np\_cell \tag{4.3}$$

$$rhs = (cc - 1) \times n\_app \times np\_cell + k \times np_cell + tc \times n\_app \times np\_cell \tag{4.4}$$

Iteration (cc)	Applicator	
	1	2
1	np_cell	k × np_cell
2	(cc-1) × np_cell × n_app	(cc-1) × np_cell × n_app + k × np_cell
3	(cc-1) × np_cell × n_app + k × np_cell + tc × n_app × np_cell	(cc-1) × np_cell × n_app + k × np_cell + tc × n_app × np_cell

Table 4.2: Equations for indexing old particles which were used to find $im3$

where k is the index of the applicator where the particles are being created from. The expression for indices for the random angles (theta new) in the cosine and sine functions are:

$$tn(1:np\_cell) = windDirection(windloop) + (0.5 - rand(1, np\_cell)) \times angleSpread \tag{4.5}$$

4.3.3 Finding permutations of applicator rates and applicators

It is necessary to use an array of all possible applicator rates as an input to the model. The model selects the most profitable combination of application rates based on the money saved. The model needs all possible combinations in order to select the most profitable combination. A function developed by Matt Fig and available from the Mathworks file exchange was used since there was no suitable inbuilt MATLAB function. The function is available from `http://www.mathworks.com/matlabcentral/fileexchange/11462`. The function `npermutek.m` takes two arguments: an array of rates and the number of applicators. For example, the function outputs the following matrix given the rates as [0 1] and the number of applicators as three. Each row is a different permutation and the columns are the individual applicators.

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

It can be seen in the above matrix, the number of combinations to test grows very quickly with increasing number of applicators. The next section will discuss how the number of permutations to test can be reduced given the right wind conditions.

4.3.4 Calculating application rate for different wind speeds

There are several different methods of calculating the application rates. The current model only considers two different rates due to the large number of permutations that would result otherwise. One of these rates must be zero, since at high wind speeds, the product will either be lost to submergence or a large proportion will be washed ashore. The other rate will be the highest of the rates calculated by previous researchers because it will ensure that a film is formed. Higher concentrations of monolayer have also been shown to reduce the amplitude of waves formed by the wind, reducing the amount of monolayer that is lost to submergence.

Brink (2011) identified several different equations developed by different researchers to find the application rate as shown in Figure 4.1. The application rate is a function of the wind speed. The most important factor in determining application rates is to consider "... the loss by wave action and submergence. Since the film can only be effective if it exists at the surface layer on top of the water, it is necessary to add sufficient material to dampen the waves and prevent 'drowning' of the alcohol by wave action" (Reiser cited in Brink 2011). Reiser found that at up to a wind speed of 27 mph it is possible to maintain a smooth surface by applying a sufficient amount of monolayer. At wind

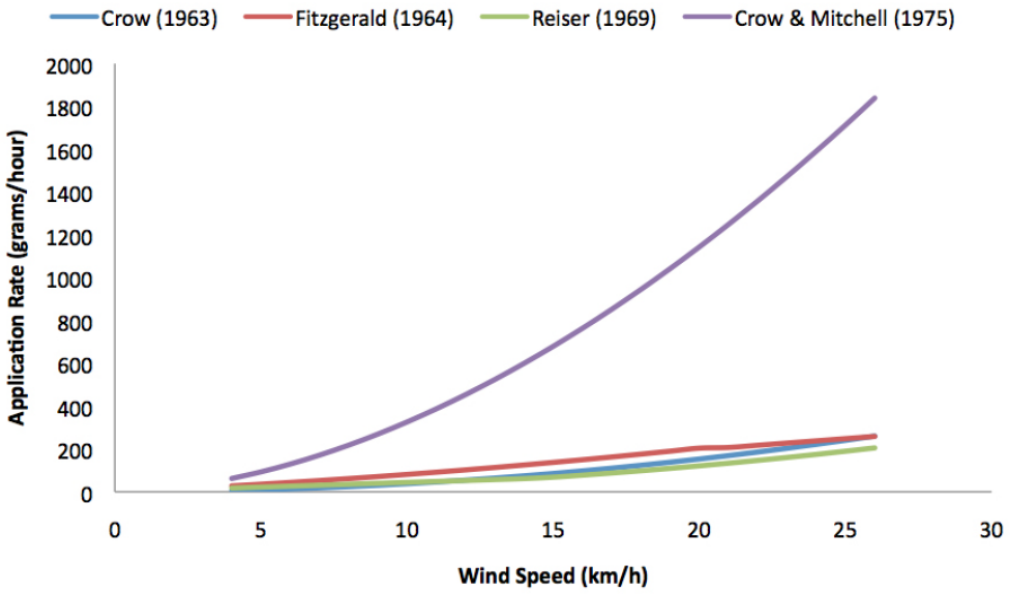


Figure 4.1: Application rates as a function of wind speed calculated using three different equations based on an area of 1 ha. Reproduced from Brink (2011).

speeds above 9 mph the suppression of the waves is important by ensuring there is enough monolayer present to dampen the waves the reduce the amount of monolayer that is submerged.

Reiser determined an equation the application rate [lb / h] for each range of wind speeds. At less than 9 mph:

$$\frac{\text{lb / h}}{\text{100 ft of shore}} = 2.9 \times 10^{-5} VM \tag{4.6}$$

and at wind speeds above 9 mph:

$$\frac{\text{lb / h}}{\text{100 ft of shore}} = 3.2 \times 10^{-5} V^2 M \tag{4.7}$$

where V is the wind velocity [mph] and M is the molecular weight of the monolayer per hydrophilic group.

Crow found an equation based on experiments on two ponds (30.5 x 36.6 x 2.1 m). The application rate is the minimum amount needed to maintain a continuous film on the surface at various wind speeds. Crow's empirical equation is:

$$R = 9.3U^{2.02} \times 10^{-6} \tag{4.8}$$

where R is the application rate [lb/h/ft of distribution line] and U is the wind speed [mph].

Fitzgerald's equation is based purely on the monolayer drift. The ratio of monolayer speed to wind speed starts at 0.03 and rises to 0.045 at wind speed 19.8 km/h and higher in a linear manner. This shows that Crow and Reiser's equation only consider the effect of the monolayer drift, which is not enough to ensure that enough monolayer is applied, as none of the forms of degradation are considered.

Crow and Mitchell's equation for the application rates is:

$$R = 1.18U^{1.81} \times 10^{-4} \tag{4.9}$$

where R is the rate [lb/hr/ft of distribution line] and U is the wind speed [mph]. Crow and Mitchell's equation has been expressed in SI units in the following equation:

$$R = \frac{dist\_line * 3.2808399 * 453.592}{3600} * 1.18 * \left(\frac{3600}{1609.344} * uWind\right)^{1.81} * 10^{-4} \tag{4.10}$$

where dist_line is the length of the equivalent distribution line [m], and $uWind(windloop)$ is the wind speed [m/s]. Crow and Mitchell's equation has been used for the model currently as the equation was developed from an experiment on a large lake, which is the only size of water body where monolayers are economically viable. Crow and Mitchell (1975) found an application rate 6.5 to 8 times greater than previously reported using a similar pond and application system. During the trial on Lake Hefner in Oklahoma in 1965-66 evaporation savings of 11.5% were found. This is significantly less than than expected from experiments on smaller ponds and it is due to "serious operation problems caused by wind, which proved to be the major obstacle to maintaining a continuous film covering the lake" (Crow & Mitchell, 1975 p. 493).

Brink (2011, p. 31) explains the reasons for the discrepancies between Crow and Mitchell's results and Crow and Reiser's equations. These reasons are:

- Wind speed measured at 2 m, where it is 28% higher than on the water surface.

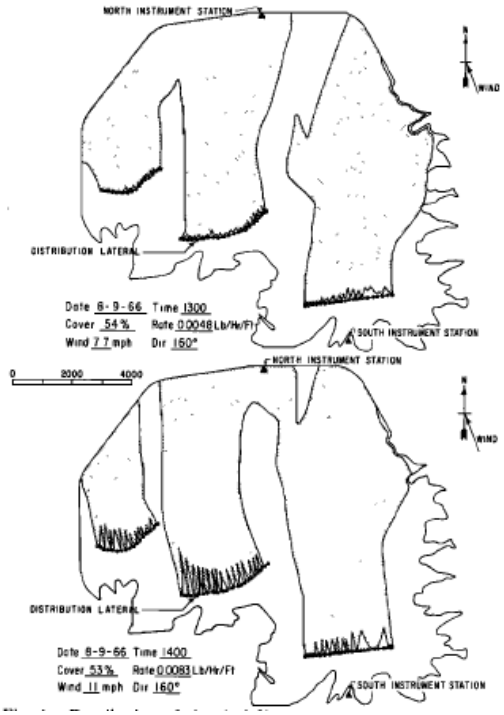


Figure 4.2: Coverage map for Crow and Mitchell's experiments on Lake Hefner. The dark lines represent the distribution lines where the monolayer was applied. Reproduced from Crow and Mitchell (1975).

- Large distance (234.4 m) between shore and applicator means that additional monolayer was required to suppress capillary wave action between the shore and the applicator, see Fig 4.2.
- Spacing between applicators caused the monolayer applied from separate applicators to merge.

The monolayer merging from different individual applicators can be included in the model by simulating multiple application rates (in addition to different permutation of individual applicators) and selecting the most profitable for each time step. The current model only includes a single application rate (Crow and Mitchell 1975) because not all the different forms of degradation have been accounted for, (namely submergence and shoreline interaction,) and this would effect which application rates are used.

4.3.5 Reducing the number of rates permutations to test

The number of permutations of application rates that need to be tested is m^n where m is the number of different rates and n is the number of applicators. There are only two different rates that are tested in the model: zero and a rate found using Equation 4.10, Crow & Mitchells equation. The number of permutations, even for only two rates, is very large. For five applicators, there are $2^5 = 32$ different permutations to test. This number can be greatly reduced by considering the proportion of monolayer that each applicator applies that will remain on the water, and not leave the domain. If the proportion of monolayer applied from an applicator that will cover the water inside the boundary is low compared to the total area covered, it is unlikely to be profitable to utilise that applicator for those conditions and it is probably not worth the extra computational time to solve.

Four different conditions were identified which were used to identify when particular applicators were not suited to the current weather conditions:

- Angle of spread equals 2π
- One Out
- Both Out
- Corner

The logic diagram in Figure 4.3 shows how the four cases can be identified. This diagram can be written as pseudo MATLAB code:

```
if angleSpread == 2*pi
     (1)
else if P1 && P2 out
     (2)
   else
     if sum( [out_xmax out_xmin out_ymax out_ymin] ) == 1
     (3)
     else
```

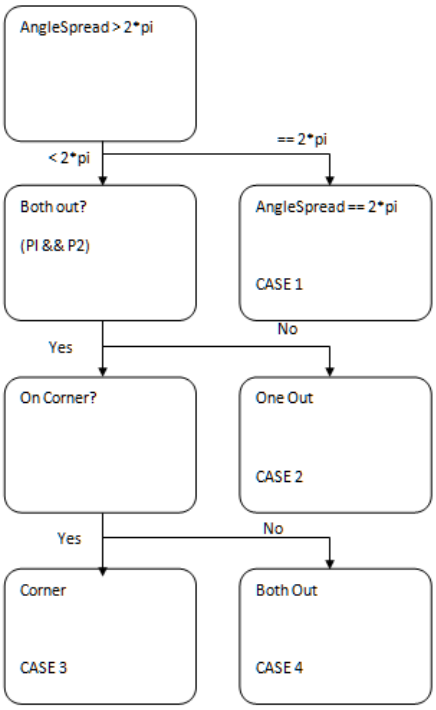


Figure 4.3: Logic diagram to discriminate between different calcArea cases.

```
        ④
        end
    end
end
```

The figures for the first two cases, CalcArea1 and CalcArea2, correspond to the numbering in the source code. See Appendix C: Source code for these two cases.

Case 1

The first case is when the angle of spread is 2π, which occurs when the wind speed is below 3.2 km/h. There are eight possible sub-cases – either on one of the four corners or on one of the four boundaries. The area outside the boundary is calculated using the symbolic toolbox in MATLAB. The double integral is found using the intersection of the circle with boundaries as the limits of integration. The four extremes of the circle

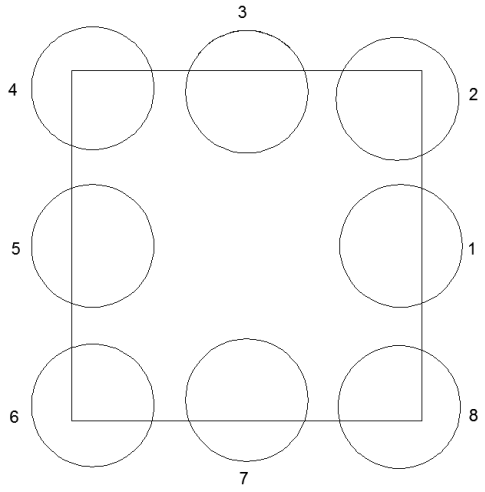


Figure 4.4: calcArea 1.

$(0, \frac{\pi}{2}, \pi and \frac{2\pi}{3})$ are used to identify the method to use to find the integral and which limits to use. The equation of the circle is broken into two parts: the upper and lower section. Each section is represented symbolically in the form of:

$$\text{eqyp} = b + sqrt(r^2 - (x - a)^2);$$

$$\text{eqym} = b - sqrt(r^2 - (x - a)^2);$$

where eqyp is the equation of y for the positive (upper) section and eqym is the negative (lower) section. A switch statement is used to select the case that was identified previously and the double integral is found to calculate the area. Where the circle is on the corner of the boundaries, two double integrals are evaluated. The limits for the integration where determined using Figure 4.4.

Case 2

The second case is when one of the two points that represent the extremes of the angle of spread at the edge of the monolayer (at the maximum calculated distance) is outside the boundary. Figure 4.5 shows all 16 cases. The first point is the point at the end of the straight line inside the boundary, and the second point is numbered on the figure. The method is similar to the previous case but instead of comparing the extremes of the circle to the boundary; this script compares the position of the two extremes of

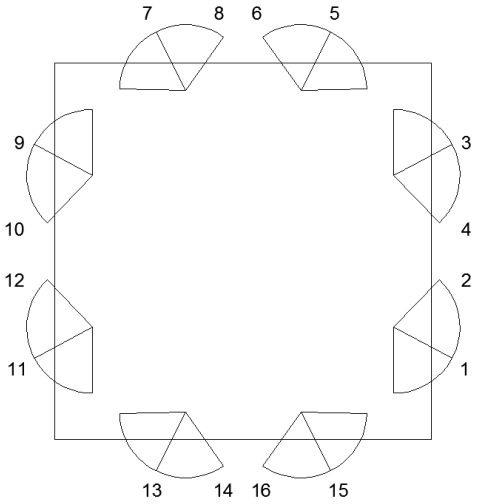


Figure 4.5: calcArea 2.

the angle of spread at the maximum fetch (the maximum distance that the monolayer travels) to the boundaries.

Case 3

The third case is when both points are outside the boundary, but only one boundary and not over a corner. The area inside the boundary is simply a triangle and is found with the following equation:

$$\text{areaInside} = \sqrt{s(s-a)(s-b)(s-c)} \tag{4.11}$$

where a, b and c are the three lengths of the triangle and s is the semi-perimeter, $s = \frac{1}{2}(a+b+c)$. The lengths of the triangle are found by solving the equations of the two lines which define the two extremes of the angle of spread and the boundary.

Case 4

The fourth case is when the monolayer extends over one of the four corners of the boundary. The area is calculated using the quadrilateral formula:

$$\text{areaInside} = \frac{1}{2}|\vec{p} \times \vec{q}| \tag{4.12}$$

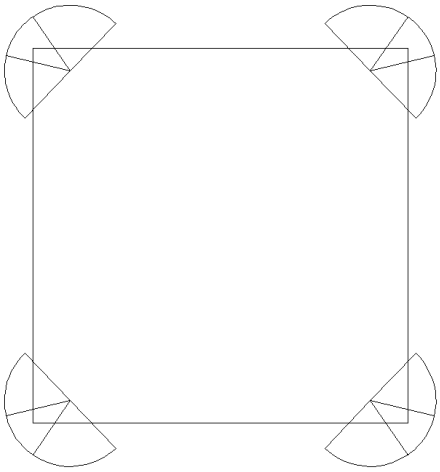


Figure 4.6: calcArea 3.

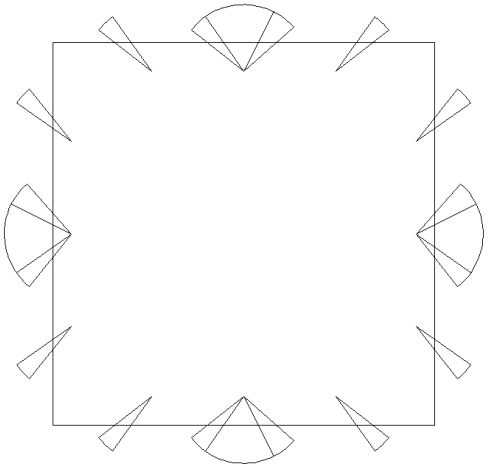


Figure 4.7: calcArea 4.

where p and q are the diagonals of the quadrilateral. The diagonals are defined in terms of the four vectors that form the sides of the quadrilateral. If the vectors are arranged so that $\vec{a} + \vec{b} + \vec{c} + \vec{d} = 0$, then $\vec{p} = \vec{b} + \vec{c}$ and $\vec{q} = \vec{a} + \vec{b}$. The corner where the product crosses the boundary can be found by comparing the x and y distances of the applicator from the upper and lower x and y boundaries. After the correct corner has been identified, the intercepts of the lines that define the extremes of the angle of spread with the boundary can be calculated. These points are used to find the four vectors $\vec{a}, \vec{b}, \vec{c}$ and $\vec{d}$ that define the quadrilateral and to calculate the area using Equation (4.12).

Reducing CPU time by simulating fewer permutations

If the area covered inside the boundary for a given applicator is less than the user-defined proportion (currently set at 0.50), then the applicator is declared unfavourable. This means it is probably not profitable and therefore shouldn't be operated. It is therefore necessary to modify the original rates matrix to reflect this by removing the column (where the entires are zero) that corresponds to the unfavourable applicator. This modified rates matrix can now be used to find the rates of each individual applicator that is worth modelling. This modified rates matrix can be significantly smaller than the original matrix where all applicators are considered. This has the potential (depending of wind vector and applicator position relative to the boundary) to save a large proportion of the central processing unit (CPU) time.

4.3.6 Finding the average values for each grid cell

For each different weather condition it is necessary to check how effective each permutation of applicator rates was at maximising the profit. This is achieved by finding an average value for each grid cell. The first step is to find which cells the particles are in. After the cells have been identified, the average mass, concentration, and age can be calculated. The main purpose of finding the average of the cells is to determine the average resistance that the monolayer provides for each cell.

4.3.7 Calculating the evaporation

The overall aim of the model is to find the evaporation with and without the monolayer. In order to find the evaporation with the monolayer, the resistance that the monolayer provides is required. After this resistance has been found, the evaporation can be calculated. The full details of the equation are in Appendix E.

4.3.8 Selecting an appropriate time step for each wind vector input

The time between wind inputs can have a large effect on the results that the model outputs. It is necessary to ensure that the time step is small enough to ensure that low to moderate wind speeds will not cause the majority of the monolayer to be applied outside the boundary. If the timestep is too large and the user defined proportion of area inside the boundary is too high, the model may not apply any product to the surface. A small timestep will not have a negative effect on the model.

4.4 Conclusion

This chapter outlined the reasons behind the code and how it was developed. The important sections of this chapter are:

- indexing particle locations in memory,
- finding the application rate for different wind speeds,
- reducing the number of combinations of rates to test by considering the area covered by each applicator,
- and using the average values for each cell to calculate the evaporation.

Chapter 5

Sample Run

5.1 Introduction

This chapter describes the program that implements the model. The program has been separated into several different scripts that each perform a component of the program. The overall objective of the program is to select the most profitable permutation of applicator rates. The major components of this objective are:

- Read variables and constants.
- Allocate space for large matrices.
- Calculate array of permutations application rates for individual applicators.
- Add particles at applicator locations; calculate position and mass of individual particles move due to wind.
- Find the average mass for each grid cell after a predetermined amount of time (e. g. time between wind vector readings).
- Use average mass to find the evaporation rates.
- Calculate the amount of money saved.
- Determine most profitable permutation of applicator rates for each wind vector.

5.2 Sample run of model

5.2.1 config. m

The main script is `Monolayer_Simulation.m`. The first script that is called is `config.m`. This script configures the program. It is where all the important variables are modified.

UWINDMIN Minimum wind speed before shape of monolayer is sector of circle and not circular [m/s]

time_combo Time between weather data readings [s]

np_cell Number of particles per cell (dx by dy)

dx, dy Grid spacing in the x & y directions [m]

x_max,x_min,y_max,y_min Limits for rectangular boundary [m]

uWind Row vector of wind speeds [m/s]

windDirection Row vector of wind directions measured from the positive x axis in an anti-clockwise direction [rad]

Temp Temperature for calculation of evaporation of monolayer, dme in `degradation.m`

applicators The location of the applicators $[x_1, y_1; x_2, y_2]$ etc.

n_app The number of applicators

5.2.2 preallocate. m

The next script is `preallocate.m`. This script file preallocates space for the large arrays to decrease the time spent dynamically allocating memory as the size of the arrays change.

nX_max,nY_max,nX_min,nY_min Limits of boundary in number of nodes based on boundary size and grid spacing.

tc Total counter

cc Combination counter

totalmass Total mass applied during simulation [g]

P_SIZE The size of all the large arrays: X,Y,mass,particle active flag

The loop on lines 16-23 finds the value of P_SIZE. The size of the arrays was calculated by realising that for each *time* step (as defined in `cfl.m`) the maximum number of particles created is the product of the particles per cell and the total number of applicators. The `while time < time_combo` loop finds the number of particles created for a single wind vector. The `for windloop = 1:length(uWind)` loop finds the total number of particles for all the wind vectors.

The remaining code preallocates the X,Y, mass, age, partact, to,tn,dmb and dm arrays using the value of P_SIZE as the limit of the array. The reason for preallocating memory for these matrices is to prevent dynamic memory allocation and is explained further in Section 4.3.2 Indexing matrices.

5.2.3 Monolayer_Simulation. m

Control returns to the `Monolayer_Simulation.m` script where the program enters the outer for loop, for windloop = 1:size(uWind,2) which loops through all the wind vectors, selecting the optimal combination of rates for each.

s=rng('shuffle'); This re-seeds the random number generator to ensure that different random numbers are used for each wind vector.

5.2.4 calcrates. m

Control passes to the `calcrates.m` script to find the number of permutations that need to be evaluated in the following `while combo <= combo_max` loop. The `calcrates.m` script contains alot of code that is aimed at reducing the number of permutations

that need to be evaluated by checking if the proportion of area of monolayer applied from each applicator is above a user defined constant, `PROPINSIDE`. If the proportion of area inside the boundary is low the effect that the monolayer will have on the overall evaporation resistance will be low and therefore that particular case is not worth evaluating. The area is calculated based on the position of the two points that define the furtherest fetch and the two extremes of the angle of spread in relation to the boundary.

There are four cases. See Section 4.3.5 for further details and Appendix C: Source Code for the complete source code.

5.2.5 Monolayer_Simulation. m

Control passes back to `Monolayer_Simulation.m` where the mass_app variable is set to zeros. The combo_max variable contains the number of different combinations that need to be simulated, which is `size(rates,1)`, where `rates` was found in `calcrates.m`. The `bestprofit` variable is initialsed with a large negative number to ensure it isn't used. The following while loop is where the majority of the time is spent.

```
combo = 1
while combo <= combo_max
```

5.2.6 calcmoveevapdollar. m

Control passes to `calcmoveevapdollar.m` which calculates the movement of the particles, the evaporation with and without the particles and finally the dollar saving associated with the application of the monolayer for a given combination of applicator rates. The `calcmoveevapdollar.m` script loads the the random number seed, `rng(s)`. The 'current' arrays for X,Y,mass, pact and totalmass are assigned to X,Y,mass, pact and totalmass. The 'current' arrays store the optimal state of the simulation so far. The next step is to set the current counter `cc` and the time to zero. The `time` variable only stores the time within the `while combo <= combo_max` loop. The global time is

called `time_overall`. The next loop in the `calcmoveevapdollar.m` is very important since it is where the position, mass and age of the particles is modified. The loop is:

```
while time < time_combo
cc = cc+1;
cfl
movepart
end
```

5.2.7 cfl. m

The scripts `cfl.m` and `movepart.m` are described below. `cfl.m` finds the `uDrift` and `angleSpread` given the wind vector (speed and direction), `time` and `time_combo`. The script first checks whether the wind speed is below the minimum wind speed to form a sector of a circle shape. If the wind speed is below, the shape is circular and is approximated using an experimental function based on the experiments conducted by Brink (2011 p. 77). Otherwise the drift speed is a function of the wind speed and the shape is a sector of a circle.

The following is the same for both shapes.The value of `courantn` must be less than 1 to ensure that the model is stable. The variable represents the Courant number. The definition of the Courant number is rearranged to find an expression for the value of dt.

Both shapes then compare the sum of dt and the current value of time to the time_combo. If the sum is large, dt is set as the difference between time_combo and time.

The angle of spread for the circular shape is 2 π and for the sector of a circle it is an equation based on fitting a trend line to the data Brink collected from the experiments conducted.

5.2.8 movepart. m

This is the most important script in the model as it modifies the position of the particles and also calculates the mass of the particles due to different types of degradation. The script starts by finding the index for the old particles, which is

$$(cc-1)*n\_app*np\_cell+k*np\_cell+tc*n\_app*np\_cell \tag{5.1}$$

where k = n_app for the above equation. The degradation is found next in `degradation.m`.

5.2.9 degradation. m

The degradation of the monolayer is expressed as a fraction, where 1 is all lost and 0 is no loss. The evaporation of the monolayer is found using Equation 3.1 described in Section 3.3.3. The degradation as a result of biological attack is function of age and is found using Equation B.2 in Section B.3. The sum of all the different forms of degradation is called `dm` and is the output of this script.

5.2.10 movepart. m

The movement of the particles can be broken down into two parts: old particles and new particles. Old particles are particles which were created in a previous time step while new particles are created in the current time step. The test for old particles is `cc > 1 || tc =0`. The old particles are moved if either the current counter is greater than one or the total counter is not zero. The new position of the old particles is:

$$\text{new position} = \text{old position} + \text{pact.} * \text{rand} * \text{dt} * \text{speed} * \text{trig(rand within wind + spread)} \tag{5.2}$$

where pact is the particle active flag (modified in `boundary.m`), rand is a random number between 0 and 1, speed is `uDrift`, and trig() is the cosine and sine function for the X and Y coordinates respectively. The age of the particles is the old age plus dt, and the mass is the old mass times (1-dm) where dm is fractional degradation found in `degradation.m`.

The new particles X and Y position is equal to:

$$\text{position} = \text{applicator position} + \text{rand} * \text{dt} * \text{speed} * \text{trig}(\text{rand within wind} + \text{spread}) \tag{5.3}$$

The age of the particles is `dt` and the mass is:

$$\left(\frac{dt * rates(combo, k)}{np\_cell}\right) \times (1 - dm) \tag{5.4}$$

The total mass is the sum of all the masses applied:

$$totalmass = totalmass + (dt * rates(combo, k)/np\_cell)$$

After the old and new particles positions and properties (mass, age) have been updated, the `boundary.m` script is executed.

5.2.11 boundary. m

This script checks if each particle is inside the boundary. If it is not, the pact flag (particle active) is set to false. The index required is the same as the index for the old particles, Equation 5.1. The logic for the script is as follows: the particle is active (true) if it is inside both the x and y boundaries. The particle is inside the boundary if it is above the minimum and below the maximum.

5.2.12 calcevapdollar. m

Control passes back to `calcevapdollar.m` after the while time ¡ time_combo loop ends, where the `avg.m` script is executed. This script finds the average mass, age and evaporation resistance.

5.2.13 avg. m

The script starts by finding the nodes that the particles (X,Y) are closest to denoted by nX and nY. The unique values of nX and nY are grouped together into a matrix by the

MATLAB function unique. An array for the average evaporation resistance, average mass, average age, average concentration and an index array, cond is preallocated.

A for loop is entered, starting at one and ending at the last unique combination of nX and nY values. The variable `cond` contains a logical array used to identify the particles that are within each node (nX, nY). This is used to find an average mass for each node. This average mass is divided by the area of the node to find the concentration. The average age is a weighted average of the age using the average mass. The most important quantity that is found using the `avg.m` script is the average evaporation for each node. If the wind speed is greater than 6.71 m/s and the concentartion is greater than twice the theoretical concentration, then it is highly probable that a monolayer film has formed that will provide some resistance to the evaporation. The amount of resistance offered by the monolayer is dependant on the wind speed.

5.2.14 calcmoveevapdollar. m

Control passes back to the `calcmoveevapdollar.m` script where the evaporation is found in the `evap.m` function and the dollar saving in calculated in `dollarsaved.m`.

5.2.15 evap. m

The first section declares all the constants required for the equations. Some of the constants can be replaced as variables. For example, the radiation, K_down needs to be a function of the the day of the year, J. The equations are from the modified Penman–Montheith equation developed by McJannet et al. (2008). The difference in evaporation is calculated by taking the difference between the evaporation calculated with and without the monolayer present.

5.2.16 dollarsaved. m

This script finds the profit arising from applying the monolayer for a particular permutation of applicator rates. The amount of water saved is calculated from the area of the

water, time and the difference in evaporation found in the `evap.m` function. The saving is the value of the water per cubic meter times the amount of water saved in cubic metres. The cost is the amount of monolayer applied ($/g) plus a running cost. The running cost includes everything other than the cost of buying the monolayer. This is the cost of buying, installing and maintaining the applicators as well as refilling them. The profit is the difference between the cost and the savings.

5.2.17 Monolayer_Simulation. m

After the profit has been found for each combination, the profit is compared to the `bestprofit` so far for the same wind conditions. The `bestprofit` variable is initialised with a large negative number to ensure that it isn't used. If a combination is either the first or more profitable than the previous best, all the key variables are stored in variables with the same name but with a p suffix. The counter for the combination is incremented. After all combinations have been simulated for a given weather condition, the most profitable combination with variables with a suffix p are saved as the current variables. These are the base conditions that all future combinations start at, and are denoted by variables with a c suffix.

After all the wind conditions have been simulated, the best profit for each wind condition is selected. The total profit is the sum of of the best profits for each wind condition.

Any required variables can be saved and graphed. There are two graphs that show how effective the monolayer is at reducing evaporation. The first is a xy scatter of the position of the particles and the second is a contour plot showing the average evaporation for each node. A video of the particles movement is made by taking frames of xy scatter plots over time to show how the particles move in the model.

5.3 Conclusion

This chapter explained how the code is executed. The complete code can be found in Appendix C: Source Code. The equations used to write the `evap.m` script are reproduced in Appendix E: Evaporation Model Equations.

Chapter 6

Results and discussion

6.1 Introduction

This chapter describes how the model is validated by performing a convergence test. This is necessary to ensure that the results obtained from the model are correct and can be trusted. A convergence test is used to compare the results of different realisations and to see the amount of variation that exists between each realisation. As the number of particles per cell, np_cell, increases and grid spacing , $dxdy$, decreases the model will approach the real value.

6.2 Code validation

The random nature of the model can cause different realisations (result of executing the program with the same input but different random numbers due to different seeds for the random number generator function) to produce different results. This limitation has been overcome by plotting the profit vs weather reading for different realisations. When the difference between the realisations is 'small' then the result is independent of the random number generator seed and the model has converged on a solution. The model will give acceptable values when the combination of grid spacing and number of particles per cell exceed a certain number of particles per unit area. The results of the

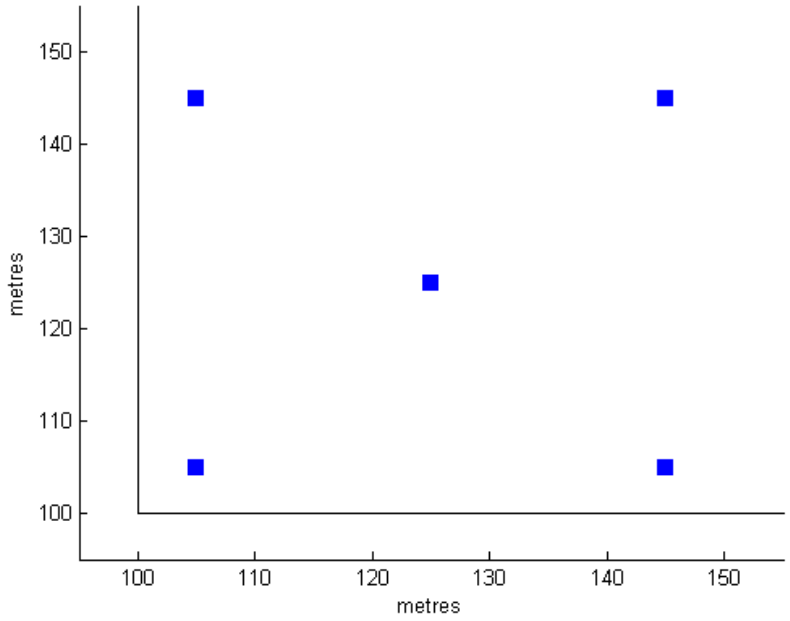


Figure 6.1: Position of applicators. The dam is 50x50 m, four applicators are placed 5 m in from the corners of each boundary and a fifth is located at the centre.

model cannot be used before it has been confirmed that the current settings permit convergence to occur.

The model (shown in Figure 6.1 [1]) was run with the same values in the `config.m` file several times, where each run is a realisation. The grid spacing is $dx = dy = 5$ m. The only variable that was varied for the 50x50 m dam was the number of particles per cell, np_{cell}. The values of np_{cell} is 10, 50, 100 and 500.

The cost savings were calculated in the `dolllarsaved.m` script. The value of water was set at $1000 per cubic meter and the cost of the monolayer was $0.01 per gram or $10 per kilogram. The running cost was set at $0 per timestep (600 s).

[1] Note that x_{min} and y_{min} are both 100. This is to ensure that the areas (calculated using a series of double integrals) in `calcArea.m` are positive. This limitation can be removed by using the `abs()` function in MATLAB on each double integral in the `calcArea` scripts – however both methods will produce the same results.

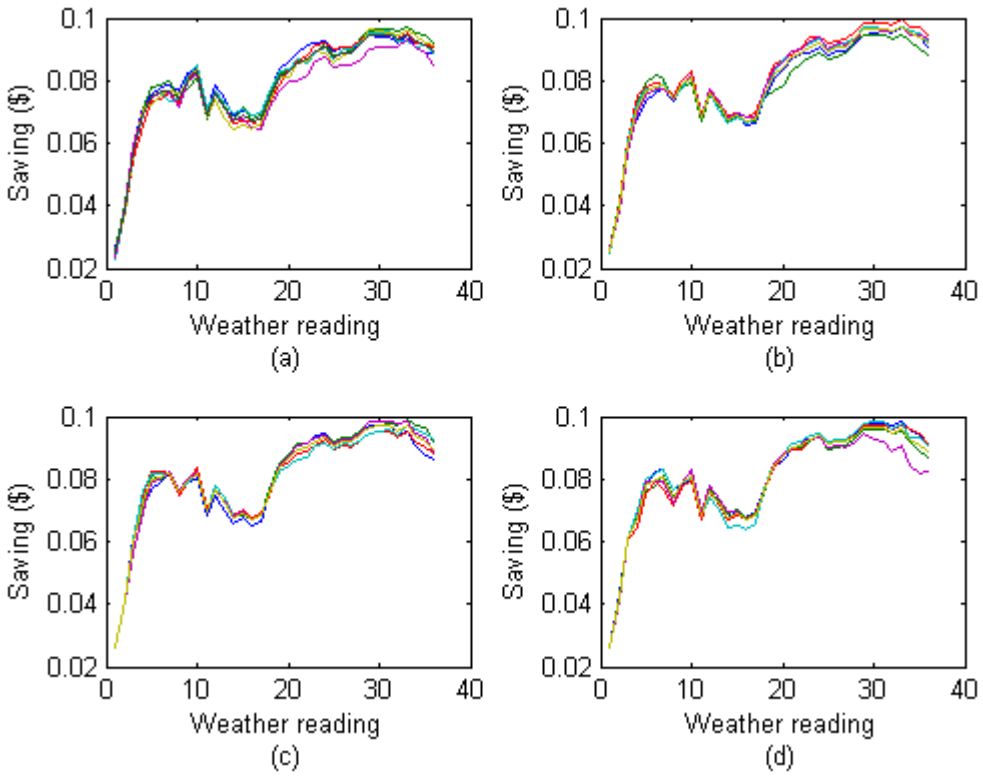


Figure 6.2: Results of simulation over 6 h with dx = dy = 5 m and np_cell = (a) 10, (b) 50, (c) 150 and (d) 500. The results show that the large number of particles can overcome the large spacing of 5 m on a 50x50 m domain. Another method that will yield similar results is to decrease the grid size while keeping np_cell constant.

6.2.1 Effect of smaller cells

The use of a smaller cell ($dx\ dy$) produces a more accurate solution than using a larger cell (when the number of particles per unit area is the same for both cases). The penalty for this increased accuracy is the time taken to find the average properties (in `avg.m`, specifically line 17) for each cell also increases.

6.2.2 Twenty four hour simulation

The program was also used to visualise how the the particles move over time. This test is mainly to show that the model behaves as expected. The model will need to be run for much longer than one day to get an indication of long term performance.

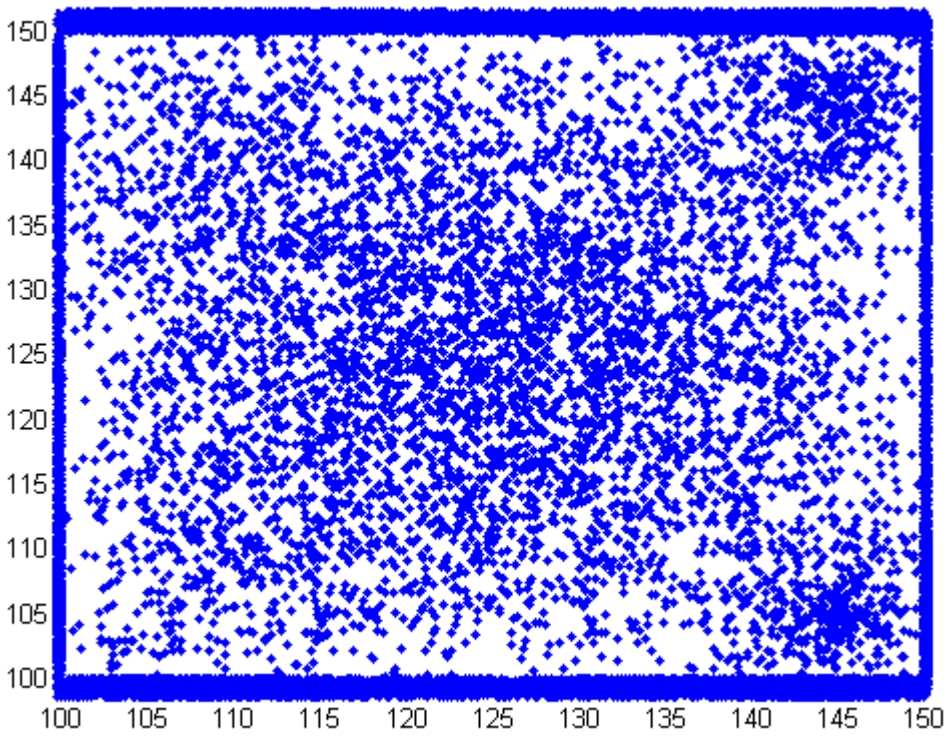


Figure 6.3: Scatter plot of simulation over 6h with dx = dy = 5 m and np_cell = 10. The size of the dam is 50x50 m.

A video was made by taking a screenshot of a scatter plot every ten minutes (the time between weather readings). The video consists of three sections of code:

`config.m` Lines 41-68 (comment out lines 70-73), and lines 107-109.

`Monolayer_Simulation.m` Lines 81-96

`Monolayer_Simulation.m` Line 128

The first point reorganised the wind vectors so each vector is repeated ten times. This makes the video smoother since a frame is taken every minute instead of every ten. Lines 107-109 open the video file and prepare it for writing. Lines 81-96 plot the points and save the graph as a frame in the video. Finally the video file is closed.

Figures 6.6 and 6.7 shows the movement of the particles with time. Each figure is taken 4 hours apart. The line of particles around the outside have stopped because

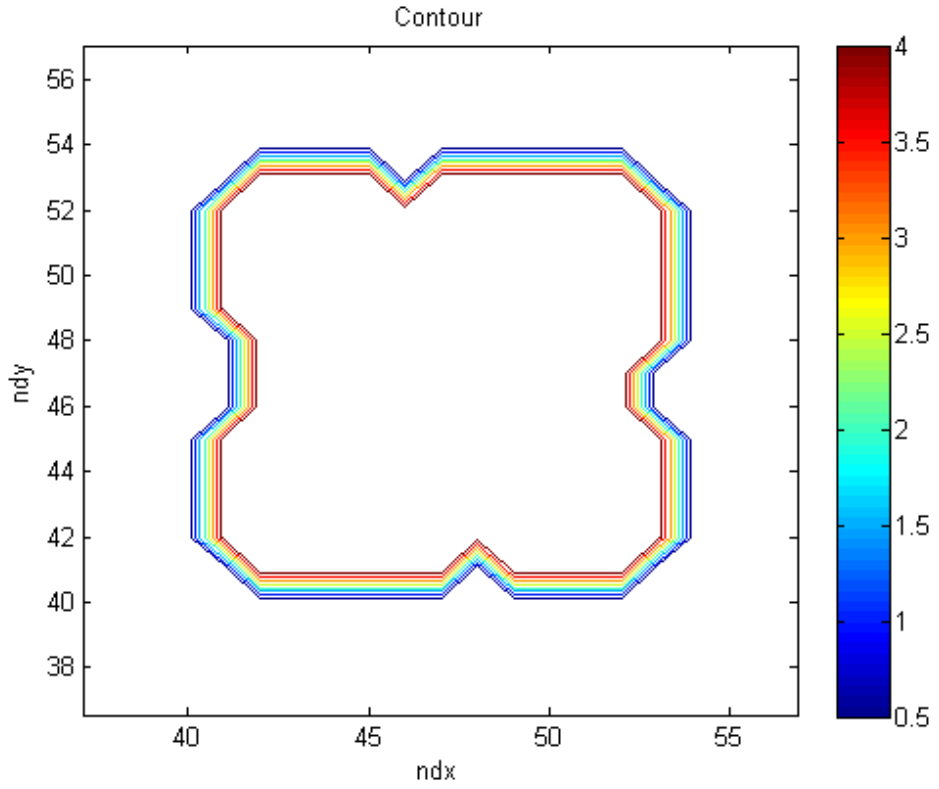


Figure 6.4: Contour plot of evaporation resistance (nr_m) after 6h with dx = dy = 5 m and np_cell = 500. The size of the dam is 50x50 m. The main disadvantage of using a large cell spacing can be seen in this plot - the average values of mass are calculated for the grid spacing and this results in drastic changes along the edges of the contour. The units for the colour bar are mg/m^2. The theoretical amount required to form a valid monolayer is 2.3 mg/m2. The impact this has on the model is discussed in Section 3.3.2: Evaporation resistance of monolayer as a function of wind speed.

they have passed outside the boundary. There is currently no code to take into account any shoreline effects (where the particles may be reintroduced onto the water surface when the wind direction changes.

Figure 6.8 shows the evaporation resistance provided by a single applicator. The size of the cell (dx dy) determines the resolution and accuracy of the contour (provided the number of particles per cell is also high enough to negate the effects of the random particles).

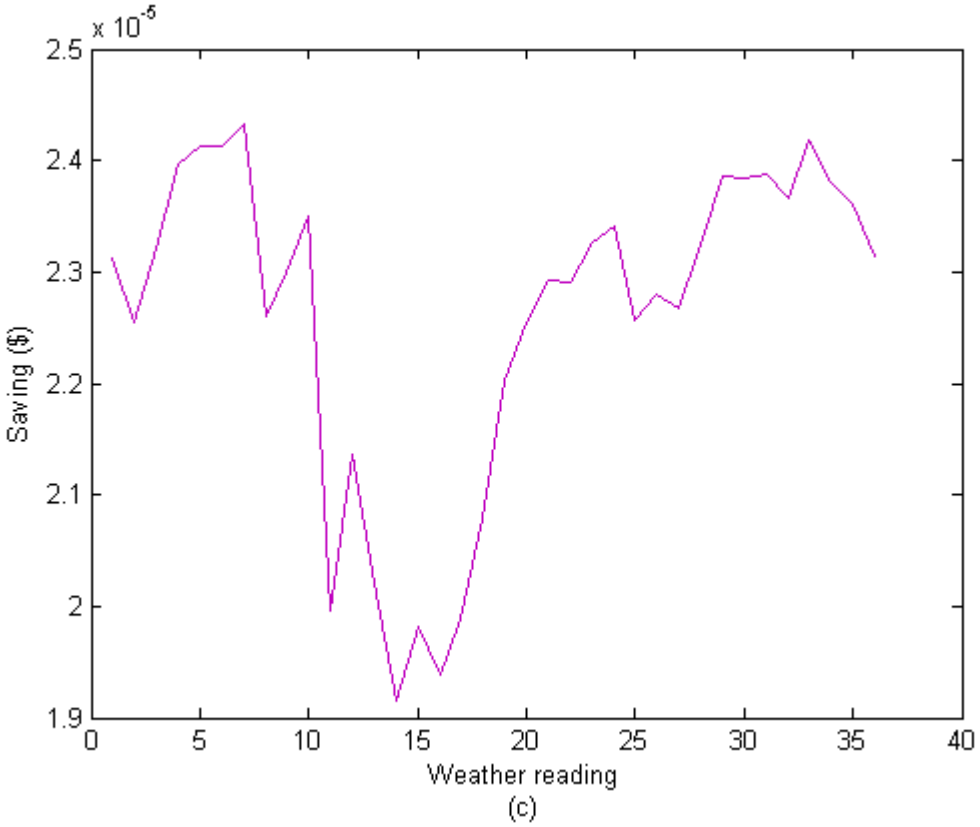


Figure 6.5: Results of simulation over 6 h with dx = dy = 1 m and np_cell = 20. This is the same particle area density (number of particles per unit area) as shown in Figure 6.5 (d).

6.3 Code extension

A larger sized dam was tested to demonstrate that the code is capable of larger domains, since this is where monolayers are more suited for evaporation control compared to competing solutions. The 500x50 m dam was run with a grid spacing of 5 m and 24 particles per cell. The applicators were located in along the length of the domain in the centre, spaced 50 m apart. The applicators were placed only in the centre because the size of the number of rates to calculate grows very quickly, significantly increasing solution times (see Section 4.3.5).

Figures 6.5 and 6.10 shows how the average of several realisations converge as the number of particles per cell is increased while keeping the grid spacing constant. Another method to check for convergence is to decrease the size of the grid spacing while keeping

the number of particles per cell constant. Figures 6.3 and 6.11 show the scatter plots for the two different dam sizes. Each dot represents a particle which has an independent mass and age.

If the model is to used for real time control purposes, (as opposed to simulation and prediction for site selection and anticipated cost savings prior to deployment,) then the np_cell and dx,dy values will need to be used from the prediction phase. This involves testing different values of np_cell and grid spacing to determine a compromise between a good approximation and a fast solution. This will need to assessed over a long period of time in order to determine the critical values (of np_cell and grid spacing) for a particular site.

6.4 Conclusion

The results show that the model is able to converge with a reasonable particle density and give usable results which can be indicative of real monolayer performance. The methodology to ensure that the results can be trusted is to follow these steps:

1. The cell size dx by dy should be relatively small.
2. A number of realisations should be calculated to ensure the difference between different realisations is small. This indicates that the model has converged on a solution that is independent of the random numbers used.
3. Several different plots can be produced to see if the results make sense including scatter (X vs Y), line (money saved vs weather reading) and contour plots (nr_m). The first plot shows the location of the particles, the second shows if there are any differences between multiple realisations and the third shows where the monolayer is reducing the evaporation of the water (and the concentration of the monolayer).

The following chapter summaries the current achievements and offers suggestions for future work.

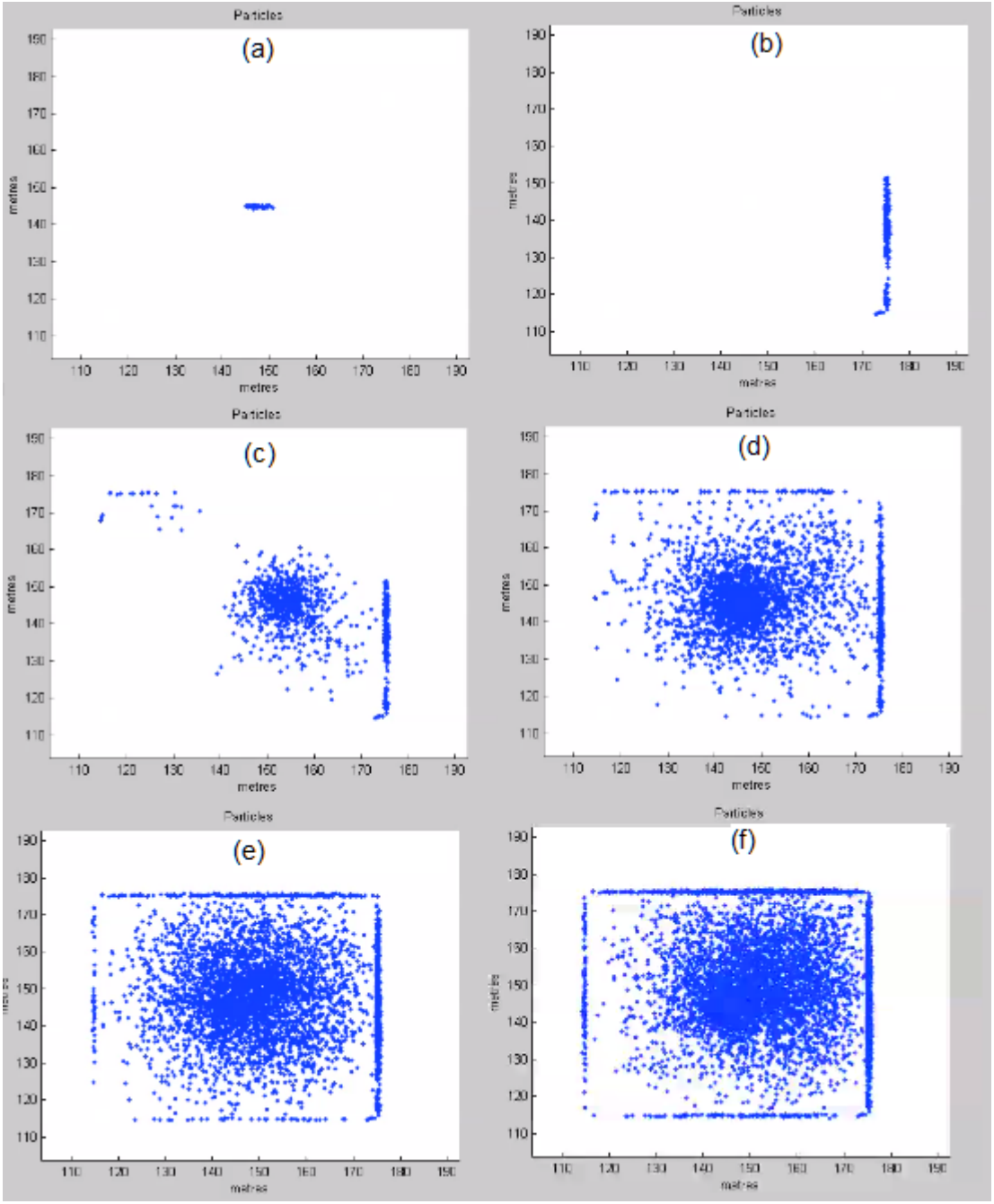


Figure 6.6: Results of 24 h simulation showing position of particles after (a) 0 h (b) 4 h (c) 8 h (d) 12 h (e) 16 h (f) 20 h. The dark line around the edge is the particles stopping after crossing the boundary.

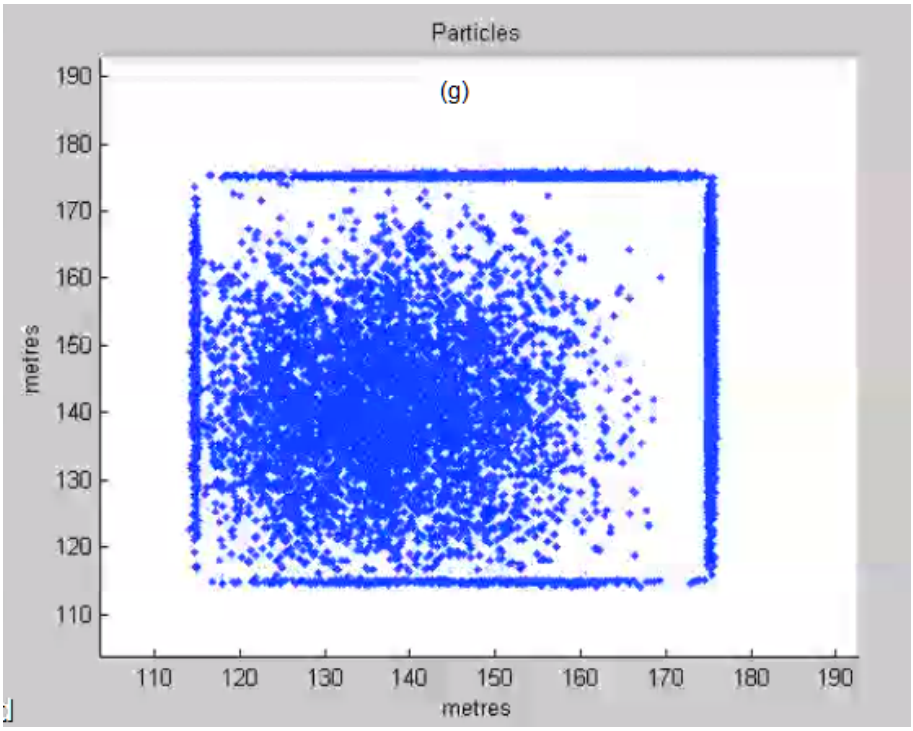


Figure 6.7: Position of particles after (g) 24 h. This series of images show how the particles move with time. Each particle has it's own effective mass (which is modified due to degradation) and age.

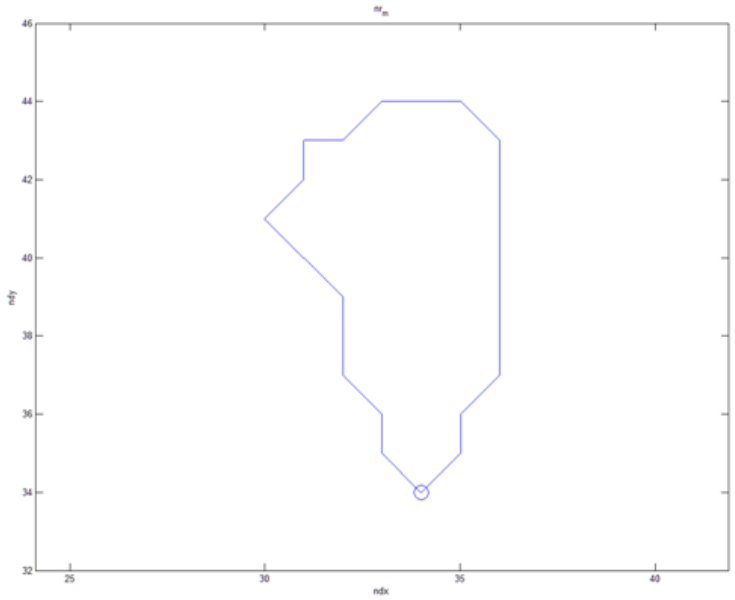


Figure 6.8: Evaporation resistance provided by a single applicator located at the circle.

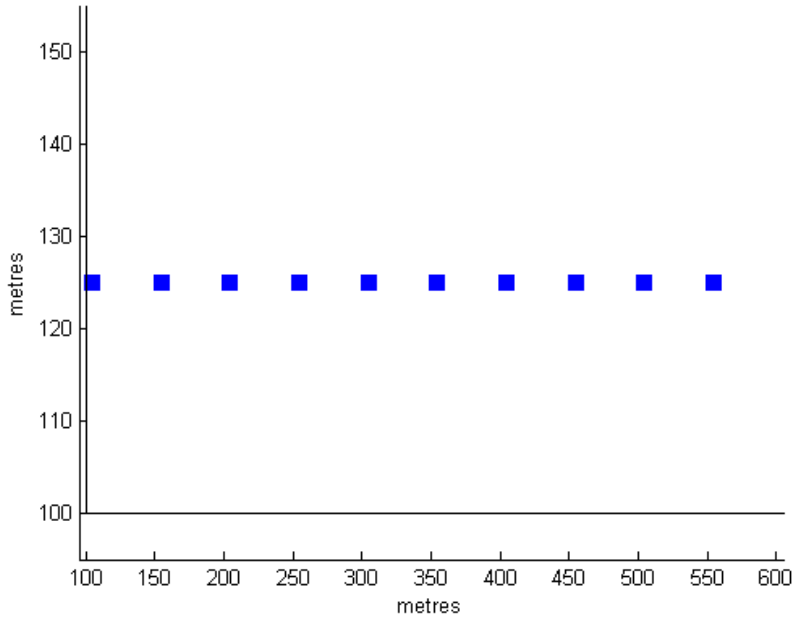


Figure 6.9: Position of applicators. The dam is 500x50 m, applicators are placed 50 m apart from each other starting 5 m in from the left hand boundary.

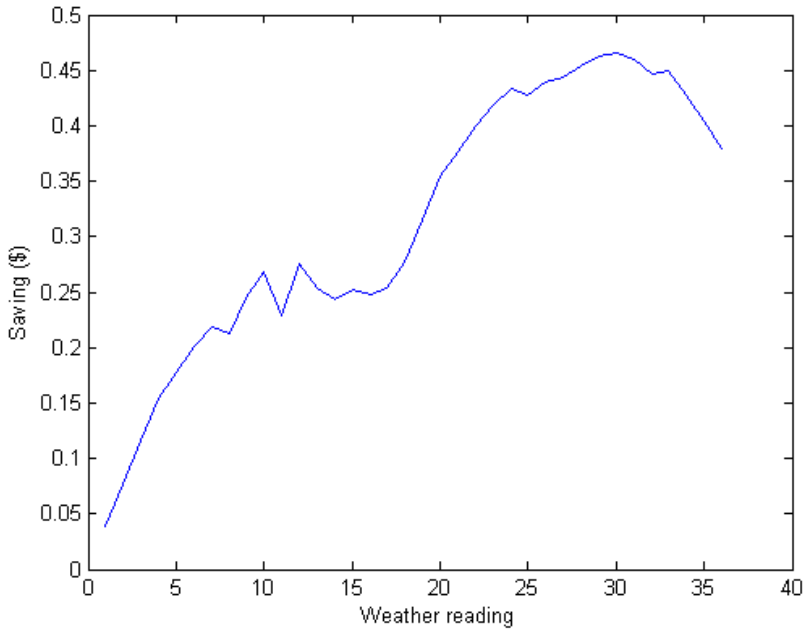


Figure 6.10: Results of simulation over 6h with dx = dy = 5 m and np_cell = 24. The size of the dam is 500x50 m.This shows that the model is capable of modelling larger dams.

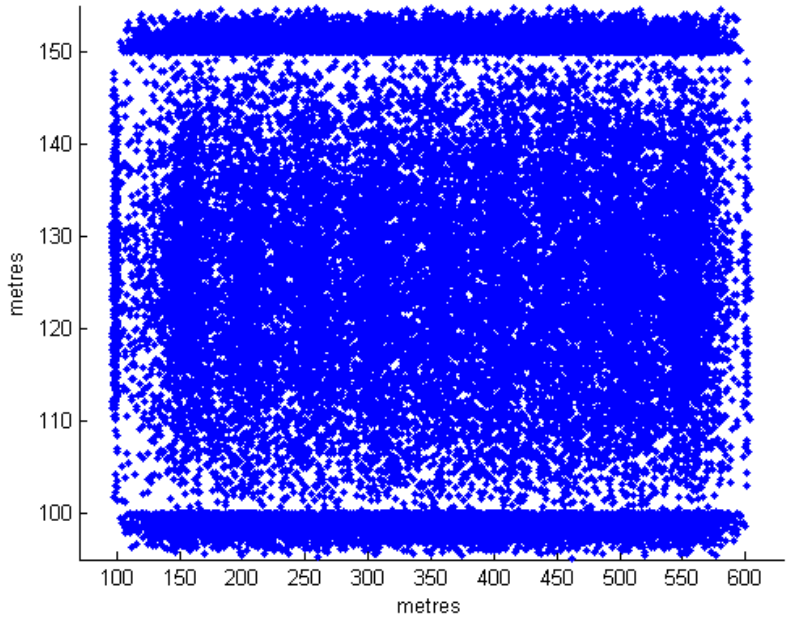


Figure 6.11: Scatter plot of simulation over 6h with dx = dy = 5 m and np_cell = 24. The size of the dam is 500x50 m

Chapter 7

Conclusions and Further Work

7.1 Achievement of Project Objectives

7.1.1 Research experiments to obtain data/ equations to describe behaviour of the monolayer

The experiments conducted by Brink (2011) provided the equations that were used to describe the movement of the monolayer - both the angle of spread and the drift speed. The relationship between the evaporation resistance and wind speed was based on McJannet et al. (2008). The degradation aspect of the model was separated into shoreline interaction, volatilisation, submergence and biological attack. No experimental data was available for either the shoreline interaction or submergence components and this has been identified as further work in sections 7.2.1 and 7.2.2.

The volatilisation effect have been previously reviewed by Brink. This research was able to obtain an approximate equation based on experimental work by Brooks and Andrews (1960). The biological effects were accounted for using experimental results from Pittaway (2008).

7.1.2 Create matrices to represent domain, boundaries and applicator positions for simulation

The matrices for the major variables are preallocated in the model and it permits an array of wind vectors to be used as input. The model also allows the applicators to be placed anywhere on the water surface. The boundaries are rectangular, but this restraint can be removed in the future.

7.1.3 Simulate monolayer movement and degradation over time, using historical weather data as input

The model simulates the permutations of the applicator rates and positions in order to identify the optimal applicator rate for each applicator. The model uses particles to represent the monolayer film. Each particle moves independently of each other and degrades over time due to volatilisation and biological degradation. Weather data is in the form of a list of wind vectors (speed and direction).

7.1.4 Use an objective function to control application of monolayer

The permutations of applicator rates and positions are compared against each other by calculating the amount of money saved. The amount of money saved is simply the difference between the value of the water saved from evaporation due to the presence of the monolayer and the cost associated with applying the monolayer.

7.1.5 Validate model by comparing results of simulation to real experiments

No suitable experiments were identified during the literature review. This is due to the difficulty in measuring effectiveness (evaporation resistance) of monolayers over large areas. This difficulty with getting detailed experimental results was the main reason for this research project. The lack of detailed experimental results makes it difficult

to validate the model further than ensuring convergence as shown in Chapter 6. It is hoped that this model can be improved upon and used to predict the effectiveness of monolayers.

7.1.6 Summary of achievements

A model has been created that can predict the cost savings due to the reduction in evaporation from applying a monolayer to the water surface. The model allows the user to specify the locations of the applicators, the dimensions of the rectangular water body, and a wind vector. The user must specify the size of the grid which is used to average the properties of the particles to determine how effective each of the permutations of the application rate is at generating the most profit. The permutation with the best profit is then used as the starting position for the next wind vector. This is repeated until all the optimal application rates have been found for all the wind vectors.

7.1.7 Comparision with Brink's model

There are unfortunately no experimental results available to compare the model against. The model that Brink (2011) developed has very different objectives compared with the model this research has achieved. Brink's model cannot be compared against this model because the purpose of Brink's model is to find the amount of monolayer applied and the coverage map for a single wind vector and mass application rate. This static model has limited real world applications. The model from this research project is dynamic since it can accept an array of wind vectors and find the optimal application rate.

7.2 Further Work

7.2.1 Shoreline interaction

The losses associated with the monolayer interacting with the shoreline will need to be investigated. The important aspects to investigate include the rate of degradation of the monolayer while it is in contact with the shoreline and the rate that the monolayer is reintroduced to the water surface when the wind direction changes. The rate of bio-degradation will also be different when the monolayer is not in contact with the water.

It will be difficult to get good results numerically so it would be necessary to perform a series of experiments. The experiments would be used to find the relationship between the important parameters that define the shoreline. The parameters include (but not limited to):

Material Examples include rock, soil, vegetation. Care must also be taken to take into account how easy it would be to identify which material is on the shoreline for large dams - is it possible to identify from satellite images, aerial photographs or from the ground. The longer it takes to identify, the higher the cost and this dramatically lowers how useful the model is.

Geometry The geometry of the shoreline is dependent on the material. The geometry (shape and size) of a rock is very different to that of vegetation (e. g. trees and reeds) and this must be investigated. The geometry should be split into simple categories that can easily be identified from satellite or aerial photographs, if possible.

7.2.2 Submergence

Submergence is the loss of monolayer due to waves which are large at higher wind speeds accounts for up to between 33 and 50% of the monolayer to be lost (see Section 2.4.5). It would be better if this can be performed numerically as opposed to experimentally

since this will make it easy to repeat the simulation for different monolayers to compare their performance. A Computational Fluid Dynamics (CFD) simulation would be best suited to investigate this aspect of the monolayer's behaviour. Once the methodology of the CFD has been validated, it can be easily modified to find the relationship for different monolayers between the wind vectors, time and the fractional mass lost. The simulation can be used to investigate both the steady state and transient response.

The steady state response entails several steps of increasing difficulty. Firstly, investigate the fraction of mass that is submerged at a constant wind speed and direction. If possible, investigate the minimum water depth that the relationship between a variety of wind speeds and fractional mass loss is valid.

The transient response is significantly more difficult to investigate because the number of different test cases can easily get too large to test if the methodology is not carefully thought out. The important parameters that will need to be investigated include the effect of time when changing wind vectors - how long does it take for the rate of fractional loss to move from steady state, through the transient phase and back to steady state? If this period of time is significant, it can be accounted for in the current model by reducing the `time_combo` variable. Further explanation of how this can be modified (without manually re-entering the wind vectors) can be found in Section 6.2.2.

The transient response modelling efforts will need to investigate how changing the wind vector suddenly (since the wind data is currently available in ten minute averages) effects the response. It may be necessary to 'smooth' the data so the wind vectors do not have sudden large changes to direction and speed, since this is not realistic.

The effect of rain can also be investigated using this CFD model.

Submergence model and application rate

The submergence model (using CFD) may also help better identify optimal application rates, since existing equations are for applying product from a long distribution line, and these are not well suited for point application. It would be ideal if the submergence model could identify a set of application rates that provide coverage at different wind

speeds. This information could then be used to calculate a single application rate as a function of wind speed. This could replace the application rate calculated in the variable `R` in `calcrates.m`. The use of a single rate (as opposed to using two or three different rates that need to be tested) would keep the simulation running as fast as possible. The current model uses a single application rate (for best performance), and each applicator either applies this rate or no product at all (see lines 176-189 in `calcrates.m` for details).

7.2.3 Effect of monolayer on temperature of water

Crow and Mitchell (1975) state that the "temperatures of film-covered water surfaces may become 5°F warmer than nonfilm areas. If the film cover is removed, evaporation occurs at a higher than normal rate until equilibrium conditions are reached. The consequences of interrupted film application may be serious. In an experimental pond, Crow (1961) found that evaporation reduction was lowered from 25 to 6.5% as a result of alternate 12-hour interruptions in film application." The temperature of the water surface will need to be modelled and change as the monolayer covers or exposes a section ($dxdy$) of the water.

7.2.4 Non-rectangular boundaries

It would be good if the model could be expanded so that it can simulate non-rectangular boundaries. The domain of the dam can either be broken into a large number of small rectangles which can approximate the curves of a dam, or straight and curved lines can be used to define the boundary. Both solutions will need to consider the end use of the model which is to be used for large dams. It is important to remove as much manual data entry as possible. Appendix D.1: Boundaries offers several different methods. The use of GPS coordinates appears to be the easiest to implement for large dams. The difficulty with this method is writing code that calculates where the boundary lies from the GPS points.

The use of non-rectangular boundaries will require modifications to the `calcArea` and

`boundary` scripts. It may be easier to replace the `calcArea` scripts with a simpler numerical method (as opposed to the current method which involves evaluating a series of double integrals). The difference in performance should also be considered.

7.2.5 Performance

The performance of the model will need to be monitored to ensure it is fast enough to give useful results for large bodies of water with a reasonable number of applicators and different possible applicator rates. The current model uses the `calcArea1.m` through `calcArea4.m` to reduce the number of calculations that need to be performed. It might be worth investigating the use of an alternative language such as C, C++ or FORTRAN if better performance is desired.

References

Barnes, G. T. (2001), 'The equilibrium penetration of monolayers: is equilibrium really established?', *Colloids and Surfaces A* **190**, 145–151.

Barnes, G. T. (2008), 'The potential for monolayers to reduce the evaporation of water from large water storages', *Agricultural Water Management* **95**(4), 339–353.

Barnes, G. T. & Gentle, I. R. (2011), *Interface science: an introduction*, Oxford University Press, New York.

Brink, G. N. (2011), Universal design framework for optimal application of chemical monolayer to open water surfaces, PhD thesis, University of Southern Queensland, Toowoomba, Australia.

Crow, F. R. & Mitchell, A. L. (1975), 'Wind effects on chemical films for evaporation suppression at lake hefner', *Water Resources Reseach* **11**(3), 493.

Frenkiel, J. (1965), 'Evaporation reduction: physicical and chemical principles and review of experiments'.

Henry, D. J., Dewan, V. I., Prime, E. L., Qiao, G. G., Solomon, D. H. & Yarovsky, I. I. (2010), 'Monolayer structure and evaporation resistance: a molecular dynamics study of octadecanol on water', **114**(11), 3869–3878.

Herzig, M., Barnes, G. & Gentle, I. (2010), 'Improved spreading rates for monolayers applied as emulsions to reduce water evaporation', *Journal of Colloid and Interface Science* **357**, 239–242.

McJannet, D. L., Cook, F., Knight, J. & Burn, S. (2008), Evaporation reduction by

monolayers: overview, modelling and effectiveness, Technical report, Urban Water Security Research Alliance.

McJannet, D. L., Webster, I., Stenson, M. P. & Sherman, B. S. (2008), Estimating open water evaporation for the murray-darling basin, Technical report, CSIRO murray-darling basin sustainable yields project.

Myers, D. (2006), *Surfactant science and technology*, 3rd edn, John Wiley and Sons, New Jersey.

Peng, J. B., Barnes, G. T. & Gentle, I. R. (2001), 'The structures of langmuir-blodgett films of fatty acids and their salts', *Advances in Colloid and Interface Science* **91**, 163–219.

Rastogi, M. C. (2003), *Surface and interface science: applications to engineering and technology*, Narosa Publishing House, India.

Schmidt, E. & Scobie, M. (2012), Improving irrigation efficiency by identifying methods to reduce evaporation losses from on-farm storages in the Granite Belt, Technical report, National Centre for Engineering in Agriculture.

Appendix A

Project Specification

ENG 4111/2 Research Project

Project Specification

For:	**Matthew du Preez**
Topic:	Simulation of Monolayer for use in Evaporation Reduction
Supervisors:	Dr Andrew Wandel
Sponsorship:	Faculty of Engineering & Surveying
Project Aim:	This project seeks to simulate monolayer movement on a dam to enable the calculation of the distribution of the monolayer over time, and select appropriate applicator rates to satisfy an objective function.

Program:

1. Research experiments to obtain data/ equations to describe behaviour of the monolayer.
2. Create matrices to represent domain, boundaries and applicator positions for simulation.
3. Simulate monolayer movement and degradation over time, using historical weather data as input.
4. Use an objective function to control application of monolayer.
5. Validate model by comparing results of simulation to real experiments.

As time and resources permit:

1. Use model to optimise location of applicators.

Agreed:

Student Name:

Date:

Supervisor Name:

Date:

Examiner/Co-Examiner:

Date:

Appendix B

Degradation Calculations

B.1 Introduction

This appendix details how the equations for the vaporisation and biological degradation were calculated as summarised in Chapter 2.

B.2 Vaporisation of monolayer

The most complete data that was found was Table B.1 and is reproduced below.

Table B.1: Loss of monolayer material due to evaporation. Reproduced from Brink (2011).

Monolayer	Fractional loss ($\times 10^{-6}$ s^{-1})		
	$5°C$	$20°C$	$40°C$
Myristyl $C_{14}OH$	20	58	1900
Cetyl $C_{16}OHl$	1	4	150
Stearyl $C_{18}OH$	0	0	20

Only the Myristyl and Cetyl monolayers had enough information to produce useful equations of best fit.

$$y_{\text{Myristyl}} = 7.3713e^{0.1325t}$$

$$y_{\text{Cetyl}} = 0.3627e^{0.1452t}$$

where y is the fractional loss and t is the time elapsed [s]. Since there is insufficient data to produce a line of best fit for Stearyl, (which was used by Brink to experimentally determine the spreading rates,) the exponent was selected as a rough average of the exponents of Myristyl and Cetylthe. This yielded the equation of the line for Stearyl $y = Ce^{0.14t}$. At the only known data point (40,20), $20 = Ce^{0.14\times 40}$, yielding C = 0.073957. The equation that describes the fractional loss of monolayer due to volitalisation is

$$y = 0.073957e^{0.14t} \times 10^{-6}s^{-1} \tag{B.1}$$

This equation was used to find the values in table B.2. The equation fits the data well, but it is limited due to the low number of data points.

Table B.2: Results of Stearyl equation

t	5	20	40
y	0.149	1.216	19.999

B.3 Biological degradation

Table B.3 is the data read from Pittaway's graph (see figure 2.4 on page 16), showing decreasing concentration with time due to biological attack.

Table B.3: Loss of monolayer material due to biological degradation

	Incubation Time (d)			
Point (mM)	0	2	3	4
Max error bar	1.02	0.89	0.68	0.55
Measured value	0.93	0.82	0.59	0.48
Min error bar	0.85	0.74	0.51	0.40

Fig. B.1 shows that the line of best fit between the error bars can be used to determine the equation of the line. This equation shows the fractional loss of the monolayer where the monolayer was initially at one, and when no more monolayer remains is at zero. The working below shows how this equation was found, where y represents the fraction of monolayer remaining and t is the time that has elapsed since the monolayer was applied.

$$y = \frac{0.52 - 1.01}{4 - 0}t + 1.00$$

where t has the units of days.

$$y = -0.1225t + 1.00$$

when $t = 0$, and $y = 1$

$$y = -0.1225\frac{t}{3600 \times 24} + 10.00,$$

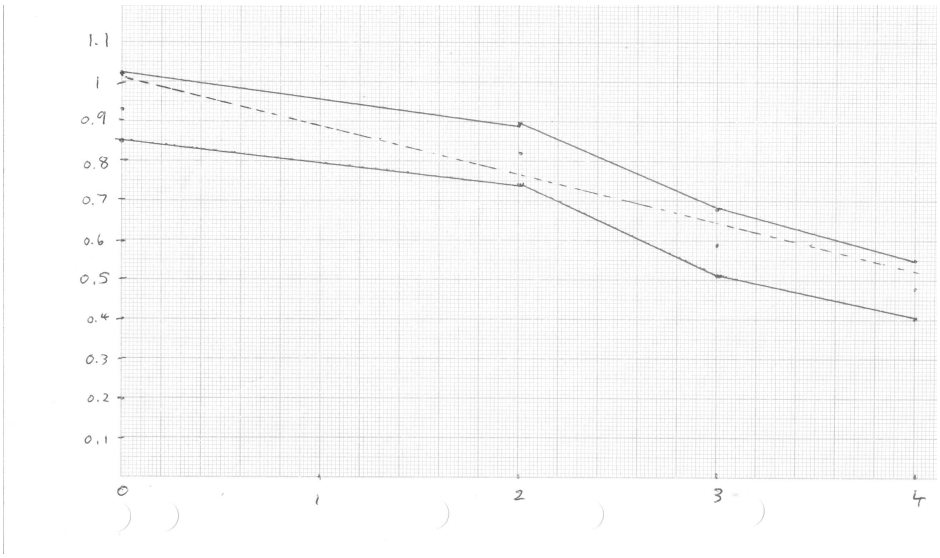


Figure B.1: Linear line of best fit for Stearyl biological degradation. The solid lines are used to connect the error bars between each point. The y axis the the fraction of monolayer remaining and the x axis is the number of days that have elapsed since the monolayer was applied.

where t has the units of seconds.

$$y' = -1.41782 \times 10^{-6} \quad /s \tag{B.2}$$

Eq. B.2 shows the rate at which the monolayer reduces in concentration, from an initial value of 1 to 0. This means that if biological degradation is the only form of degradation the monolayer has a maximum useful life of approximately 8 days.

Appendix C

Source Code

C.1 avg.m

This script calculates the average properties of the particles so that the evaporation resistance can be found.

Listing C.1: avg script.

```
% avg

nX(1:im3) = int8(ceil( (X(1:im3)-dx/2) / dx )+1);
nY(1:im3) = int8(ceil( (Y(1:im3)-dy/2) / dy )+1);
unXY = unique([nX(1:im3)' nY(1:im3)'],'rows');
% unique values of nX, nY in ascending order nX then nY.
nr_m = zeros(length(unXY(:,2)),length(unXY(:,1)));

nmass=zeros(1,size(unXY,1));
nage=zeros(1,length(unXY));
nconcentration=zeros(1,length(unXY));
cond=zeros(1,size(unXY,1));

for k=1:size(unXY,1)
    % get index where unique (x,y) matches all particles
        % following line takes majority of time for avg script
    cond = find( (nX == unXY(k,1)) & (nY == unXY(k,2)) );
    % add all particles at same (x,y) position
    nmass(k) = sum(mass(cond));
    nmass(unXY(k,2)+nX_min,unXY(k,1)+nY_min)=nmass(k);
    nconcentration(unXY(k,2)+nX_min,unXY(k,1)+nY_min)=...
        nmass(unXY(k,2)+nX_min,unXY(k,1)+nY_min)/(dx*dy);

    % weighted average age
    nage(unXY(k,2)+nX_min,unXY(k,1)+nY_min)=...
        sum( (age(cond).*mass(cond))/...
                nmass(unXY(k,2)+nX_min,unXY(k,1)+nY_min) );
```

```
    %r_m: uWind<6.71 & concentration>2x (where 1x=2.3mg/m^2)...
        % * eq. of line
    nr_m(unXY(k,2)+nX_min,unXY(k,1)+nY_min)=...
        (uWind(windloop)<UWINDMAX)*...
        (nconcentration(unXY(k,2)+nX_min,unXY(k,1)+nY_min)>2*2.3e-3)*...
        ((-5.0/UWINDMAX)*uWind(windloop)+5);
end
```

C.2 boundary.m

This script calculates whether the particles are inside or outside a rectangular boundary.

Listing C.2: boundary script.

```
% boundary
% Rectangular boundary. Sets pact flag to 'false'
% when particle passes outside boundary.

% Are the particles active? (true/false)

% index for boundary
ib1 = 1+windloop*np_cell*k*(cc-1);

%ib2 = windloop*np_cell*k*count;
ib2= im3;

% Are particles < x_max
pact_x1(1:ib2) = (X(1:ib2) <x_max);
% Are particles > x_min
pact_x2(1:ib2) = (X(1:ib2) >x_min);
% Are both pact_x1 and pact_x2 true
pact_x(1:ib2) = (pact_x1(1:ib2) & pact_x2(1:ib2));

% Are particles < y_max
pact_y1(1:ib2) = (Y(1:ib2) <y_max);
% Are particles > y_min
pact_y2(1:ib2) = (Y(1:ib2) >y_min);
% Are both pact_y1 and pact_y2 true
pact_y(1:ib2) = (pact_y1(1:ib2) & pact_y2(1:ib2));

% Are both part_x and part_y true
pact(1:ib2) = (pact_x(1:ib2) & pact_y(1:ib2));
```

C.3 calcArea1.m

This script calculates the proportion of area covered by the monolayer for case 1.

Listing C.3: calcArea1 script.

```
% calcArea1
```

```
% when angleSpread == 2*pi
% finding areaOutside
% There are 8 possible cases to test for.
% After testing evaluate double integral over limits to get area.
%%%%%%%%%
% Identify case

% make matlab think warning is actually error --> can now use try.. catch
% block to handle warning as it defined as error.
state = warning('error', 'symbolic:sym:int:notFound');
warning('error', 'symbolic:sym:int:notFound');
try

c1 = a+r;
c2 = b+r;
c3 = a-r;
c4 = b-r;
% identify case, numbering from pos x axis in an anti-clockwise
% (positive) direction
if      (c1>x_max && (c2<y_max && c4>y_min))
    areaCase1 =  1; % mid right
elseif  (c3<x_min && (c2<y_max && c4>y_min))
    areaCase1 =  5; % mid left
elseif  (c2>y_max && (c1<x_max && c3>y_min))
    areaCase1 =  3; % mid top
elseif  (c4<y_min && (c1<x_max && c3>y_min))
    areaCase1 =  7; % mid bottom
elseif  (c1>x_max && c2>y_max)
    areaCase1 =  2; % top right
elseif  (c3<x_min && c2>y_max)
    areaCase1 =  4; % top left
elseif  (c1>x_max && c2>y_min)
    areaCase1 =  8; % bottom right
elseif (c1>x_min&&c3<x_min && c2>y_min&&c3<y_min)
    areaCase1 =  6; % bottom left
else
    break
end
syms x y
% Equation for a circle
% (x-a)^2 + (y-b)^2 = r^2
eqyp = b + sqrt(r^2-(x-a)^2); % y=
eqym = b - sqrt(r^2-(x-a)^2); % y=
%eqxp = a + sqrt(r^2-(y-b)^2); % x=
%eqxm = a - sqrt(r^2-(y-b)^2); % x=

% Solve for limits
%[limx1,limy1]=solve(eqyp-y,x_max-x);
%[limx2,limy2]=solve(eqym-y,x_max-x);

% Find area
% area=double(int(int(1,x= eq(lower),x= eq(upper)),
% xlim(lower),xlim(upper)))

% numbered in an anti-clockwise direction starting at mid right.
switch areaCase1
```

```
    case 1
        areaOutside = double(int(int(1,eqym,eqyp),x_max,c1));
    case 2
        [limx1,limy1]=solve(eqyp-y,y_max-y);
        areaOutside = double(int(int(1,eqym,eqyp),x_max,c1))+...
                      double(int(int(1,y_max,eqyp),limx1(1),x_max));
    case 3
        [limx1,limy1]=solve(eqyp-y,y_max-y);
        areaOutside = double(int(int(1,y_max,eqyp),limx1(1),limx1(2)));
    case 4
         [limx1,limy1]=solve(eqyp-y,y_max-y);
         areaOutside = double(int(int(1,eqym,eqyp),c3,x_max))+...
                       double(int(int(1,y_max,eqyp),x_max,limx1(2)));
    case 5
        areaOutside = double(int(int(1,eqym,eqyp),c3x_max));
    case 6
        [limx1,limy1]=solve(eqyp-y,y_min-y);
         areaOutside = double(int(int(1,eqym,eqyp),c3,x_max))+...
                       double(int(int(1,eqym,y_min),x_max,limx1(2)));
    case 7
        [limx1,limy1]=solve(eqym-y,y_min-y);
        areaOutside = double(int(int(1,eqym,y_min),limx1(1),limx1(2)));
    case 8
       [limx1,limy1]=solve(eqym-y,y_min-y);
        areaOutside = double(int(int(1,eqym,y_min),limx1(1),x_max))+...
                      double(int(int(1,eqym,eqyp),x_max,c1));
    otherwise
        areaOutside = Inf; % doesn't match a case
end
catch
    % cannot find integral - cannot exclude from future checks -
        % ie make areaOutside small.
    areaOutside = 1E-9;
end

%set warning back to previous state.
warning(state)
```

C.4 calcArea2.m

This script calculates the proportion of area covered by the monolayer for case 2.

Listing C.4: calcArea2 script.

```
% calcArea2
% one out
% finding areaoutside

% There are 16 possible cases to test for.
% After testing evaluate double integral over limits to get area.

%%%%%%%%%
% Identify case

% make matlab think warning is actually error --> can now use try.. catch
```

```
% block to handle warning as it defined as error.
state = warning('error', 'symbolic:sym:int:notFound');
warning('error', 'symbolic:sym:int:notFound');

try
% identify case
% Starting numbering from bottom right and moving around
% in an anti-clockwise direction.
if      ( (x1>x_min&&x1<x_max)&&(y1<b) &&...
          (x2>x_max)&&(y2<y_max&&y2<b) );
          areaCase2=1;

elseif  ( (x1>x_min&&x1<x_max)&&(y1<b) &&...
          (x2>x_max)&&(y2<y_max&&y2>b) );
          areaCase2=2;

elseif  ( (x1>x_min&&x1<x_max)&&(y1>b) &&...
          (x2>x_max)&&(y2<y_max&&y2>b) );
          areaCase2=3;

elseif  ( (x1>x_min&&x1<x_max)&&(y1>b) &&...
          (x2>x_max)&&(y2<y_max&&y2<b) );
          areaCase2=4;

elseif  ( (x1>a)&&(y1<y_max) &&...
          (x2>a&&x2<x_max)&&(y2>y_max) );
          areaCase2=5;

elseif  ( (x1>a)&&(y1<y_max) &&...
          (x2<a&&x2<x_max)&&(y2>y_max) );
          areaCase2=6;

elseif  ( (x1<a)&&(y1<y_max) &&...
          (x2<a&&x2<x_max)&&(y2>y_max) );
          areaCase2=7;

elseif  ( (x1<a)&&(y1<y_max) &&...
          (x2>a&&x2<x_max)&&(y2>y_max) );
          areaCase2=8;

elseif  ( (x1>x_min)&&(y1>b) &&...
          (x2<x_min)&&(y2>b) );
          areaCase2=9;

elseif  ( (x1>x_min)&&(y1>b) &&...
          (x2<x_min)&&(y2<b) );
          areaCase2=10;

elseif  ( (x1>x_min)&&(y1<b) &&...
          (x2<x_min)&&(y2<b) );
          areaCase2=11;

elseif  ( (x1>x_min)&&(y1<b) &&...
          (x2<x_min)&&(y2>b) );
          areaCase2=12;

elseif  ( (x1>x_min&&x1<a)&&(y1>x_min) &&...
          (x2<a)&&(y2<b) );
          areaCase2=13;

elseif  ( (x1>x_min&&x1<a)&&(y1>x_min) &&...
          (x2>a)&&(y2<b) );
```

```
        areaCase2=14;

elseif  ( (x1>x_min&&x1>a)&&(y1>x_min) &&...
        (x2>a)&&(y2<b) );
        areaCase2=15;

elseif  ( (x1>x_min&&x1>a)&&(y1>x_min) &&...
        (x2<a)&&(y2<b) );
        areaCase2=16;
else
        error('cannot select case for calcArea2')
end


% now get 3 equation of line (for wedge) between
% (a,b)&(x1,y1), (x1,y1)&(x2,y2), (x2,y2)&(a,b)
% or
% equation of circle for angleSpread=2*pi
syms x y
% y= mx+b % m= gradient, b=y intercept
m1 = (y1-b)/(x1-a);
b1 = b-m1*a;
eq1 = m1*x + b1; % x is variable, eq1=y coord
eq11 = (y-b1)/m1; % eq11 = x

% y= mx+b % m= gradient, b=y intercept
m2 = (b-y2)/(a-x2);
b2 = y2-m2*x2;
eq2 = m2*x + b2; % x is variable, eq3=y coord
eq22 = (y-b2)/m2; % eq22 = x

% (x-a)^2 + (y-b)^2 = r^2
eqyp = b + sqrt(r^2-(x-a)^2); % y=
eqym = b - sqrt(r^2-(x-a)^2); % y=
eqxp = a + sqrt(r^2-(y-b)^2); % x=
eqxm = a - sqrt(r^2-(y-b)^2); % x=

% Solve for limits
%[limx1,limy1]=solve(eqyp-y,x_max-x);
%[limx2,limy2]=solve(eqym-y,x_max-x);

% Find area
% area=double(int(int(1,x= eq(lower),x= eq(upper)),xlim(lower),xlim(upper)

switch areaCase2
    case 1
        areaOutside=double(int(int(1,eqym,eq2),x_max,x2));
    case 2
        areaOutside = double(int(int(1,eqym,b),x_max,a+r))+... % bottom s
                      double(int(int(1,b,eq22),x_max,x2))+... % left of to
                      double(int(int(1,b,eqyp),x2,a+r)); % right of top
    case 3
        areaOutside=double(int(int(1,eq2,eqyp),x_max,x2));
    case 4
        areaOutside = double(int(int(1,b,eqyp),x_max,a+r))+... % top secti
                      double(int(int(1,eq2,b),x_max,x2))+... % left of bot
                      double(int(int(1,eqym,b),x2,a+r)); % right of bottom
    case 5
        [limx1,limy1]=solve(y_max-y,eq2-y); % straight line and intersect
        [limx2,limy2]=solve(eqyp-y,y_max-y);  % top of circle and y_max
```

```
        areaOutside=double(int(int(1,y_max,eq2),limx1,x2))+...
                    double(int(int(1,y_max,eqyp),x2,limx2(2)));
    case 6
        [limx1,limy1]=solve(y_max-y,eq2-y);  % straight line and y_max
        [limx2,limy2]=solve(eqyp-y,y_max-y);  % top of cirlce and y_max
        areaOutside=double(int(int(1,eq2,eqyp),x2,limx1))+...
                    double(int(int(1,y_max,eqyp),limx1,limx2(2)));
    case 7
        [limx1,limy1]=solve(y_max-y,eq2-y); % straight line and intersect
        [limx2,limy2]=solve(eqyp-y,y_max-y);  % top of circle and y_max
        areaOutside=double(int(int(1,y_max,eqyp),limx2(1),x2))+...
                    double(int(int(1,y_max,eq2),x2,limx1));
    case 8
        [limx1,limy1]=solve(y_max-y,eq2-y);  % straight line and y_max
        [limx2,limy2]=solve(eqyp-y,y_max-y);  % top of circle and y_max
        areaOutside=double(int(int(1,y_max,eqyp),limx2(1),limx1))+...
                    double(int(int(1,eq2,eqyp),limx1,x2));
    case 9
        areaOutside=double(int(int(1,eq2,eqyp),x2,x2));
    case 10
        areaOutside = double(int(int(1,b,eqyp),a-r,x_min)+... % top sectio
                      int(int(1,eqym,b),a-r,x2)+... % left of bottom
                       int(int(1,eq2,b),x2,x_min)); % right of bottom
    case 11
        areaOutside=double(int(int(1,eqym,eq2),x2,x_min));
    case 12
        areaOutside = double(int(int(1,eqym,b),a-r,x_min))+... % bottom s
                      double(int(int(1,b,eqyp),a-r,x2))+... % left of top
                      double(int(int(1,b,eq2),x2,x_min)); % right of top
    case 13
        [limx1,limy1]=solve(y_min-y,eq2-y); % straight line and intersect
        [limx2,limy2]=solve(eqym-y,y_min-y);  % bottom of cirlce and y_mi
        areaOutside=double(int(int(1,eqym,y_min),limx2(1),x2))+...
                    double(int(int(1,eq2,y_min),x2,limx1));
    case 14
        [limx1,limy1]=solve(y_min-y,eq2-y); % straight line and intersect
        [limx2,limy2]=solve(eqym-y,y_min-y);  % bottom of cirlce and y_mi
        areaOutside=double(int(int(1,eqym,y_min),limx2(1),limx1))+...
                    double(int(int(1,eq2,y_min),limx1,x2));
    case 15
        [limx1,limy1]=solve(y_min-y,eq2-y); % straight line and intersect
        [limx2,limy2]=solve(eqym-y,y_min-y);  % bottom of cirlce and y_mi
        areaOutside=double(int(int(1,eq2,y_min),limx1,x2))+...
                    double(int(int(1,eqym,y_min),x2,limx2(2)));
    case 16
        [limx1,limy1]=solve(y_min-y,eq2-y); % straight line and intersect
        [limx2,limy2]=solve(eqym-y,y_min-y);  % bottom of cirlce and y_mi
        areaOutside=double(int(int(1,eqym,eq2),x2,limx1))+...
                    double(int(int(1,eqym,y_min),limx1,limx2(2)));
    otherwise
        break;
end
catch
    % cannot find integral
    areaOutside = 1E-9;
end
%set warning back to previous state.
```

```
warning(state)
```

C.5 calcArea3.m

This script calculates the proportion of area covered by the monolayer for case 3.

Listing C.5: avg calcArea3.

```
% calcArea3
% both out (but only on one of the four boundaries ie not corner)
% finding areaInside - triangle
syms x y
% y= mx+b % m= gradient, b=y intercept
m1 = (y1-b)/(x1-a);
b1 = b-m1*a;
eq1 = m1*x + b1; % x is variable, eq1=y coord

% y= mx+b % m= gradient, b=y intercept
m2 = (b-y2)/(a-x2);
b2 = y2-m2*x2;
eq2 = m2*x + b2; % x is variable, eq3=y coord

% four possible sub-cases (for each boundary)

% find points where boundary intesects wedge
if out_xmax
    [limx1,limy1]=solve(eq1-y,x_max-x);
    [limx2,limy2]=solve(eq2-y,x_max-x);
elseif out_xmin
    [limx1,limy1]=solve(eq1-y,x_min-x);
    [limx2,limy2]=solve(eq2-y,x_min-x);
elseif out_ymax
    [limx1,limy1]=solve(eq1-y,y_max-y);
    [limx2,limy2]=solve(eq2-y,y_max-y);
else %out_ymin
    [limx1,limy1]=solve(eq1-y,y_min-y);
    [limx2,limy2]=solve(eq2-y,y_min-y);
end
% Calculate area using Heron's Forumula
% areaInside = sqrt(s(s-a)(s-b)(s-c)) where s=0.5*(a+b+c)
% areaInside = sqrt(st*(st-s1)*(st-s2)*(st-s3)) where
% st=0.5*(s1+s2+s3)
s1= sqrt((limx1-a)^2+(limy1-b)^2);
s2= sqrt((limx2-a)^2+(limy2-b)^2);
s3=sqrt((limx1-limx2)^2+(limy1-limx2)^2);
st=0.5*sum([s1 s2 s3]);
areaInside = sqrt(st*(st-s1)*(st-s2)*(st-s3));
```

C.6 calcArea4.m

This script calculates the proportion of area covered by the monolayer for case 4.

Listing C.6: calcArea4 script.

```
% calcArea4
% corner
% finding areaInside - quadrilateral

% given a quadrilateral made of 4 vectors so that:
% a + b + c + d = 0;
% then the diagonals are: p = b + c and q = a + b
% area = (1/2) * norm((cross(p,q))
%                 where norm is the 'size of' matlab func.

syms x y
% y= mx+b % m= gradient, b=y intercept
m1 = (y1-b)/(x1-a);
b1 = b-m1*a;
eq1 = m1*x + b1; % x is variable, eq1=y coord

% y= mx+b % m= gradient, b=y intercept
m2 = (b-y2)/(a-x2);
b2 = y2-m2*x2;
eq2 = m2*x + b2; % x is variable, eq3=y coord

% find the corner
if (x_max-a)<(a-x_min)
    if (y_max-b)<(b-y_min)
        areaCase4 = 1; %top right
    else
        areaCase4 = 2; % bottom right
    end
else
    if (y_max-b)<(b-y_min)
        areaCase4 = 3; % top left
    else
        areaCase4 = 4; % bottom left
    end
end

switch areaCase4
    case 1
        [px1,py1]=solve(eq1-y,y_max-y);
        [px2,py2]=solve(eq2-y,x_max-x);
        corner = [x_max y_max];
    case 2
        [px1,py1]=solve(eq1-y,y_min-y);
        [px2,py2]=solve(eq2-y,x_max-x);
        corner = [x_max y_min];
    case 3
        [px1,py1]=solve(eq1-y,x_min-x);
        [px2,py2]=solve(eq2-y,y_max-y);
        corner = [x_min y_max];
    case 4
        [px1,py1]=solve(eq1-y,y_min-y);
        [px2,py2]=solve(eq2-y,x_min-x);
        corner = [x_min y_max];
    otherwise
```

```
55          error('Could not calc points of intercept for calcArea4')
end

% get 4 vectors to define quadrilateral
va = double([px1-a py1-b]);
60 vb = double([corner(1)-px1 corner(2)-py2]);
vc = double([px2-corner(1) py2-corner(2)]);
% don't need vd to find diagonals

% get diagonal vectors
65 p = vb + vc;
q = va + vb;

% calc area (need 3dim to use cross)
areaInside = (1/2)*norm(cross([p 1],[q 1]));
```

C.7 calcmoveevapdollar.m

This script calculates calls the `movepart.m`, `avg.m` and `dollarsaved.m` scripts inorder to calculate amount of money saved based on the movement of the monolayer from the initial "current" arrays that are used so that different permutations of applicator rates can be compared.

Listing C.7: calcmoveevapdollar script.

```
% calcmoveevapdollar

%=================
% Load rand seed
5 rng(s)

%=====================
% Load current arrays
X = Xc; Y = Yc; mass = massc;
10 pact = pactc; age=agec;
pact_x=pact_xc;pact_x1=pact_x1c;pact_x2=pact_x2c;
pact_y=pact_yc;pact_y1=pact_y1c;pact_y2=pact_y2c;
totalmass = totalmass_c;

15 %=================================================
% Loop through time for each rates combination
cc=0; % combination counter
time=0;

20 while time < time_combo
    cc=cc+1;
    cfl
    movepart
end
25
%=================================================
% Find average mass (and age) for each cell
avg

30 %=================================
```

```
% Find evap with and without product
% & Find $, save index for best combo so far
[E_diff] = evap(nr_m);
dollarsaved
```

C.8 calcrates.m

This script finds the permutations of mass application rates and applicators. It also calls the four calcArea scripts in order to reduce the amount of permutations that are simulated. The permutations are not simulated if the amount of area covered by an applicator is less than a user defined constant PROPINSIDE which is the proportion of area inside the boundary that the monolayer will cover for each individual applicator.

Listing C.8: calcrates script.

```
% calcrates:
% At the start of each new windspeed and windDirection (windloop),
% find the matrix containing the application rates to be tested (combos)
% Each 'combo' consists of a unique permutation of rates.
%            app1     app2     appn
% combo      rate     rate     rate

if uWind(combo) > UWINDMAX
        % don't apply since it will have no effect on evap resistance
    rates = zeros(1,size(applicators,1));

else

%preallocate
unfavourableApp=zeros(1,size(applicators,1));

% crow and mitchell
% R = (1.18U^1.81)e-4, R[lb/h/ft], U[mi/h]
% for dist_line ft, per second, grams, wind in m/s

% [m] length of distribution line (spacing between applicators)
dist_line = 50;

% [g/s/m] = [3.28 m =ft] [hr to s] [ lb to 453g] [m/s to 3600/1609 mi/h]
R = dist_line*3.2808399*(1/3600)*453.592*    ...
        1.18*(3600/1609.344*uWind(windloop))^1.81 *10^-4;

% get list of apps with un-favourable conditions
%(ie  greater than  xx% area inside boundary)
r = uDrift*time_combo;
t1 = windDirection(windloop) + angleSpread/2;
t2 = windDirection(windloop) - angleSpread/2;
totalArea = angleSpread*r^2;

% Loop through all apps and determine if area inside boundary is
% greater than PROPINSIDE, which is the minimum acceptable
% proportion of area inside the boundary.
```

```
% This is determine if it is worthwhile to simulate the applicator,
%  or if it is just wasting CPU time for no gain.

for k= 1:size(applicators,1)

    % get info to select areaCase
    %%%%%%%%%%
    areaInside=0;  areaOutside=0;

    % start by getting points
    a = applicators(k,1);
    b = applicators(k,2);
    x1 = a + r*cos(t1);
    y1 = b + r*sin(t1);
    % get third point for wedge (p1 = p2 for circle)
    x2 = a + r*cos(t2);
    y2 = b + r*sin(t2);

    % check if any point is outside boundary
    % boundary: x_min,x_max,y_min,y_max
    out_xmin1 = (x1<x_min);
    out_xmin2 = (x2<x_min);
    out_xmin = out_xmin1 || out_xmin2;
    out_xmax1 = (x1>x_max);
    out_xmax2 = (x2>x_max);
    out_xmax = out_xmax1 || out_xmax2;
    out_ymin1 = (y1<y_min);
    out_ymin2 = (y2<y_min);
    out_ymin = out_ymin1 || out_ymin2;
    out_ymax1 = (y1>y_max);
    out_ymax2 = (y2>y_max);
    out_ymax =  out_ymax1 ||  out_ymax2;
    out = out_xmin || out_xmax || out_ymin || out_ymax;

    %%%%%%%

    % Find which case it is:
    % 1: angleSpread == 2*pi
    % 2: one out
    % 3: both out
    % 4: corner
    if out
    if angleSpread == 2*pi
        areaCase = 1;
    else

        if      ( ( (x1>x_min&&x1<x_max)&&(y1<b) &&...
        (x2>x_max)&&(y2<y_max&&y2<b) ) ||...
( (x1>x_min&&x1<x_max)&&(y1<b) &&...
        (x2>x_max)&&(y2<y_max&&y2>b) ) ||...
 ( (x1>x_min&&x1<x_max)&&(y1>b) &&...
        (x2>x_max)&&(y2<y_max&&y2>b) ) ||...
( (x1>x_min&&x1<x_max)&&(y1>b) &&...
        (x2>x_max)&&(y2<y_max&&y2<b) ) ||...
 ( (x1>a)&&(y1<y_max) &&...
        (x2>a&&x2<x_max)&&(y2>y_max) ) ||...
( (x1>a)&&(y1<y_max) &&...
        (x2<a&&x2<x_max)&&(y2>y_max) ) ||...
```

```
 ( (x1<a)&&(y1<y_max) &&...
        (x2<a&&x2<x_max)&&(y2>y_max) ) ||...
( (x1<a)&&(y1<y_max) &&...
        (x2>a&&x2<x_max)&&(y2>y_max) ) ||...
 ( (x1>x_min)&&(y1>b) &&...
        (x2<x_min)&&(y2>b) ) ||...
 ( (x1>x_min)&&(y1>b) &&...
        (x2<x_min)&&(y2<b) ) ||...
 ( (x1>x_min)&&(y1<b) &&...
        (x2<x_min)&&(y2<b) ) ||...
 ( (x1>x_min)&&(y1<b) &&...
        (x2<x_min)&&(y2>b) ) ||...
 ( (x1>x_min&&x1<a)&&(y1>x_min) &&...
        (x2<a)&&(y2<b) ) ||...
 ( (x1>x_min&&x1<a)&&(y1>x_min) &&...
        (x2>a)&&(y2<b) ) ||...
(x1>x_min&&x1>a)&&(y1>x_min) &&...
        (x2>a)&&(y2<b) ||...
( (x1>x_min&&x1>a)&&(y1>x_min) &&...
        (x2<a)&&(y2<b) ) )



%          if ~(...% if NOT (pi & p2 outside boundary)
%      (out_xmin1 && out_xmin2 && ~out_ymin && ~out_ymax) ||... % left
%      (out_xmax1 && out_xmax2 && ~out_ymin && ~out_ymax) ||... % right
%      (out_ymin1 && out_ymin2 && ~out_xmin && ~out_xmax) ||... % below
%      (out_ymax1 && out_ymax2 && ~out_xmin && ~out_xmax) ||... % above
%      ((out_ymax1 && (~out_xmin1 && ~out_xmax1)) && (out_xmin2 && (~out_ym
%      ((out_xmin1 && (~out_ymin1 && ~out_ymax1)) && (out_xmax2 && (~out_xm
%      ((out_xmax1 && (~out_ymin1 && ~out_ymax1)) && (out_ymax2 && (~out_xm
%      ((out_ymin1 && (~out_xmin1 && ~out_xmax1)) && (out_xmin2 && (~out_ym
% below left
%      );
            areaCase = 2;
        else
                if sum([out_xmax1&&out_xmax2 out_xmin1&&out_xmin2...
                        out_ymax1&&out_ymax2 out_ymin1&&out_ymin2]) == 1
                                                % only out on one boundary
                    areaCase = 3;
                else
                    areaCase = 4;
                end
        end
    end

    %%%%%%%%%
    % Now find the area corresponding to the position of the wedge
        % relative to the boundary.

    switch areaCase
        case 1
            calcArea1
        case 2
            calcArea2
        case 3
            calcArea3
```

```
        case 4
            calcArea4
        otherwise
            error('Error selecting areaCase')
    end
    %%%%%%%%%%
    %Calculate if area outside is too large in comparison to total area
        % minimum proportion of insideArea for acceptable applicator
                PROPINSIDE = .50;
                if areaInside == 0
                    propArea = (totalArea-areaOutside)/totalArea;
                else
                    propArea = areaInside/totalArea;
                end
                if double(propArea) < PROPINSIDE
                    unfavourableApp(k)=k;
                end
    end
end
    %%%%%%%%%%%%%%%%%%%%%%%%
    % loop through list of unfavouable apps and remove
        % from list of combos to be tested
    rates = logical(npermutek([0,1],size(applicators,1)));
    for k = 1:size(applicators,1)
        if unfavourableApp(k)
                % select rows to remove
            row = (rates(:,unfavourableApp(k)) == 0);
                % delete unfavourable app rates from rates matrix
            rates = rates(row,:);
        end
    end

    % set mass app rate
    rates = rates*R;

end
```

C.9 cfl.m

This script finds the timestep using the Courant-Fredrichs-Lewy condition to ensure that the model is stable. The value of `uWind(windloop)` determines the shape of the monolayer spread. The script also calculates the `angleSpread` (angle of spread) and `uDrift` (drift speed) based on the wind speed.

Listing C.9: cfl script.

```
% cfl
% CFL finds uDrift and the angleSpread given
%                                   wind speed, time & time_combo.
if  uWind(windloop) < UWINDMIN
```

```
    % circular shape, without wind stress
    % k_D = power trend line fit to experimetal data constant
    % dt = time elapsed (s)
    % n = scaling exponent (dimensionless)
    k_D = 0.1436;     % from 6m tank with 6x monolayers p77 Brink
    n = 0.7351;       % from 6m tank with 6x monolayers p77 Brink

    % get speed for Courant number by estimating uDrift
    LTIME = 10; % large time value (s)
    STIME = 1;  % small time value (s)
    DIFFTIME = LTIME - STIME; %diff
    uDrift = (k_D*LTIME^n - k_D*STIME^n)/DIFFTIME;

    courantn = 0.99;
    dt= (min(dx,dy)*courantn)/uDrift;
        % highest value of time for this step

    time = time + dt;
    if  time > time_combo
        dt = dt - (time-time_combo);
        time = time_combo;
    end
    angleSpread = 2*pi;

else
    % non-circular segment shape, with wind stress
    % Turbulence.  Above what uWind is it turbulent?
    % uWindpart = uWind + normrnd(0,0.05*uWind, 1,PARTICLES);
    uDrift = 0.0459*uWind(windloop) - 0.0661; % (m/s) p106 Brink

 if uDrift < 0 % stop speed being negative
     k_D = 0.1436;     % from 6m tank with 6x monolayers p77 Brink
    n = 0.7351;       % from 6m tank with 6x monolayers p77 Brink

    % get speed for Courant number by estimating uDrift
    LTIME = 10; % large time value (s)
    STIME = 1;  % small time value (s)
    DIFFTIME = LTIME - STIME; %diff
    uDrift = (k_D*LTIME^n - k_D*STIME^n)/DIFFTIME;
end

    courantn = 0.99;
    dt= (min(dx,dy)*courantn)/uDrift;
        % highest value of time for this step

    time = time + dt;
    if time > time_combo
        dt = dt - (time-time_combo);
        time = time_combo;
    end

     % angleSpread = 446.29*uWind^1.419; % degrees
     angleSpread =  (pi/180)* 446.29*uWind(windloop)^(-1.419); % radians
end
```

C.10 config.m

This script inputs user defined variables that control the simulation. The script is explained in 5.2.1.

Listing C.10: config script.

```
% config

% Minimum speed before circular segment shape (m/s)
UWINDMIN = 3.2 /3.6;
% Max wind speed evaporation resistance is effected
%  by monolayer application
UWINDMAX = 6.71;

 % time between weather data readings (s)
time_combo = 600;
% time_overall is 'time between wind reading' *
% 'no. of wind readings' * time_combo
np_cell = 20;        % number of particles in cell

% Grid
dx = 1;
dy = 1;
x_max = 150;
x_min = 100;
y_max = 150;
y_min = 100;

% dx = 5;
% dy = 5;
% x_max = 600;
% x_min = 100;
% y_max = 150;
% y_min = 100;


% get data from
% 'HM01X_Data_040082_30492615860123.txt'

% wind vector
 load input
 uWind = uWind(73:108)';
 windDirection = windDirection(73:108)';

%  numb = 144; % 144 readings = 24hr
%
%  % get in another format for movie
%  tempa=uWind';
%  tempb=windDirection';
%  clear uWind
%  clear windDirection
%  uWind=[]; windDirection=[];

%  for i=1:numb
%      uWind = [uWind,repmat(tempa(i),1,10)]; % 10 from 600/60 = 10
%      windDirection = [windDirection,repmat(tempb(i),1,10)];
%  end
%  %uWind = uWind';
```

```
%  %windDirection = windDirection';
%
%  % take selection of readings
%  uWind = uWind(360:720);
%  windDirection = windDirection(360:720);

% Wind (m/s) row vector
%uWind =(1/3.6)*[8,5,5,11,11,8,8,8,2,4,8,9,9,13,11,13,13,15,11,4,5,9,8,0]

% [0,..., 2*pi] where 0 = Pos X. anticlockwise = pos. (row vector)
% Imported data is from true north.

%windDirection = 90+[170,150,110,40,40,30,40,360,330,180,260,250,270,...
%  260,270,260,270,270,280,280,260,260,270,0];

windDirection = 90+ windDirection; % measured from north
% Get direction < 360 deg
windDirection(windDirection>360) = windDirection(windDirection>360)-360;
windDirection = (pi/180)*windDirection;

Temp = 15; % for evaporation of monolayer (degC)

% Applicators - one per row (x,y)
% From p146 config 8.5 a
applicators = [ 105,105;
                105,145;
                145,105;
                145,145;
                125,125];

% applicators = [ 105,125;
%                 155,125;
%                 205,125;
%                 255,125;
%                 305,125;
%                 355,125;
%                 405,125;
%                 455,125;
%                 505,125;
%                 555,125];

n_app = size(applicators,1);

          % Rates, 'app rate' based on column, 'combo' based on row (mg/s
%     rates = [50,50;
%              100,100;
%              150,150
%              ];

% initialse movie
%-------------------------
% movv = VideoWriter('monolayer2','MPEG-4');
% movv.FrameRate=15;
% open(movv)
```

C.11 degradation.m

This script finds the fractional degradation that has occurred during the time step. This is where future types of degradation can be added (due to submergence, shoreline interaction etc.) to include them in the model. The index im3 is the latest particle that has been added to the simulation.

Listing C.11: degradation script.

```
% degradation
% fractional mass loss where
% 1 = all lost and 0 = no loss
% evaporation of monolayer
dme = dt*0.073957*exp(0.14*Temp)*10^-6;
% biological degradation

% need different degradation for each particle since each has
% different age
dmb = age(1,1:im3).*1.41782*10^-6;

% total
dm(1,1:im3) = dme + dmb(1,1:im3);
```

C.12 dollarsaved.m

This script finds the amount of money saved by the application of monolayer for each permutation of applicator rates. The most profitable permutation of rates is the optimal set of application rates.

Listing C.12: dollarsaved script.

```
% dollar saved
%-------------------------------------------------------
% Find savings for each combination of rates
%-------------------------------------------------------

% saving = $/amount of water * amountsaved
% saving = 500/ML* no. of ML %% t = t for this step %%%

% E_diff/(1e3*3600*24) -- from mm/day to m/s
%E_diff(rate, m/s) * time(s) * area(m^2) = volume (m^3)
amountsaved =  E_diff/(1e3*3600*24)*time_combo* (dx*dy);
%$ during timestep t
saving(windloop,combo) = sum(sum(1e3 * amountsaved/1e3));

% cost = amount applied *($/g) + running cost
cost(windloop,combo) = sum(mass_app(combo,:))*10e-2  + 0;

```

```
% $ during timestep t
profit(windloop,combo) = saving(windloop,combo) - cost(windloop,combo);
```

C.13 `evap.m`

This function finds the evaporation with and without the monolayer present on the water surface. It uses the equations developed by McJannet, Webster, Stenson and Sherman (2008) presented in Appendix E: Evaporation Model Equations.

Listing C.13: evap function.

```
function [E_diff] = evap(nr_m)
% Evaporation Model for monolayer savings
% McJannet, Webster, Stenson, Sherman 2008
% Appendix B: Model algorithm - Penman-Monteith equation

% Format convention:
% variable, variable_sub/superscript, variable_superscript_subscript

% Input
% Constants
albedo = 0.08;  % albedo of water, MJ.kg^-1
C_a = 0.001013; % specific heat of air, MJ.kg^-1.K^-1
C_w = 0.004185; % specific heat of water, MJ.kg^-1.K^-1
rho_a = 1.2;    % density of air, kg.m^-3
rho_w = 1000;   % density of water, kg.m^-3
sigma = 4.8E-9; % Stefan-Boltzmann constant, MJ.m^-2.K^-4.d^-1
T_w0 = 20;      % temperature of water at previous time step

% Variables
A = 0.01;       % water body area, km^2
K_down = 12;    % total daily incoming short wave radiation, MJ.m^-2.d^-1
T_a = 20;       % mean daily air temperature, degC
U_10 = 3;       % mean daily wind speed at 10m, m.s^-1

T_air = 25;
% vapour pressure at air temp, kPa - antoine equation,
% wikipedia, water 1->100degC
e_a = (10^(8.07131-(1730.63/(T_air+233.426))))*(101.325/760);

Z = 15;         % water body depth, m
psi = 50   ;    % water body altitude, m
phi = 23*pi/180;% water body latitude, radians
J = 11;         % day of the year
%r_m = 0;       % monolayer resistance, s.m^-1
r_m = nr_m;

% Calculations
% latent heat of vaporisation, MJ.kg^-1 (eq 8)
lambda = 2.501-T_a*2.361E-3;
% psychometric constant, kPa.degC^-1 (eq 9)
gamma = C_a*100/0.622*lambda;

% wind function, MJ.m^-2.d^-1.kPa^-1 (eq 11)
```

```
fu = ((5/A)^0.05)*(3.80+1.57*U_10);
% aerodynamic resistance, s.m^-1 (eq 10)
r_a = (rho_a*C_a)/(gamma*fu/86400);

% Get L_DOWN
% inverse relative distance Earth-Sun (eq 21)
d_r = 1+0.033*cos(2*pi*J/365);
% solar decimation (eq 20)
delta = 0.409*sin(2*pi*J/365-1.39);
% x-factor (eq 19)
X = 1 - ((tan(phi))^2)*((tan(delta))^2);
% sunset hour angle (eq 18)
omega_s = pi/2 - atan(-tan(phi)*tan(delta)/sqrt(X));
% extraterrestrial short wave radiation, MJ.m^-2.d^-1 (eq 17)
K_et =  (24*60/pi)*0.082*d_r*(omega_s*sin(phi)*sin(delta)+...
    cos(phi)*cos(delta)*sin(omega_s));
% clear sky short wave radiation, MJ.m^-2.d^-1 (eq 16)
K_clear = (0.75+2E-5*psi)*K_et;
% ratio of incoming short wave to clear sky short wave radiation (eq 15)
K_ratio = K_down/K_clear;
% fraction of cloud cover (0-->1) (eq 14)
if K_ratio < 0.9
    C_f  = 1.1 -K_ratio;
else
    C_f = 2*(1-K_ratio);
end
% incoming long wave radiation, MJ.m^-2.d^-1 (eq 13)
L_down = (C_f+(1-C_f)*(1-(0.261*exp(-7.77E-4*T_a^2))))*...
                sigma*(T_a+273.15)^4;

% Get L_UP
% dew point temperature, degC (eq 26) (remember log(x) = ln(x) in matlab)
T_d = (116.9+237.2*log(e_a))/(16.78-log(e_a));
% wet bulb temperature, degC (eq 25)
T_n = (0.00066*100*T_a+(4098*e_a/(T_d+237.3)^2)*T_d) /...
    (0.00066*100+(4098*e_a/(T_d+237.3)^2));
% outgoing long wave radiation at wet bulb temp, MJ.m^2.d^-1 (eq 29)
L_up_n = sigma*(T_a+273.15)^4+(4*sigma*(T_a+273.15)^3)*(T_n-T_a);
% net radiation at wet bulb temp, MJ.m^-2.d^-1 (eq 28)
Q_asterisk_n = K_down*(1-albedo)+(L_down-L_up_n);
% slope of the temperature saturation water vapour curve at...
    % wet bulb temp, kPa.degC^-1 (eq 27)
delta_n = (4098*(0.6108*exp(17.27*T_n/(T_n+237.3)))) / ((T_n+237.3)^2);
% equilibrium temperature, degC (eq 24)
T_e = T_n + Q_asterisk_n/(4*sigma*(T_n+273.15)^3+fu*(delta_n+gamma));
% time constant, d
tau = (rho_w*C_w*Z)/(4*sigma*(T_n+273.15)^3+fu*(delta_n+gamma));
% water temperature, degC (eq 23)
T_w = T_e + (T_w0-T_e)*exp(-1/tau);
% outgoing long wave radiation at water temp, MJ.m^-2.d^-1 (eq 22)
L_up = 0.97*sigma*(T_w+273.15)^4;

% net radiation, MJ.m^-2.d^-1 (eq 12)
Q_asterisk = K_down*(1-albedo)+(L_down-L_up);

% change in heat storage in water body, MJ.m^-2.d^-1
```

```
N = rho_w*C_w*Z*(T_w-T_w0);
% saturated vapour pressure at water temp, kPa (eq 32)
e_asterisk_w = 0.6108*exp(17.27*T_w/(T_w+237.3));
% slope of the temperature saturation water vapour curve at water temp,...
    % kPa.degC^-1
delta_w = (4098*(0.6108*exp(17.27*T_w/(T_w+237.3)))) / ((T_w+237.3)^2);

% evaporation, mm.d^-1 (eq 7)
E_with = (1/lambda)*( (delta_w*(Q_asterisk-N)+86400*rho_a*C_a*...
    (e_asterisk_w-e_a)/r_a+r_m)/(delta_w+gamma));
E_without = (1/lambda)*( (delta_w*(Q_asterisk-N)+86400*rho_a*C_a*...
    (e_asterisk_w-e_a)/r_a+0)/(delta_w+gamma));
E_diff = abs(E_without-E_with);
end
```

C.14 Monolayer_Simulation.m

This script is the main file from which all the other scripts and functions are called.

Listing C.14: Monolayer_Simulation script.

```
% Monolayer Simulation
clear, clc
tic
config
preallocate
%location=0;
% Loop through all wind speeds(and directions)

% comment out to make sure different rand numbers are used
%s = rng('default');
disp('preallocated, starting simulation')

for windloop = 1:size(uWind,2)
    combo=1;
    % Reseed rand based on current time, and save state
    % comment out to test effect of varying dx,dy or np_cell
    s = rng('shuffle');

    % find rates size to find combo max
    calcrates
    mass_app = zeros(size(rates,1),size(rates,2));
    combo_max = size(rates,1);

    % Initialise bestprofit with large negative number to ensure
    % it isn't used
    bestprofit = -1e6;

    % Loop through different applicator rates

    while combo <= combo_max
        calcmoveevapdollar

        % Check if this combo is the best so far
        if combo==1 || (profit(windloop,combo)>bestprofit)
            % Store best profit
```

```
                bestprofit = profit(windloop, combo);
                % Set most profitable arrays (p)
                Xp = X;
                Yp = Y;
                massp = mass;
                pactp = pact;
                agep = age;
                pact_xp = pact_x;
                pact_x1p = pact_x1;
                pact_x2p = pact_x2;
                pact_yp = pact_y;
                pact_y1p = pact_y1;
                pact_y2p = pact_x2;
                totalmass_p = totalmass;
                if windloop == size(uWind,2) % if last wind condition
                    nr_m_p = nr_m; % average coverage
                end
            end

            combo=combo+1;
        end

        %keep tc counter for next iteration
        tc=tc+cc;

        % Save best arrays(p) as current (c). Export any data
        Xc = Xp;
        Yc = Yp;
        massc = massp;
        pactc = pactp;
        agec = agep;
        pact_xc = pact_xp;
        pact_x1c = pact_x1p;
        pact_x2c = pact_x2p;
        pact_yc = pact_yp;
        pact_y1c = pact_y1p;
        pact_y2c = pact_x2p;
        totalmass_c = totalmass_p;

        time_overall = time_overall + time;
        disp(windloop)

        %-------------------------
        % Make a Movie!
        %-------------------------
%       % fullscreen = get(0,'ScreenSize');
%       figure%('Position',[0 0 560 420])
%       handle = scatter(Xc,Yc,'.','blue'); title('Particles'), % m
%       hold on
%       % Plot applicators
%       scatter(applicators(:,1),applicators(:,2),'s','filled',...
%                   'sizeData',10^2)
%       hold on
%       % Plot rectangle to show boundary
%       rectangle('Position', [x_min,y_min,x_max,y_max])
%       set(gca,'XLim',[x_min-5 x_max+5]), xlabel('metres')
%       set(gca,'YLim',[y_min-5 y_max+5]), ylabel('metres')
%
%       set(gcf,'Renderer','zbuffer');
%       writeVideo(movv,getframe(gcf));
%       close

        %-------------
```

```
100

end

% Total profit: sum of best profits for each windloop
bestprofit = max(profit,[],2);
totalprofit = sum(max(profit,[],2));
toc
%graph
%scatter(Xc,Yc,'.'), axis([x_min-30 30+x_max y_min-5 5+y_max])
%disp('total profit: '), disp(totalprofit)
% contour(nr_m_p, linspace(0, 10e-3, 10))


% save output
%
% b403 = bestprofit;
% n403 = nr_m_p;
% Xc403 = Xc;
% Yc403 = Yc;
% save('results403.mat',...
%      'b403','n403','Xc403','Yc403')

% Close this movie
            %close(movv);
```

C.15 movepart.m

This script calculates the positions, age and mass (due to degradation) of the new and old particles. Old particles are particles that were created on a previous time step and new particles originate from the applicator positions. The boundary script checks if any particles have crossed the boundary and if so sets `pact` (particle active) to false to prevent any movement in the future. The shoreline script will need to set `pact` to true for the respective particles after it calculates which particles will return to the surface of the water.

Listing C.15: movepart script.

```
% movepart
%=========================
%  Move particles
5
%
% r = a + (b-a).*rand(n,1);
% r= random number
% a,b = interval from 0.5 to -0.5
10 % r = 0.5 + (-0.5-0.5).*rand(np_cell,1)
```

```
% r = 0.5 - rand(np_cell,1)

% index for old particles
k = n_app;
im3 = (cc-1)*n_app*np_cell + k*np_cell + tc*n_app*np_cell;

degradation

% old particles
% if old particles exist
if cc>1 || tc~=0
    % theta old
    to(1:im3) = windDirection(windloop) + (0.5-rand(1,im3)) *angleSpread;

    % new position = old position + pact .* rand*dt*speed * ...
        %        trig(rand within wind+spread)
    X(1,1:im3)= X(1:im3) + pact(1,1:im3) .* rand(1,im3)*dt*uDrift .*...
                cos(to(1:im3));
    Y(1,1:im3)= Y(1:im3) + pact(1,1:im3) .* rand(1,im3)*dt*uDrift .*...
                sin(to(1:im3));
    age(1,1:im3) =  age(1,1:im3)+dt;
    mass(1,1:im3) = mass(1,1:im3) .* (1-dm(1,1:im3));
end

% new particles
for k=1:n_app
    %theta new
    if rates(combo,k)~=0 % don't move particles if they have no mass
        tn(1:np_cell) = windDirection(windloop) +...
                        (0.5-rand(1,np_cell)) *angleSpread;
        lhs = 1 + (cc-1)*n_app*np_cell + (k-1)*np_cell...
                        + tc*n_app*np_cell;
        rhs = (cc-1)*n_app*np_cell + k*np_cell + tc*n_app*np_cell;

        X(1, lhs:rhs) = applicators(k,1) + rand(1,np_cell)*dt*uDrift...
                        .* cos(tn);
        Y(1, lhs:rhs) = applicators(k,2) + rand(1,np_cell)*dt*uDrift...
                        .* sin(tn);
        age(1, lhs:rhs) = dt;
        mass(1, lhs:rhs) = (dt*rates(combo,k)/np_cell) .*...
                        (1-dm(1,lhs:rhs));
        totalmass = totalmass + (dt*rates(combo,k)/np_cell);
    end
end

boundary
```

C.16 npermutek.m

This function was written by Matt Fig and is available from `http://www.mathworks.com/matlabcentral/fileexchange/11462`. The function calculates a matrix containing the all the possible permutations. There are two required inputs; firstly the mass

rates (e. g. [0 1]), and secondly the number of applicators.

Listing C.16: npermutek function.

```
function [Matrix,Index] = npermutek(N,K)
%NPERMUTEK Permutation of elements with replacement/repetition.
% MAT = NPERMUTEK(N,K) returns all possible samplings of length K from
% vector N of type: ordered sample with replacement.
% MAT has size (length(N)^K)-by-K, where K must be a scalar.
% [MAT, IDX] = NPERMUTEK(N,K) also returns IDX such that MAT = N(IDX).
% N may be of class: single, double, or char.  If N is single or double,
% both MAT and IDX will be of the same class.
%
% For N = 1:M, for some integer M>1, all(MAT(:)==IDX(:)), so there is no
% benefit to calling NPERMUTEK with two output arguments.
%
% Examples:
%          MAT = npermutek([2 4 5],2)
%
%  MAT =
%
%        2     2
%        2     4
%        2     5
%        4     2
%        4     4
%        4     5
%        5     2
%        5     4
%        5     5
%
% NPERMUTEK also works on characters.
%
%          MAT = npermutek(['a' 'b' 'c'],2)
%  MAT =
%
%       aa
%       ab
%       ac
%       ba
%       bb
%       bc
%       ca
%       cb
%       cc
%
% See also perms, nchoosek
%
% Also on the web:
% http://mathworld.wolfram.com/BallPicking.html
% See the section on Enumerative combinatorics below:
% http://en.wikipedia.org/wiki/Permutations_and_combinations
% Author:  Matt Fig
% Contact:  popkenai@yahoo.com

if nargin ~= 2
    error('NPERMUTEK requires two arguments. See help.')
end

if isempty(N) || K == 0,
```

```
    Matrix = [];
    Index = Matrix;
    return
elseif floor(K) ~= K || K<0 || ~isreal(K) || numel(K)~=1
    error('Second argument should be a real positive integer. See help.')
end

LN = numel(N);  % Used in calculating the Matrix and Index.

if K==1
    Matrix = N(:); % This one is easy to calculate.
    Index = (1:LN).';
    return
elseif LN==1
    Index = ones(K,1);
    Matrix = N(1,Index);
    return
end

CLS = class(N);

if ischar(N)
    CLS = 'double';  % We will deal with this at the end.
end

L = LN^K;  % This is the number of rows the outputs will have.
Matrix = zeros(L,K,CLS);  % Preallocation.
D = diff(N(1:LN));  % Use this for cumsumming later.
LD = length(D);  % See comment on LN.
VL = [-sum(D) D].';  % These values will be put into Matrix.
% Now start building the matrix.
TMP = VL(:,ones(L/LN,1,CLS));  % Instead of repmatting.
Matrix(:,K) = TMP(:);  % We don't need to do two these in loop.
Matrix(1:LN^(K-1):L,1) = VL;  % The first column is the simplest.

if nargout==1
    % Here we only have to build Matrix the rest of the way.
    for ii = 2:K-1
        ROWS = 1:LN^(ii-1):L;  % Indices into the rows for this col.
        TMP = VL(:,ones(length(ROWS)/(LD+1),1,CLS));  % Match dimension.
        Matrix(ROWS,K-ii+1) = TMP(:);  % Build it up, insert values.
    end

else
    % Here we have to finish Matrix and build Index.
    Index = zeros(L,K,CLS);  % Preallocation.
    VL2 = ones(size(VL),CLS);  % Follow the logic in VL above.
    VL2(1) = 1-LN;  % These are the drops for cumsum.
    TMP2 = VL2(:,ones(L/LN,1,CLS));  % Instead of repmatting.
    Index(:,K) = TMP2(:);  % We don't need to do two these in loop.
    Index(1:LN^(K-1):L,1) = 1;

    for ii = 2:K-1
        ROWS = 1:LN^(ii-1):L;  % Indices into the rows for this col.
        F = ones(length(ROWS)/(LD+1),1,CLS);  % Don't do it twice!
        TMP = VL(:,F);  % Match dimensions.
        TMP2 = VL2(:,F);
        Matrix(ROWS,K-ii+1) = TMP(:); % Build them up, insert values.
        Index(ROWS,K-ii+1) = TMP2(:);
    end
```

```
    Index(1,:) = 1;  % The first row must be 1 for proper cumsumming.
    Index = cumsum(Index);  % This is the time hog.
end

Matrix(1,:) = N(1);  % For proper cumsumming.
Matrix = cumsum(Matrix);  % This is the time hog.

if ischar(N)
    Matrix = char(Matrix);  % char was implicitly cast to double above.
end

% Matt Fig
% 19 Jun 2006 (Updated 11 May 2009)
% http://www.mathworks.com/matlabcentral/fileexchange/11462
```

C.17 preallocate.m

This script preallocates memory for the large matrices required for the simulation, drastically reducing the time required to run the model.

Listing C.17: preallocate script.

```
% preallocate

% Initialize variables
nX_max = ceil( (x_max-dx/2) / dx )+1;
nY_max = ceil( (y_max-dy/2) / dy )+1;
nX_min = ceil( (x_min-dx/2) / dx )+1;
nY_min = ceil( (y_min-dy/2) / dy )+1;

tc = 0; % total counter
cc = 0; % combination counter

totalmass = 0;

% Calc size of arrays
P_SIZE=0; a=0;
for windloop = 1:length(uWind)
    time=0;
    a=a+1;
    while time < time_combo
        cfl
        P_SIZE = P_SIZE + np_cell*n_app;
    end
end

% combo=1;
% calcrates
% profit=zeros(a,size(rates,1));

% reset time
time = 0;
time_overall=0;

%Preallocate arrays
Xc = ones(1,P_SIZE); % create Xc & Yc outside boundary
Yc = ones(1,P_SIZE);
```

```
massc = zeros(1,P_SIZE);
agec = zeros(1,P_SIZE);
pact_xc = false(1,P_SIZE);
pact_x1c = false(1,P_SIZE);
pact_x2c = false(1,P_SIZE);
pact_yc = false(1,P_SIZE);
pact_y1c = false(1,P_SIZE);
pact_y2c = false(1,P_SIZE);
pactc = true(1,P_SIZE);
totalmass_c = 0;

to=zeros(1,P_SIZE);
tn=zeros(1,np_cell);

dmb = zeros(1,P_SIZE);
dm = zeros(1,P_SIZE);
```

C.18 inputdata.txt

This is the data used for the simulations. It is the first 144 values from `HM01X Data 040082 30492615860123.txt` (see line 33 in `config.m`) where each reading is taken every 10 minutes giving a total of 24 h. The first column is the wind speed [m/s] and the second column is the direction in degrees from north in an anti-clockwise direction.

Listing C.18: inputdata text file.

```
5.2489   266.8
4.603    249.2
4.122    259.4
4.216    247.2
5.4002   255.3
4.785    264
4.3082   237.3
3.5272   240.1
4.7273   252.3
3.7672   231.1
3.9488   227.5
4.736    288.7
4.0827   229
4.6778   260.4
2.7887   234.8
3.9173   261.6
5.0185   254.5
4.1885   239.3
4.0487   243.9
4.3242   260.7
5.0732   265.4
3.328    246.5
4.5878   251.6
4.7255   262.2
3.4422   258.1
3.0828   251.4
3.059    243
2.5278   250.4
2.0803   238
```

```
1.6777  248.7
1.5685  259.7
1.3668  247.7
1.583   249.9
1.5067  244.7
1.6492  240.2
1.8235  244.9
1.2929  237.7
1.0343  193.6
1.3812  175.3
1.6945  215.6
1.729   191.8
0.70633 160.4
0.37817 153.2
0.539   175.4
0.45233 225.1
0.66067 216.9
2.122   296
1.7732  243.6
1.051   110.8
1.1952  176.3
0.78867 196.6
0.93417 180.5
1.2958  243.4
0.66383 127.4
0.49533 218.5
0.62367 127.1
0.54783 207.6
0.91317 180
1.2512  170
1.5377  173.8
0.76517 172.2
0.61133 152.2
0.4855  153.5
0.61383 173
0.70833 253
0.47167 156.2
0.55683 162.6
0.85033 208.7
0.719   167.9
0.48883 174.8
0.41067 114.5
0.40917 143.4
0.70083 174.8
0.847   188.3
0.6785  178.4
0.48017 155.3
0.4375  233.3
0.43733 221.5
0.38483 236.6
0.836   269.7
0.73017 208.4
0.60333 256.4
1.523   278.8
1.1555  286.8
1.457   320.2
1.731   320.2
1.5608  304.1
1.6727  298.4
1.5402  295.4
1.3078  305
0.985   301.4
0.85133 209
0.7525  183.7
0.75517 157.4
0.66217 148.7
```

```
0.62517  143.7
0.84467  205.7
0.78367  198.7
0.81867  187.8
0.66933  193.9
0.50633  195.4
0.5115   143.7
0.50183  136
0.55817  141.1
0.42633  123.6
0.5235   142.5
0.57517  218.6
0.697    316.3
1.536    324.7
1.9292   318.5
1.947    297.8
1.2323   244.8
0.72183  177
0.729    166.3
0.54717  176.1
0.43717  257.2
0.45083  305.9
0.46117  257.4
0.62517  306.2
0.53067  312.2
0.43533  257.9
0.38033  296.3
0.57017  248.1
1.2727   320.7
0.56467  251
0.4185   229.5
0.45     280.6
0.51067  325
0.4575   299.4
0.83083  281.4
0.59333  225.3
0.778    125.2
0.79983  117.5
1.0402   130.6
0.88783  110
1.3507   107.9
1.8875   121.7
1.1547   156.8
1.3272   112
1.2477   145.7
1.5367   246.7
1.2857   137.6
1.2262   133.3
1.1443   180.9
```

Appendix D

Preliminary Ideas

This document was prepared at the start of the project was used as a starting point. It also has ideas for extending the project further. Please note that this document does not reflect the final project - see Chapter 5: Sample Run for a walk through of the code in Appendix C: Source Code.

D.1 Boundaries

There are several possible ways to input the boundaries:

- Satellite photographs. Need to stitch together photos, convert to b & w, filter noise, take coordinates of the edge, and calculate scale. Processing, filtering and stitching together photos is too difficult.
- GPS. Enter coordinates and convert to 2D. Overlay a grid. Possible to have North direction vector making it easier to add the weather data.
- Graphically pick points. Clink on boundaries on a) photo b) generic grid. Both a) and b) require an image to be imported and then a list of coordinates is created based on the position of the mouse click. The data then needed to be scaled and a grid created.

- Enter list of points using keyboard; create grid
- Use equations to describe boundaries and then points are extracted based on the resolution required.

The two best ways are GPS for real world locations and a list of points entered via keyboard for testing purposes.

To convert form geodetic to ECEF (Cartesian) the following equations are required:

$$X = (N(\phi) + h)cos\phi cos\lambda \tag{D.1}$$

$$Y = (N(\phi) + h)cos\phi sin\lambda \tag{D.2}$$

$$Z = (N(\phi)(1 - e^2) + h)sin\lambda \tag{D.3}$$

where

$$N(\phi) = \frac{a}{\sqrt{1 - e^2 sin^2\phi}} \tag{D.4}$$

and a = semi major axis
e^2= square to the first numerical eccentricity of ellipsoid
first eccentricity, $e = \sqrt{1 - \frac{b^2}{a^2}}$

ECEF has the following axes, with the origin in the centre of the planet. X is defined as the intersection of the Greenwich meridian and the equator; Z is the mean spin axis of the planet positive to the north; and the Z axis completes the right hand system.

The Aerospace toolbox in Simulink provides blocks to convert from geodetic to ECEF and back again but it is easy to convert from geodetic to ECEF as it involves direct substitution.

One way to import the GPS coordinates into a 2D grid for MATLAB:

- Convert from geodetic to ECEF

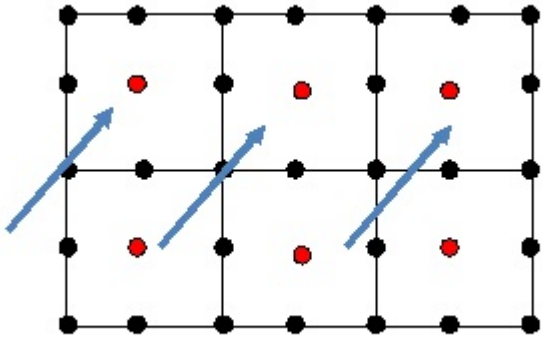


Figure D.1: Grid showing how particles move between different grid squares.

- Convert 3D collection of points into a 2D system by:
  - Create a plane using any random 3 points
  - Move the points on the plane by using a vector between the origin and the point.
  - Rotate the plane so that there are only (x,y) coordinates.

D.2 Grid

The red circle represents the average value for the area in the square. This assumption requires that the area is small compared to the overall area and the time step of the simulation is small.

D.3 Applicator information

D.3.1 Applicator Positions (AP)

These are the different ways that the applicators can be positioned on the water. The simulation will compare the different positions and it will then be possible to try to identify trends that can be used to place the applicators.

Applicator	Position			
	1		2	
	x	y	x	y
1				
2				

Table D.1: Applicator Rates

D.3.2 Applicator Rates Combinations (ARC)

The applicator rates combinations are evaluated for each time step to determine which combination has the best $ savings. The combination that offers the highest saving is then used for that time step and is used to modify the mesh for the next time step. Each column in the table represents a different simulation.

D.4 Trial

Each trail will represent a different time e.g. different weeks in a year when the different applicator positions are investigated. Each trail is a repeat of all the possible applicator positions.

Applicator	Simulation		
	1	2	3
1	x_1	x_2	x_3
2	x_1	x_1	x_1
3	x_1	x_1	x_1
4	x_1	x_1	x_1
5	x_1	x_1	x_1

Table D.2: Applicator Rates Combinations. The rates x_1, x_2 and x_3 are measured in kg/s.

D.4.1 Current weather

The current weather is the result of the Monte Carlo Simulation during the previous time step.

D.4.2 Future weather

This is the result of the current Monte Carlo Simulation. This function will use a probability density table and a random number to find the weather conditions. The weather conditions include wind speed and direction as well as rainfall. It is necessary to use the weather conditions for the different applicator rates (since they all occur at the same time) as well as the applicator positions in order to provide a fair comparison. Different weather conditions will be achieved in different trials.

D.4.3 Monolayer Behaviour Functions

No. 2-5 act to reduce the concentration.

Movement of Monolayer- direction and distance

There will a smaller time step used to calculate the movement of the monomer.

Shoreline Absorption

Volatilisation

This is the evaporation of monolayer.

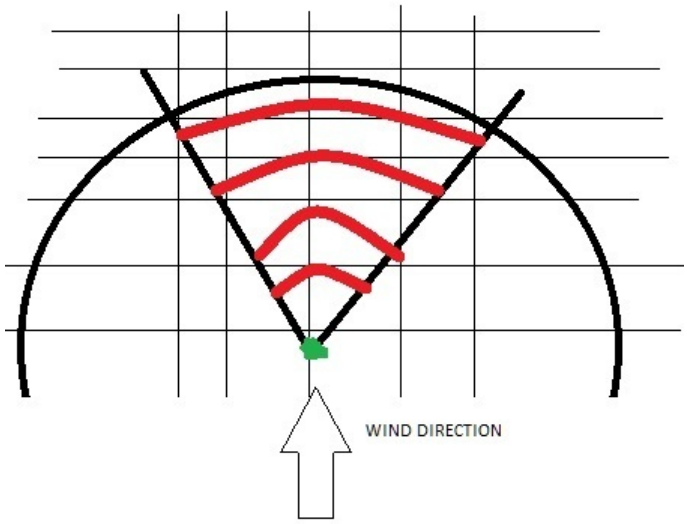


Figure D.2: Monolayer spreading under the influence of the wind. Each red arcs represents a group of particles that were released at the same time. The location of each particle can then be identified to a particular grid square given the angle of spread and the speed. Once identified with a square of the grid the particles can be summed with the representative point at the centre of the square (See 2: Grid).

Submergence

Submergence is the loss of the monolayer due to it breaking up and sinking below the surface of the water. This is caused by waves and rainfall.

Biological degradation - aging

D.4.4 Percent coverage

The percent coverage is calculated at the end of each time step for each ARC

D.4.5 Saving

The saving = value of water saved - (cost of chemical + fixed costs)

D.4.6 Select best ARC for current time step

The ARC with the best saving is then saved and used for the next time step by modify the grid. The other results are discarded.

D.4.7 Complete for all time values for AP simulation

Continue to solve for applicator positions. Once completed solve for next possible applicator position until all positions have been evaluated.

D.5 Results

The results can then be used to identify trends to aid in the placement of the applicators. The following information will be required:

- The application rates (kg/s), % coverage and $ saving for each applicator position
- The applicator positions
- A method of comparing the two above points.

The easiest way to compare the different applicator positions is to find the position that saves the most money per year. This involves simulating a variety of weather conditions that are representative of best, worst case and average conditions. It will also be important to ensure the simulations are run for long enough to ensure the applicator positions are adequate for all these conditions since it is impractical to have different positions for different seasons.

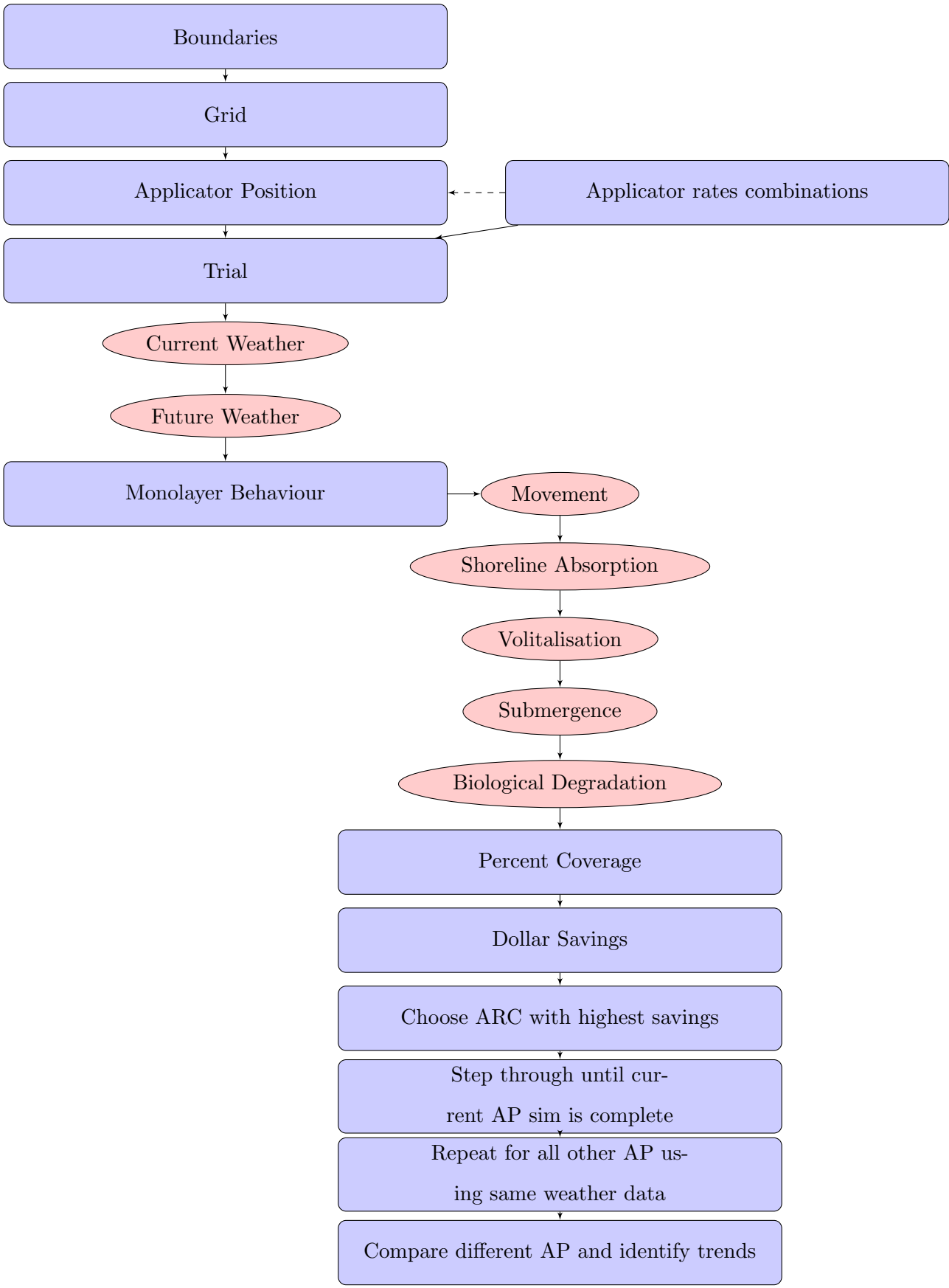


Figure D.3: Layout for simulation code

Appendix E

Evaporation Model Equations

Source: McJannet, DL Webster, IT Stenson, MP Sherman, BS 2008, Estimating open water evaporation for the murray-darling basin: Appendix B: Model algorithms, CSIRO murray-darling basin sustainable yields project.

10 Appendix B: Model algorithms

Evaporation (E in mm d^{-1}) from a water body can be estimated using the Penman-Monteith equation (Monteith, 1965):

$$E = \frac{1}{\lambda}\left(\frac{\Delta_w(Q^* - N) + 86400\rho_a C_a(e_w^* - e_a)/r_a}{\Delta_w + \gamma}\right) \quad \text{Equation 7}$$

where ρ_a is density of air (kg m^{-3}) and C_a is specific heat of air (MJ kg^{-1} $^oK^{-1}$).

Latent heat of vaporisation, λ (MJ kg^{-1}), at air temperature, T_a (℃), is calculated as follows:

$$\lambda = 2.501 - T_a 2.361\times10^{-3} \quad \text{Equation 8}$$

The psychometric constant, γ (kPa ℃ $^{-1}$) is calculated from:

$$\gamma = \frac{C_a 100}{0.622\lambda} \quad \text{Equation 9}$$

Aerodynamic resistance, r_a (s m^{-1}), is calculated using the following equation (Calder and Neal, 1984):

$$r_a = \frac{\rho_a C_a}{\gamma(f(u)/86400)} \quad \text{Equation 10}$$

The wind function, $f(u)$ (MJ m^{-2} d^{-1} kPa^{-1}), is calculated from wind speed at 10 m, U_{10} (m s^{-1}), and area, A (km^2), (Sweers, 1976):

$$f(u) = \left(\frac{5}{A}\right)^{0.05}(3.80 + 1.57U_{10}) \quad \text{Equation 11}$$

Net radiation, Q^* (MJ m^{-2} d^{-1}), is calculated using solar radiation inputs, $K\downarrow$ (MJ m^{-2} d^{-1}), as follows:

$$Q^* = K\downarrow(1-\alpha) + (L\downarrow - L\uparrow) \quad \text{Equation 12}$$

Incoming long-wave radiation, $L\downarrow$ (MJ m^{-2} d^{-1}), is calculated from the equations of Oke (1987) and Idso and Jackson (1969):

$$L\downarrow = (C_f + (1-C_f)(1-(0.261\exp(-7.77\times10^{-4}T_a^{\,2}))))\sigma(T_a + 273.15)^4 \quad \text{Equation 13}$$

Equations 14 though 21 are used to calculate $L\downarrow$. Fraction of cloud cover (value from 0 to 1 with 1 being 100% cover) is calculated using the following equation (Jegede et al., 2006):

If K_{Ratio} ≤ 0.9 then use $C_f = 1.1 - K_{Ratio}$ Equation 14

If K_{Ratio} > 0.9 then use $C_f = 2(1 - K_{Ratio})$

Ratio of incoming short-wave radiation to clear sky short-wave radiation (K_{Ratio}) is calculated from:

$$K_{Ratio} = \frac{K\downarrow}{K_{Clear}}$$ Equation 15

Clear sky short-wave radiation (K_{Clear} in MJ m^{-2} d^{-1}) is calculated using water body altitude (ψ in m) as follows (Allen et al., 1998):

$$K_{Clear} = (0.75 + 2\times10^{-5}\psi)K_{ET}$$ Equation 16

Extraterrestrial short-wave radiation (K_{ET} in MJ m^{-2} d^{-1}) is calculated using latitude (φ in radians) as follows:

$$K_{ET} = \frac{24(60)}{\pi}0.082d_r(\varpi_s \sin(\varphi)\sin(\delta) + \cos(\varphi)\cos(\delta)\sin(\varpi_s))$$ Equation 17

Sunset hour angle, ϖ_s, is calculated from:

$$\varpi_s = \frac{\pi}{2} - \arctan\left(\frac{-\tan(\varphi)\tan(\delta)}{X^{0.5}}\right)$$ Equation 18

The X-factor, X, is calculated from:

$$X = 1 - (\tan(\varphi))^2(\tan(\delta))^2$$ Equation 19

Solar decimation, δ, is calculated using the day of the year, J, as follows:

$$\delta = 0.409\sin\left(\frac{2\pi}{365}J - 1.39\right)$$ Equation 20

The inverse relative distance Earth-Sun, d_r, is calculated using:

$$d_r = 1 + 0.033\cos\left(\frac{2\pi}{365}J\right)$$ Equation 21

Outgoing long-wave radiation at water temperature ($L\uparrow$ in MJ m^{-2} d^{-1}) is calculated using the Stefan-Boltzmann constant, σ (MJ m^{-2} ^{o}K^{-4} d^{-1}), as follows:

$$L\uparrow = 0.97\sigma(T_w + 273.15)^4$$ Equation 22

Equations 23 through 29 are used to calculate $L\uparrow$. Water temperature, T_w (℃), is calculated from the following equation (de Bruin, 1982):

$$T_w = T_e + (T_{w0} - T_e)\exp(-1/\tau)$$ Equation 23

Equilibrium temperature, T_e (℃), is calculated from the following equation (d e Bruin, 1982):

$$T_e = T_n + \frac{Q^*_n}{4\sigma(T_n + 273.15)^3 + f(u)(\Delta_n + \gamma)}$$ Equation 24

Wet-bulb temperature, T_n (℃), is calculated using vapour pressure, e_a (kPa), as follows (Jensen et al., 1990):

$$T_n = \frac{0.00066 \times 100 T_a + (4098 e_a / (T_d + 237.3)^2) T_d}{0.00066 \times 100 + (4098 e_a / (T_d + 237.3)^2)}$$ Equation 25

Dew point temperature, T_d (℃), is calculated from:

$$T_d = \frac{116.9 + 237.3 Ln(e_a)}{16.78 - Ln(e_a)}$$ Equation 26

Slope of the temperature saturation water vapour curve at wet bulb temperature, Δ_n (kPa ℃ $^{-1}$), is calculated from:

$$\Delta_n = \frac{4098\left[0.6108 \exp\left(\frac{17.27 T_n}{(T_n + 237.3)}\right)\right]}{(T_n + 237.3)^2}$$ Equation 27

Net radiation at wet-bulb temperature, Q^*_n (MJ m^{-2} d^{-1}), is calculated using albedo, α , as follows:

$$Q^*_n = K\downarrow(1-\alpha) + (L\downarrow - L\uparrow_n)$$ Equation 28

Outgoing long-wave radiation at wet-bulb temperature, $L\uparrow_n$ (MJ m^{-2} d^{-1}), is calculated from:

$$L\uparrow_n = \sigma(T_a + 273.15)^4 + 4\sigma(T_a + 273.15)^3 (T_n - T_a)$$ Equation 29

The time constant (τ in days) is calculated using the density of water (ρ_w in kg m^{-3}), specific heat of water (C_w in MJ kg^{-1} ^{o}K^{-1}), and depth of water (Z in m) as follows (de Bruin, 1982):

$$\tau = \frac{\rho_w C_w Z}{4\sigma(T_n + 273.15)^3 + f(u)(\Delta_n + \gamma)}$$ Equation 30

Change in heat storage in the water body, N (MJ m^{-2} d^{-1}), is calculated from:

$$N = \rho_w C_w Z (T_w - T_{w0})$$ Equation 31

Saturated vapour pressure at water temperature, e_w^* (kPa), is calculated from:

$$e_w^* = 0.6108 \times \exp\left(\frac{17.27 T_w}{T_w + 237.3}\right)$$ Equation 32

Slope of the temperature saturation water vapour curve at water temperature, Δ_w (kPa ℃$^{-1}$), is calculated from:

$$\Delta_w = \frac{4098\left[0.6108 \times \exp\left(\frac{17.27 T_w}{(T_w + 237.3)}\right)\right]}{(T_w + 237.3)^2}$$ Equation 33