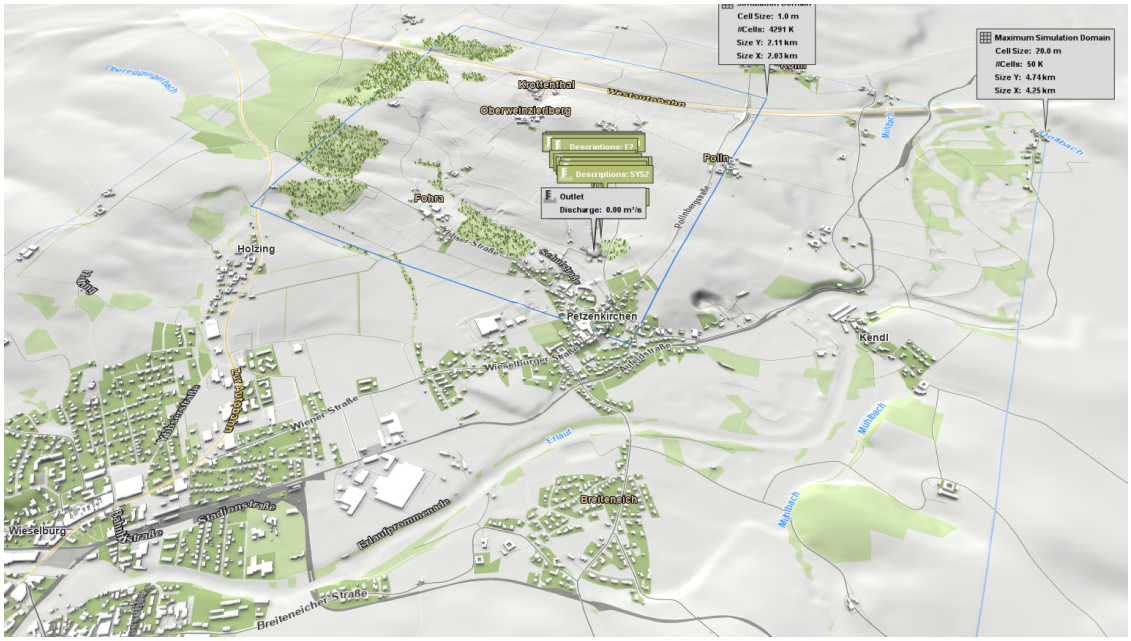
CHALMERS
UNIVERSITY OF TECHNOLOGY



Fast Rainfall Runoff Simulation and Parameter Calibration

Implementation of a bidirectionally coupled infiltration model and a method for parameter calibration for large scale flood simulations

Master's thesis in Complex Adaptive Systems

Daniel Hesslow

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:25

Fast Rainfall runoff simulation and parameter optimization

Implementation of a bidirectionally coupled infiltration model and a method for parameter calibration for large scale flood simulations

Daniel Hesslow

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2020

Fast Rainfall runoff simulation and parameter calibration
Implementation of a bidirectionally coupled infiltration model and a method for parameter calibration for large scale flood simulations
Daniel Hesslow

Supervisor: Andreas Buttinger, VRVis: Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH
Examiner: Håkan Nilsson, Mechanics and Maritime Sciences, Fluid Dynamics

Master's Thesis 2020:25
Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Screen shot from the visdom application for simulation of flooding events, here over this town of petzenkirchen.

Department of Mechanics and Maritime Sciences
Gothenburg, Sweden 2020

Abstract

Since the number of flooding events are expected to rise in the coming years as a consequence of global warming, accurate simulation of such events are now more important than ever. Running simulations and seeing the effects of different possible actions serves as a very important tool to mitigate the consequences of such events. For the results of the simulations to be accurate it is important that both the parameters that govern the surface flow and the subsurface flow are known, or that they can be accurately estimated. While it is feasible to measure some parameters to sufficient accuracy, such as the topology, this is not true for for all parameters. The subsurface flow is governed by the soil characteristics at all points in the simulation space and may vary over the depth. Additionally, measuring the soil characteristics at any one point is expensive. It is, therefore, not feasible to measure the soil characteristics at all points and all depths to a sufficient accuracy.
The traditional approach is to have an expert estimate all such parameters, however this is costly and if ground truth data from previous flooding events are available the parameters can instead be tuned to fit with the previous events.
In this thesis an efficient and numerically accurate way to calculate the infiltration of the multi-layer Green-Ampt model is presented as well as a method for automatically optimizing the parameters of large-scale fluid simulations. The developed methods are implemented in the VISDOM-application developed by VRVIS and evaluated on different scenarios.

Keywords: Infiltration, Runoff, Bayesian Optimization, Parameter Calibration

Acknowledgements

I would like to thank my supervisor, Andreas Buttinger, for always answering my questions quickly and his general guidance over the process. I would also like to thank the whole group of people working on VISDOM, lead by Jürgen Wasser, for letting me use their software for this thesis. In general I would also like to thank the whole of VRVis for both making be feel welcome in Vienna and providing a nice place to work.
I would also like to thank Håkan Nilsson for being my examiner. Further I would like to thank everybody who has collected the hydrological data used in this thesis as well as the open source community in general for all great software libraries out there without which this thesis would not be possible.
Finally, I would also like to thank ERASMUS for the financial help.

Daniel Hesslow, Vienna, July 2020

Contents

List of Figures

1

Introduction

1.1 Background and Problem Description

Since the number of flooding events are expected to rise in the coming years as a consequence of global warming, accurate simulation of such events are now more important than ever. Running simulations and seeing the effects of different possible actions serves as a very important tool to mitigate the consequences of such events.

The traditional approach when it comes to computing the infiltration is to assume that the infiltration happens over a flat surface with constant parameters such that the infiltration is constant over space. The infiltration rate for all times can then be calculated in a separate pass without taking any surface flow into account. During the simulation, the previously calculated infiltration rate can be used without any additional runtime computation. We will refer to this as static infiltration. Instead, in this thesis, methods to dynamically calculate the infiltration rate during the simulation will be investigated. We refer to this as bidirectionally coupled infiltration, since the infiltration is both affected by the water simulation and vice versa. The infiltration rate may differ for different cells in the simulation. Further, it is important that such a method can run efficiently so that it is possible to run it even on large scale simulations.

However, any method to calculate the infiltration is only accurate if the parameters used in the model is accurate. The traditional approach is to let experts estimate each parameter of the model, potentially based off of real world measurements. However, this process is expensive, and if we let the parameters freely vary in space, it becomes entirely unfeasible. To combat this problem, methods for calibrating such model parameters will be investigated. In particular, the goal of the calibration procedure is to automatically find model parameters that makes the simulation and the ground truth data from previous flooding events match as well as possible. This is, however, a difficult optimization problem to solve. Large scale simulations can take many hours, or even days, to run. As such it will not be possible to sample the parameter space many thousands of times. Further, in general we do not have access to the gradients of the error with respect to the parameters. The problem is thus to be able to adequately estimate the infiltration parameters in as few iterations as possible.

1.2 Previous Work

The development of simulation software that includes a bidirectionally coupled infiltration model has previously been done by Delestre, Darboux, James, *et al.*[1]. Fernández-Pato, Caviedes-Voullième, and García-Navarro [2] has also developed software to simulate the shallow water flow and also includes some simple parameter calibration. For simpler models for the surface flow, assuming for example open channel flow, more work for parameter calibration has been done. Wang, Sang, Jiao, *et al.* [3] use a combination of manual tuning and the Nelder-Mead optimization algorithm [4]. Many authors, for example Zhang, Wang, and Meng [5], has also used the SCE-UA algorithm which was specifically designed as a black box optimization algorithm for conceptual runoff modelling [6] [7]. Ding, Jia, and Wang [8] has evaluated multiple different methods for finding optimal values for Manning's roughness coefficients in shallow water simulations and conclude that L-BFGS-B works well for this task.

1.3 Aim and Limitations

The aim of this thesis to twofold: First, to develop and implement an infiltration model. The infiltration model must be sufficiently accurate so that the results of the simulation can be trusted for real world situations, but it must also be fast enough so that it is possible to execute it even on large scale simulations. The infiltration model will be integrated into the existing codebase at VRVis.
Secondly, an optimization procedure to find sufficiently good parameters for the infiltration model will be implemented. The optimization procedure will take data from real world flooding events and attempt to find parameters so as to make the simulation and the ground truth data match.

1.4 Structure of the Thesis

- **Chapter 2: Theory** contains the theoretical background that is necessary for the rest of the thesis, this section both contains background information form other sources as well as new derivations.
- **Chapter 3: Implementation** contains an overview of implementation process of the software that has been developed, both for the infiltration modelling and the optimization procedure.
- **Chapter 4: Results** contains the results of the thesis, both data from the infiltration procedure as well as from the parameter optimization.
- **Chapter 4: Closure** discusses the findings of the thesis and possible avenues for future work on this subject.

2

Theory

2.1 Infiltration Models

2.1.1 Darcy's Law

After conducting experiments on sandbeds in 1856, Darcy found an empirical equation for for flow in saturated porous soils [9]. It relates the flow between two points in space, P_1, P_2 to the difference between their hyrdaulic head as follows

$$f = K_s \left(\frac{\Phi(P_1) - \Phi(P_2)}{\|P_1 - P_2\|} \right)$$

Where f is the flow rate, K_s is a constant depending on the soil, known as the saturated hydraulic conductivity, and $\Phi(P)$ is the hydraulic head at the point P. The hydraulic head is a measurement of the potential energy from gravity and pressure at a point in terms of the height of the top of the fluid column needed to produce such a pressure. The hydraulic head in stationary water column is constant, further down the energy is in the form of pressure and further up in the form of gravitational potential energy.

2.1.2 Richards Equation

The Richards equation

$$\frac{\partial \theta}{\partial t} = \frac{\partial}{\partial z} \left[K(\theta) \left(\frac{\partial \Phi}{\partial z} + 1 \right) \right]$$

describes flow of water in potentially unsaturated soils. If the soil is saturated it simplifies to Darcy's Law. Here θ is the water content at a particular depth. Notice that the hydraulic conductivity, K, in the Richards equation is a function of the water content θ. To be able to solve the Richards equations one must relate Φ and θ which is done through the so called water retention curve. Solving the Richards equation, even in one dimension as it is described here, is costly and can only be achieved in simulations of limited size. Further, in real world cases, most soil parameters need to be estimated, since the exact soil type is not known everywhere. Estimating the water retention curve for all points in space increases the difficulty.

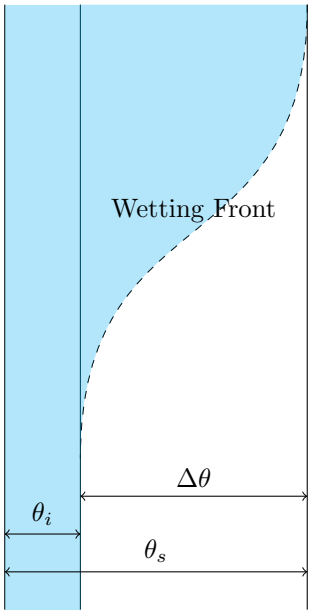


Figure 2.1: Illustration of the water content at a given moment of time. The wetting front will continuously move downwards, and is gradually increasing from a water content of θ_i to θ_s.

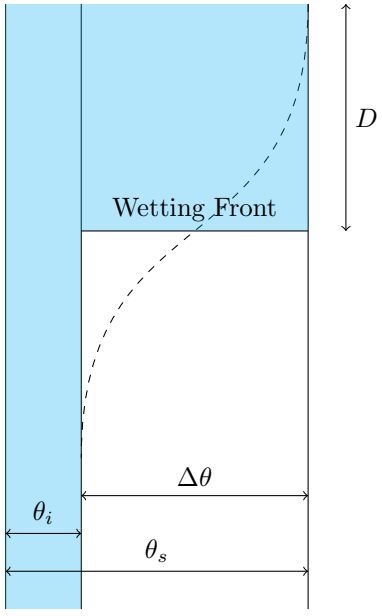


Figure 2.2: Illustration of the piston flow approximation used in the Green-Ampt model. The wetting front is approximated as instantly increasing from a water content of θ_i to θ_s.

2.2 Green-Ampt

During infiltration, the soil is in general not fully saturated, see figure 2.1 for an illustration. However to be able to solve the infiltration rate without solving the expensive Richards equations, one can make the approximation that the flow is piston-like, see fig. 2.2, that is to say that the soil at a given depth becomes instantly saturated once the wetting front reaches it.
The flow from a point, P_s, at the surface to a point, P_D, at the wetting front directly below it can then be calculated from Darcy's law. The hydraulic head at P_s, $\Phi(P_s)$, is the height of the water column above it, which will be denoted h_0. The hydraulic head at P_D is $\Phi(P_D) = -D - \Psi$ where Ψ is a soil parameter called the suction head describing the negative pressure caused by the capillary action of the soil and D is the depth of the wetting front. The flow between these points are then, according to Darcy's law,

$$f = K_s\left(\frac{\Phi(P_s) - \Phi(P_D)}{\|P_s - P_D\|}\right) = K_s\frac{(h_0 + \Psi + D)}{D}$$

Let $F = \int f dt$ then $D = \frac{F}{\Delta\theta}$, this leads to the usual form of the so called Green-Ampt model [10]

$$f = K_s\left(\frac{\Delta\theta(h_0 + \Psi) + F}{F}\right)$$

For simplicity, in future references, we will drop the Δ in front of the θ since the other quantities related to the water content are of no interest. Further in keeping with other literature using the Green-Ampt model we will refer to θ as the porosity. The Green-Ampt model can be solved by integrating both sides of the equation as follows

$$\begin{aligned}
K &= \frac{dF}{dt}\left(\frac{F}{\theta(h_0 + \Psi) + F}\right) \\
\int_{t=0}^{t=T} K dt &= \int_{t=0}^{t=T}\left(\frac{F}{\theta(h_0 + \Psi) + F}\right)\frac{dF}{dt}dt \\
KT &= \int_{F=F(0)}^{F=F(T)}\left(\frac{F}{\theta(h_0 + \Psi) + F}\right)dF \\
KT &= F(T) - F(0) + \theta(h_0 + \Psi)log\left(\frac{\theta(h_0 + \Psi) + F(0)}{\theta(h_0 + \Psi) + F(T)}\right)
\end{aligned}$$

Commonly a simpler version of the solution with $F(0) = 0$ is presented, however, the slightly more general version is necessary to use when the model is coupled to the water simulation as will be seen in later chapters.

The Green-Ampt model can also be solved in closed form using the Lambert-W function which is defined as

$$f(w) = we^w = z$$
$$W(z) = f^{-1}(z) = w$$

Since the Lambert-W function has multiple branches, care must be taken to select the appropriate branch. For the Green-Ampt equation the first negative branch produces positive infiltration rates and should therefore be chosen. The solution of the Green-Ampt model in terms of the Lambert-W function is

$$F(t) = -\Psi\Theta \left(W_{-1} \left(-\frac{(F_0 + \Psi\Theta)\, e^{-\frac{F_0 + \Psi\Theta + kt}{\Psi\Theta}}}{\Psi\Theta} \right) + 1 \right)$$

In practice, however, iterative methods must be used to accurately solve the Lambert-W function. Nevertheless, if one has access to efficient solvers for the Lambert-W function this formulation may simplify the implementation. Since the Lambert-W function is well-studied in literature, many approximations exist, if only approximate soluaitons to the Green-Ampt models are neccesary, this formulation may be of further interest. Approximate solutions to the Green-Ampt equations are studied in detail by Barry, Parlange, Li, *et al.* [11].

2.2.1 MultiLayered Green-Ampt Model

A multilayered model can be constructed by having the same parameters change discretely over the depth, yielding layers with different parameters. This is important since usually, in nature, the soil does not remain constant over all depths. Such a model can be constructed with an effective Green-Ampt model [12] by using the parameters of the suction head and porosity from the layer where the wetting front is currently, as well as a an effective hydraulic conductivity constructed by taking the harmonic mean of the hydraulic conductivities within the infiltration column [13]. Formally the effective hydraulic conductivity can be defined as

$$K_e = \frac{D}{\int_{d=0}^{D} \frac{1}{K_d}},$$

where D is the depth of the infiltration column and K_d is the saturated hydraulic conductivity at depth d. It is important to note, that the depth of the infiltrated water column is not equivalent to the cumulative infiltration, F, since the depth takes the porosity of the medium into account. For a model with multiple discrete layers with hydraulic conductivity K_i and thickness Z_i the hydraulic conductivity can be calculated as

$$K_e = \frac{D}{\sum_{i=1}^{n} \frac{Z_i}{K_i}}$$

Here Z_n is the effective depth of the final layer, which is to say $Z_n = D - \sum_{i=1}^{n-1} Z_i$

it is important to note that the $n-1$ first layers can be combined into a single layer while still capturing the full model since

$$\frac{D}{\sum_{i=1}^{n}\frac{Z_i}{K_i}} = \frac{D}{\frac{\sum_{i=1}^{n-1} Z_i}{\sum_{i=1}^{n-1}\frac{Z_i}{K_i}} + \frac{Z_n}{K_n}}$$

The new amalgamated layer will be described by the properties:

$$\begin{aligned} Z &= \sum_{i=1}^{n-1} Z_i \\ K &= \sum_{i=1}^{n-1} \frac{Z_i}{K_i} \\ \theta &= \frac{\sum_{i=1}^{n-1} \theta_i Z_i}{Z} \end{aligned}$$

With no loss generality we will therefore assume there to be two layers. If more layers are desired on can combine the two previous layers into one once the wetting front reaches the third layer and thereafter continue the simulation.

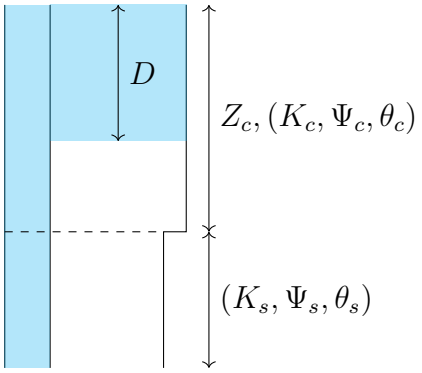


Figure 2.3: Illustration of the two layer Green-Ampt Model.

2.2.2 Two layered Green-Ampt model

In this section we will denote the parameters that are associated with the first layer, the crust, with a subscript c, ie. θ_c, Ψ_c, K_c and the parameter that are associated with the second layer, the soil with a subscript s, ie. θ_s, Ψ_s, K_s see figure 2.3 for an illustration.

To solve the two layered Green-Ampt model, the easiest approach is to first reformulate it in terms of the infiltrated depth instead of the cumulative infiltration.

$$
\begin{aligned}
K_e &= \frac{D}{\frac{Z_c}{K_c} + \frac{D-Z_c}{K_s}} \\
\frac{dD}{dt} &= \frac{1}{\theta_c} K_e \frac{h_0 + \Psi_c + D}{D} \\
\frac{dD}{dt} &= \frac{1}{\theta_c} \frac{h_0 + \Psi_c + D}{\frac{Z_c}{K_c} + \frac{D-Z_c}{K_s}}
\end{aligned}
$$

As with the one layer model, it can be solved by integrating both sides of the equation.

$$
\begin{aligned}
\frac{1}{\theta_c} &= \frac{dD}{dt} \frac{\frac{Z_c}{K_c} + \frac{D-Z_c}{K_s}}{h_0 + \Psi_c + D} \\
\int_{t=0}^{T} \frac{1}{\theta_c} dt &= \int_{t=0}^{T} \frac{\frac{Z_c}{K_c} + \frac{D-Z_c}{K_s}}{h_0 + \Psi_c + D} \frac{dD}{dt} dt \\
\frac{T}{\theta_c} &= \int_{D=D_0}^{D=D_T} \frac{\frac{Z_c}{K_c} + \frac{D-Z_c}{K_s}}{h_0 + \Psi_c + D} dD \\
\frac{T}{\theta_c} &= \frac{K_c \left(D_T - D_0\right) + \left(Z_c K_c - Z_c K_s + (\Psi_c + h_0) K_c\right) \log\left(\frac{D_0 + (\Psi_c + h_0)}{D_t + (\Psi_c + h_0)}\right)}{K_c K_s} \\
\frac{K_c K_s T}{\theta_c} &= K_c \left(D_T - D_0\right) + \left(Z_c K_c - Z_c K_s + (\Psi_c + h_0) K_c\right) \log\left(\frac{D_0 + (\Psi_c + h_0)}{D_t + (\Psi_c + h_0)}\right)
\end{aligned}
$$

Solutions in terms of the Lambert-W function also exist, and is easiest to obtain using computer algebra systems such as sympy [14] to directly solve the differential equations. The resulting solutions are however very large, and suffers from numerical issues, as such they will not be presented here.

2.3 Chi Squared- and Noncentral Chi Square distributions

If a collection of k random variables X_i is standard normally distributed, $X_i \sim \mathcal{N}(\mu = 0, \sigma = 1)$, then the sum of the squares of those variables will be chi square distributed with k degrees of freedom, $\|X\|_2^2 = \sum_{i=1}^{k} X_i^2 \sim \chi^2(k)$
If those variables instead have different means, $X_i \sim \mathcal{N}(\mu = \mu_i, \sigma = 1)$, then their squared sum will be distributed according to the noncentral chi square distribution, $\|X\|_2^2 \sim \chi'(k, \sum_{i=1}^{k} \mu_i^2)$

2.4 Numerics

When constructing numerical methods it is important to make sure that errors, that are inherently introduced by calculating quantities in finite precision, does not

accumulate over time such that the final result is far from the correct result. To be able to achieve this it is important to understand how the infinite precision reals are represented on computers. Let $\oplus$ and $\ominus$ denote the addition and subtraction operators on finite precision floating point numbers.

2.4.1 IEEE Floating Point Numbers

While many different encodings of the reals are available, the defacto standard used in computers today is the IEEE-754 standard. The standard describes 7 different formats, three of which are commonly available in hardware. The common formats are binary16, binary32, and binary64 where the number indicates the number of bits used to encode the number. Binary16, Binary32 and Binary64 are commonly referred to as half, float and double respectively.
An IEEE floating point number consists of three parts, the sign, s, the exponent q, and the mantissa or significand, c. The real represented by the floating point number can then be calculated as $sc2^q$. Note that to represent all numbers it is sufficient for c to be in the range $[1, 2)$. In IEEE this is indeed the case, ignoring the special case of so called denormal numbers which is not of particular interest to this discussion. q and s can then be seen as describing an interval from $s2^q$ to $s2^{q+1}$ inside of which c is used to uniformly select a number. let $\text{ulp}(x)$ denote the distance between two subsequent numbers in the range $R = [s2^q, s2^{q+1})$ such that $x \in R$
The result of an operation on a floating point number such as addition, multiplication, division etc. is guaranteed to be exact if its result is exactly representable. If, however, this is not the case the result is rounded, according to a specified rounding rule, to a representable number. The error of the addition of two number a and b is at most $\frac{\text{ulp}(a+b)}{2} \leq \max(\text{ulp}(a), \text{ulp}(b))$. The same inequality also holds for subtraction.

2.4.2 Compensated summation

Assume one is to add up many numbers $\sum_{i=1}^{n} X_i$ using floating point addition. The naive approach seen in listing 2.1 will have have large errors if the sum is large compared to the individual numbers. However, it is possible to do better, consider A large and x small, and we wish to compute $A' = A + x$ but because it is finite precision we know that there will likely be some error in the computation which we also want to keep track of. Then $A + x = A \oplus x + (x - ((A \oplus x) - A))$. Since $|(A \oplus x) - A)| \leq \max x, \text{ulp}((A \oplus x) - A)) \leq \text{ulp}(x)$ the error of the error correction term $(x \ominus ((A \oplus x) \ominus A))$ is smaller than $2\text{ulp}(x)$, which is much smaller than $\text{ulp}(A)$ if $A \gg x$
From this one can derive the algorithm for compensated summation, or kahan summation after its inventor [15], seen in listing 2.2.

Listing 2.1: The naive method for summing a list of variables X

```
def naive_sum(X):
    acc = 0
    for x_i in X:
```

```
        acc += x_i
    return acc
```

Listing 2.2: The kahan summation method for summing a list of variables X

```
def kahan_sum(X):
    acc = 0
    c = 0
    for x_i in X:
        x'_i = x_i+c
        acc' = acc + x'_i
        c = x'_i-(acc' - acc)
        acc = acc'
    return acc
```

2.5 Gaussian Processes

While, it is not strictly necessary to understand how Gaussian processes work to understand or use the methods described in this thesis they will nevertheless be used so some of the theoretical background will be discussed here. To understand the rest of this thesis one must only be aware of the fact the gaussian processes (GP) provide a framework to, for a function f, construct a probability distribution over possible values in each each point, $f(x)$, given a number of previous observations $\{x_i, y_i = f(x_i)\}$. In particular this probability distribution will be a normal distribution.
Using any popular framework for gaussian processes one can query the mean and standard deviation of a point given the previous samples. Formally a Gaussian Process is a, potentially infinite, collection of random variables. The joint distribution of any finite subcollection of these random variables is gaussian. While we won't show it here the second requirement is fulfilled if and only if the covariance function is positive definite. The same holds true for kernels and as such kernels and covariance functions is generally used interchangeably in reference to gaussian precesses. There are of course a wide variety of kernels possible to chose from. However in practice only a few are commonly used, the most common being the radial basis function (RBF) and the Matérn kernel.
Let $d = \frac{\|x,x'\|_2}{l}$ be the scaled euclidean distance between the two points x and x'. In the context of Gaussian Process l can be though of as a characteristic length scale of the problem describing how quickly the problem varies. The RBF and Matern-kernel is then defined as

$$RBF(d) = e^{-\frac{d^2}{2}}$$
$$Matérn_\nu(d) = \frac{2^{1-\nu}}{\Gamma(\nu)}(\sqrt{2\nu}d)^\nu K_\nu(\sqrt{2\nu}d),$$

Here ν is a non negative parameter, Γ is the gamma function and K_ν the modified Bessel Function of the second kind. For the Matérn kernel a common choice for ν is either $\frac{3}{2}$ or $\frac{5}{2}$ where simpler expressions are available,

$$Matérn_{\frac{3}{2}}(d) = (1 + \sqrt{3}d)e^{-\sqrt{3}d}$$
$$Matérn_{\frac{5}{2}}(d) = (1 + \sqrt{5}d + \frac{5d^2}{3})e^{-\sqrt{5}d}$$

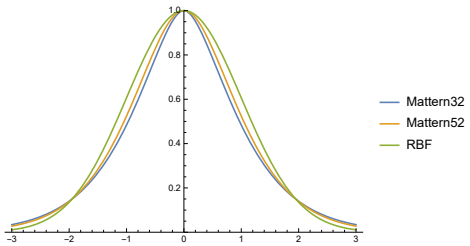
It has been argued that one possible problem with the radial basis function is that it is too smooth [16]. The problem being that if a function is exactly known on an arbitrarily small domain, the rest of the function is, according to the RBF covariance function, also known. This in turn, is argued to be unrealistic for physical processes. The Matérn covariance function can intuitively be thought of as a modification to the RBF such that it is only ν-times differentiable, instead of infinitely differentiable. In fact if we let ν go to infinity the Matérn-kernel approaches the RBF kernel. For the comparison of the different covariance functions on a simple problem see fig. 2.4 The posterior distribution of the Gaussian process is then given by

$$\begin{aligned} \mu &= K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} y \\ \sigma^2 &= K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} K(X, X_*) \end{aligned} \tag{2.1}$$

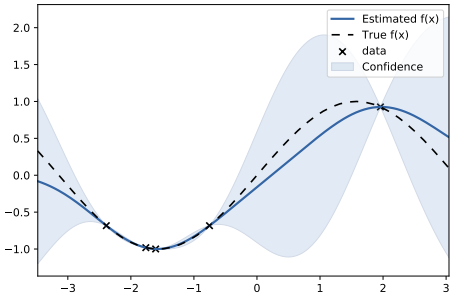
Here $\{X, y\}$ are the observations and X_* is the points where the predictive distribution is to be evaluated, σ_n^2 is a parameter describing the the expected variance of the measurements y = f(X). For information about the derivation of these equations as well as in depth explanations of gaussian processes in general we refer to Rasmussen and Williams [17].

2.6 Optimization Methods, Background

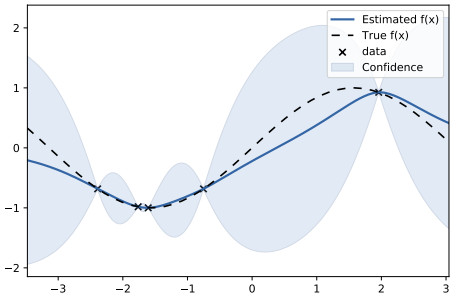
The field of optimization is well-developed and there exists many optimization-algorithms useful in different scenarios. Broadly, they can be categorized by the number of derivatives that is needed in each sample point. For many common use-cases in machine learning first order methods such as gradient decent, ADAM [18] or BFGS [19], offers a good compromise between the time taken to calculate the necessary information in each sample point and the convergence rate of the algorithm. However, there exists a wide variety of problems where the gradient is either not possible to calculate or is too expensive calculate. For these problems zeroth order methods must be used where only the value of the function to be optimized is available, as such, these methods are also commonly referred to as black-box optimization methods.
The problem of automatic calibration of infiltration parameters addressed in this thesis is just such a problem. We are to find infiltration parameters such that the simulation and recorded data matches as closely as possible. While it is indeed possible to calculate gradients in fluid simulations, as has been demonstrated by for example Schenck and Fox [20], it is not available in the software that we are working with, VISDOM, and implementing it would be outside the scope of this thesis. Further, computing gradients during simulation would increase both the memory consumption and computational difficulty of the problem. As such this thesis will be limited to gradient free methods for the parameter optimization.



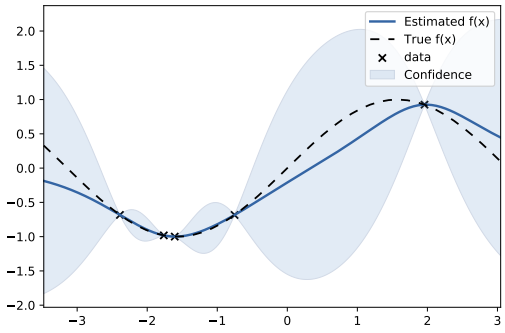
(a) Comparison of the shape of the different kernels



(b) The predictive distribution given by the RBF kernel.



(c) The predictive distribution given by the Matern32 kernel.



(d) The predictive distribution given by the Matern52 kernel.

Figure 2.4: The result of fitting a gaussian process on five noise-free points from y=sin(x), with different covariance functions.

Black-box optimization algorithms can be split into two main types. Those for functions where evaluating new samples is cheap and those where it is expensive. In the first group, there are both naive methods such as grid search or random search but also more sophisticated methods such as differential evolution, particle swarm optimization, genetic algorithms and the fairly recent algorithm SHGO.
However, if evaluating a new sample point is very expensive we can allow the optimization algorithm to be much slower. Recently, this class of algorithms has gained a lot of interest in the context of hyper-parameter optimization, in particular in the context of deep learning. The state of the art in such cases are so called Sequential Model Based Optimization or SMBO. Since this is also the context of our problem the methods used here will be thoroughly examined in the following sections.

2.6.1 Sequential Model-Based Optimization - SMBO

Sequential model based optimization works in two steps. First a model is constructed based on the samples that have already been evaluated. Then that model is optimized based on some criteria called the acquisition function. The optima of the acquisition function is then chosen as the next sample point to be evaluated, and the process is then repeated see listing 2.3.
It is important to note that in the framework of SMBO there is still a lot of freedom

left in choosing both the model and the acquisition function. In particular the model usually contains a probability distribution over possible values for each point in space. A good acquisition function then makes the trade off between sampling regions that are known to be good and regions which are uncertain. In other words it controls the trade off between exploration and exploitation.
There are two methods that are by far more common than any others in this context, Tree Parzen Estimators (TPE) and Gaussian Process Bayesian Optimization (GPBO).

Listing 2.3: Pseudo Code for Sequential Model Based Optimisation, where f(x) is the expensive function that is to be minimized and $g(x, M)$ is the acquisition function

```
def SMBO(f, g):
    history = []
    loop:
      M = fitModel(history)
      x* = argmin_x g(x, M)
      history.append({x*, f(x*)})
    return min(history)
```

2.6.2 Acquisition function

There are a few common acquisition functions, Probability of Improvement (PI) is defined as

$$PI_{y^*}(x) = P(f(x) < y^*)$$

and is probably the simplest acquisition functions that still see some use. However since it doesn't take into account how much better a new value is than the best previously known one, it tends to perform quite badly and favour exploitation much over exploration. The slightly more complicated acquisition function Expected Improvement (EI) addresses this problem and is defined as

$$EI_{y^*}(x) = \int_{y=\infty}^{y^*} (y^* - y) P(f(x) = y)$$

or equivalently

$$EI_{y^*}(x) = y^* - E(\min(f(x), y^*)$$

If we are only to run the algorithm for one more iteration it is clear from the second definition that EI is optimal. However, for any real problem we will need to sample many times and then this greedy solution will still intuitively favour exploitation over exploration. For thorough discussion on this topic as well as a better acquisition we refer to Qin, Klabjan, and Russo [21].
Finally we will also consider Upper Confidence Bound (UCB), which is very popular in theoretical works but also works rather well in practice. UCB is often described as "optimism in the face of uncertainty", that is to say if we are uncertain about the true

value of a random variable we assume it is better than we think. This is achieved by picking the nth percentile of the CDF produced by the model. Intuitively it is clear that this should lead to more exploration than PI. The tradeoff between exploration and exploitation is then a tunable parameter of the function which means that as long as we chose good parameters for UCB it should be able to strike a balance between exploration and exploitation.

2.6.3 Gaussian Process Bayesian Optimization - GPBO

If an implementation of gaussian processes is available, Gaussian Process Bayesian Optimization is fairly straight forward. The only difficulty lies in efficiently optimizing the acquisition function, once the dimensionality of the input space grows large methods such as grid search quickly becomes unfeasible.

In literature DIRECT [22] is commonly used, see for example [23], [24], however, this method suffers from the same problems. Recently, more specialized versions of evolutionary algorithms have also been employed [23]. For our purposes we will simply use random search for a predetermined number of iterations. While this will not find the optima of the acquisition function, we will at least find a good candidate. If one is concerned with theoretical guarantees of convergence, it should be noted that this is a bad choice since it means that we have the exact same order of convergence as random search. However, the number of samples we can afford to evaluate the true function on is so low that analysis as number of samples grows large is of no practical importance. This choice is done for practical reasons only, better alternatives exists and should be used for problems where the time spent optimizing the acquisition function is significant.

2.6.4 Tree Parzen Estimator - TPE

Tree Parzen Estimators (TPE) introduced by Bergstra, Bardenet, Bengio, *et al.* [25] has recently gained a lot traction as it has been empirically shown to preform as well or better than GPBO on many problems while being less computationally expensive, especially as the dimensionality of the input grows large. The idea is to separate the observations into two subsets: those above desired value y^* and those below. For each subset a density is formed through kernel density estimation. We will refer to the density lower than y^* as $l(x)$ and that greater than y^* as $g(x)$. The probability of x given y can then be estimated as

$$p(x|y) = \begin{cases} l(x), & \text{if } f(x) < y^* \\ h(x), & \text{otherwise} \end{cases}$$

Let y^* be defined through $p(y < y^*) = \gamma$. The expected improvement over y^* can then easily be optimized by optimizing the ratio: $\frac{l(x)}{h(x)}$. The proof follows below.

$$
\begin{aligned}
EI_{y^*}(x) &= \int_{y=-\infty}^{y^*} (y^* - y) p(y|x) dy \\
&= \int_{y=-\infty}^{y^*} (y^* - y) \frac{p(y)p(x|y)}{p(x)} dy \\
&= \int_{y=-\infty}^{y^*} (y^* - y) \frac{p(y)l(x)}{p(x)} dy \\
&= \frac{l(x)}{p(x)} \int_{y=-\infty}^{y^*} (y^* - y) p(y) dy \\
&\propto \frac{\gamma l(x)}{p(x)} \\
&= \frac{\gamma l(x)}{\gamma l(x) + (1-\gamma) h(x)} \\
&= \frac{1}{1 + (\gamma^{-1} - 1)\frac{h(x)}{l(x)}} \\
\text{argmax}_x EI_{y^*}(x) &= \text{argmax}_x \frac{l(x)}{h(x)}
\end{aligned}
$$

Good candidate samples can efficiently be found by drawing samples form l(x) and evaluating their expected improvement.

2.6.5 Target Vector

While we have previously considered our problem as a black-box optimization problem with a single output this formulation discards important information: we know the errors for each time step of the simulation, not only its $\ell 2$-norm. If one were to manually tune such parameters it is easy to see that one would not discard this information, one parameter may for example control the value at small times while another does so for larger times. If it is possible to infer such structures it is clear that it should improve the sample efficiency of the optimization algorithm.

2.6.5.1 GPBO

GPBO for target vectors has previously been investigated by Uhrenholt and Jensen [26]. They come up with a simple and elegant approach, based on modelling each timestep independently as a separate gaussian process. They then note that the $\ell 2$-norm of the of the resulting vector of gaussian variates is approximately distributed according to a scaled version of the noncentral chi square distribution. Formally

$$
\begin{aligned}
f(x) = y, x \in \mathcal{R}^n \\
\hat{y}_i \sim \mathcal{N}(\mu_i, \sigma_i) \\
\frac{\|\hat{y}\|_2^2}{\sum_{i=1}^k \sigma_i^2} \overset{\text{approx}}{\sim} \chi'^2(k, \sum \mu_i^2)
\end{aligned}
$$

They show that this approximation is bias free, given the previous assumption that the y_i are independent. The expected improvement is then given by the truncated mean of the noncentral chi square distribution. Which can be computed in closed form as

$$w \sim \chi'^2(K, \lambda)$$
$$\mathbb{E}(w|w < a) = KF_{K+2,\lambda}(a) + \lambda F_{K+4,\lambda}(a)$$

Where $F_{K,\lambda}$ is the CDF of the non central chi square distribution. We note two things about this approximation, first if σ_i is given by a a gaussian process then all σ_i are equal. This is easy to see from how σ is computed in a gaussian process, see eq. 2.1. This means that this approximation is exact given that y_i are independent and can be calculated as

$$\frac{\|\hat{y}\|_2^2}{k\sigma^2} \sim \chi'^2(k, \sum \mu_i^2)$$

Further, the assumption that y_i is independent is quite strong and does not hold in general. To show that this is problematic consider the following, we are trying to find the parameter c as to minimize $f(c) = \int c dt$. If we optimize it using the proposed method we must first chose the number of components, k'. This will then cause us to model it as a non central chi square distribution with k' degrees of freedom, while in fact for this simple function, all samples are the same and the number of degrees of freedom is one. The number of degrees of freedom can be bounded by the number of arguments to f. Another possible model is therefore

$$\frac{\|\hat{y}\|_2^2}{k\sigma^2} \sim \chi'^2(\min(k, n), \sum \mu_i^2)$$

2.6.5.2 TPE

While generalizations to multiple outputs has been explored for GPBO, no similar work has been done for TPE, therefore we attempt to derive one such generalization. We will consider the simpler case of a target vector of length two first. First note that there are two potentially different approximations for p(x).

$$p(x) \approx \gamma_1 l_1(x) + (1 - \gamma_1) g_1(x)$$
$$p(x) \approx \gamma_2 l_2(x) + (1 - \gamma_2) g_2(x)$$

$$
\begin{aligned}
EI_{y^*}(x) &= \int_{y_1=-\infty}^{y^*} \int_{y_2=y_1}^{y^*} (y^* - (y_1 + y_2)) p(y_1|x) p(y_2|x) dy_2 dy_1 \\
&\approx \int_{y_1=-\infty}^{y^*} \int_{y_2=y_1}^{y^*} (y^* - (y_1 + y_2)) \frac{p(y_1)p(y_2)p(x|y_1)p(x|y_2)}{p(x)^2} dy_2 dy_1 \\
&= \frac{l_1(x)l_2(x)}{p(x)^2} \int_{y_1=-\infty}^{y^*} \int_{y_2=y_1}^{y^*} (y^* - (y_1 + y_2)) p(y_1)p(y_2) dy_2 dy_1 \\
&\propto \frac{l_1(x)l_2(x)}{p(x)^2} \\
&\approx \frac{l_1(x)l_2(x)}{(\gamma_2 l_2(x) + (1-\gamma_2)g_2(x)) * (\gamma_1 l_1(x) + (1-\gamma_1)g_1(x))} \\
&= \Big((\gamma_2 + (1-\gamma_2)\frac{g_2(x)}{l_2(x)})(\gamma_1 + (1-\gamma_1)\frac{g_1(x)}{l_1(x)})\Big)^{-1}
\end{aligned}
$$

Where the first approximation comes from $p(y_1, y_2) \approx p(y_1)p(y_2)$, which is to say, the timesteps are assumed to be independent. In general where the number of outputs are more than two the final formula becomes

$$
EI(x) = \prod_{k=1}^{n} ((\gamma_i + (1-\gamma_i)\frac{g_i(x)}{l_i(x)})^{-1},
$$

where γ_i must be set such that y^* is equal for all outputs.

2.6.6 SHGO, Simplicial Homology Global Optimisation

SHGO is a two step global optimization algorithm. The first step consists of finding sub-domains which are locally approximately convex. Once such domains have been identified, a standard optimization algorithm for convex problems can be used such as for example SLSQP. The algorithm uses concepts from Simplicial Homology Global Optimisation Theory to find these sub-domains, however since this is much beyond the scope of this thesis further information is deferred to the original paper [27].

3

Implementation

3.1 Visdom

Visdom is a software application developed by VRVis. Its purpose is to provide a way to run simulations to help decision making in various scenarios, primarily related to mitigating the effect of flooding events, but other scenarios such as crowd simulations are also available.
The application consists of two different views: one where it is possible to design a specific scenario see fig. 3.1, and one where it is possible to evaluate the scenario and see the results of taking different actions see fig 3.2.

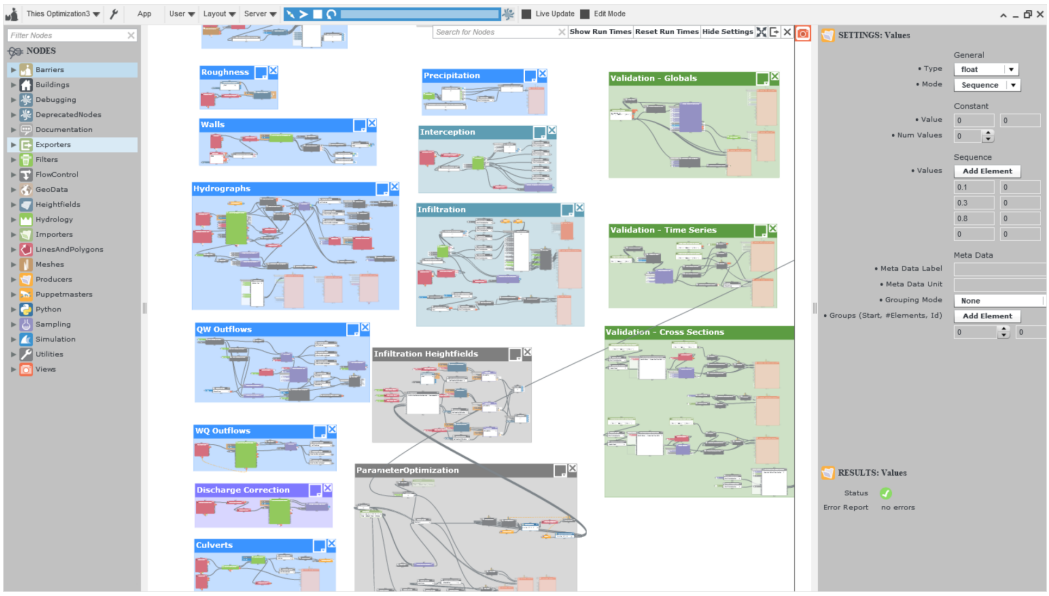


Figure 3.1: The deign view of the visdom application.

In the design view, one can use visual programming by connecting together nodes to set up the desired scenario. There exist hundreds of different nodes of varying complexity, some running the full shallow water simulation, others plotting results or importing data. By connecting these nodes both simple simulations of for example Green-Ampt infiltration with fixed water height can be created, but also much more complex scenarios that for example automatically create timelines with different parameters and evaluating the effects of for example introducing sand bag barriers in various locations on the damage caused on buildings can be created. In this thesis we will introduce two new things into the Visdom application. First, the

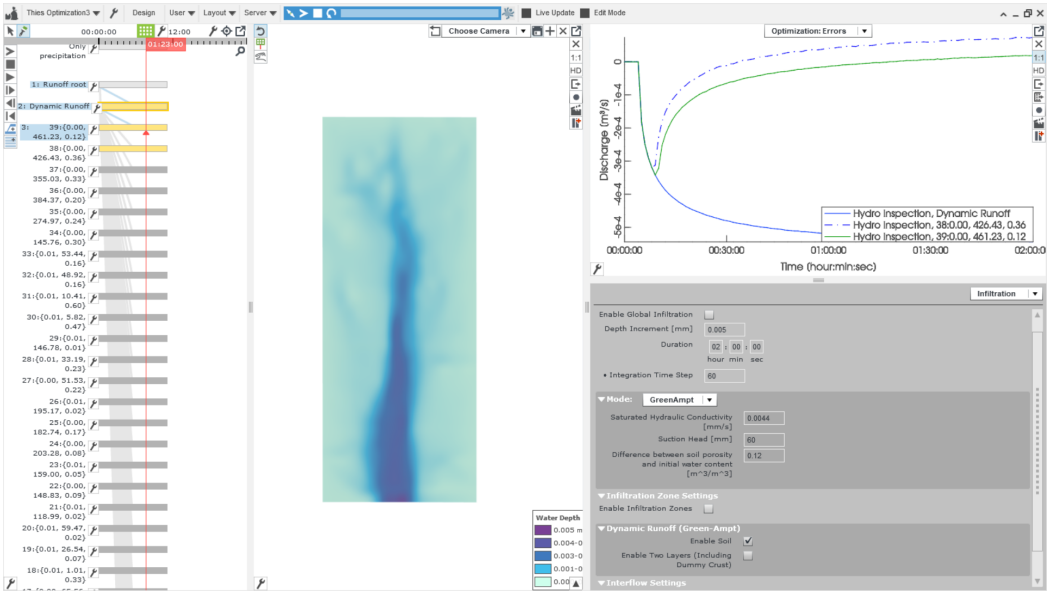


Figure 3.2: The application view of the visdom application

node that simulates the shallow water equations will be modified so as to calculate the infiltration dynamically using the specified Green-Ampt parameter. Second, a new node will be introduced that can handle the optimization of parameters of other nodes. In particular, this new node will be used to optimize the infiltration parameters of the shallow water simulation.
The Visdom-application is mainly written in C++, with the performance critical shallow water simulation also having an optional GPU-accelerated path written in CUDA. To be able to utilize libraries written in python for the optimization an additional node will be created that integrates python into the C++ application.

3.2 Implementation of Bidirectionally-coupled Infiltration

3.2.1 Exact Numerical Solutions to the Green-Ampt Model

In section 2.2 two different solutions to the Green-Ampt model were shown. We will use the implicit solution, instead of the one using the Lamber-W function, since solving the Lambert-W function is more difficult. Additionally we will compare it with a simple Euler-Forward approach using the infiltration rate directly, as was done in [28].
We notice that both the single and two-layer Green-Ampt equation can be formulated as solving an equation on the form.

$$c_1 = x - x_0 - c_2 \log\left(\frac{x + c_3}{x_0 + c_3}\right),$$

where c_1,c_2, c_3, x_0 are all constants. While it is possible to reduce this to 3 constants, on the form

$$c_1' = x + c_2' \log(c_3' x),$$

more accurate results can be achieved by instead directly solving for $\Delta x = x - x_0$. Since x and x_0 are potentially large, calculating their difference will lead to so called catastrophic cancellation.

$$\begin{aligned} c_1 &= \Delta x + c_2 \log\left(\frac{x_0 + \Delta x + c_3}{x_0 + c_3}\right) \\ c_1 &= \Delta x + c_2 \log\left(1 + \frac{\Delta x}{x_0 + c_3}\right) \\ c_3' &= \frac{1}{x_0 + c_3} \\ 0 &= -c_1 + \Delta x + c_2 \log(1 + c_3' \Delta x) \end{aligned}$$

The derivative of which is easily calculated as

$$\frac{d}{d\Delta x}\left(-c_1 + \Delta x + c_2 \log(1 + c_3' \Delta x)\right) = \frac{c_2 c_3'}{c_3' \Delta x + 1} + 1$$

Both the single and multi-layered model can thus easily be solved through Newton-Raphson iteration. If Δx is sufficiently small, or c_3' is sufficiently large the log term will approach one, therefore, a good approximation can be found by taylor expanding around one yielding the following expression

$$\log(c_3' \Delta_x + 1) \approx c_3' \Delta x - \frac{c_3'^2 \Delta x^2}{2}$$

Which gives a closed form approximate expression for Δx,

$$\Delta x \approx \frac{c_2 c_3 + \sqrt{(c_2 c_3 + 1)^2 - 2 c_1 c_2 c_3'^2} + 1}{c_2 c_3'^2}$$

This approximate solution for Δx can be used as a starting point for the Newton-Raphson iteration. Higher order root finding methods can also be used without much overhead since higher order derivatives are easily calculated, in practice the approximate solution given above yields very good starting points so the simple Newton-Raphson iteration converges in few iterations.

3.2.2 Euler-Forward approximation of the Green-Ampt Model

Another approach instead of solving the Green-Ampt model exactly, is to simply calculate the current infiltration rate and multiply that with the delta time. Since the infiltration rate is available in a simple closed form expression,

$$f = K_e \frac{D + \Psi + h_0}{D},$$

this is very efficient. However one issue that needs to be taken care of is the situation where the cumulative infiltration is equal to zero, which will happen at the start of the simulation. In this case the infiltration rate diverges to infinity. Previously Delestre, Darboux, James, *et al.* [28] has taken the same, euler forward approach, and they solved the divergence issue by simply clamping the infiltration rate to an arbitrary maximum allowed infiltration rate. We will take the same approach here.

3.3 Coupling the infiltration model and the surface flow

For each simulation cell we add two more variables, one representing the cumulative infiltration, and another representing the current error in the infiltration, similar to how compensated summation from listing 2.2. Additionally for each cell we store either three or seven more constants for the infiltration parameters, depending on if the one- or two-layer model is used. The infiltration can then be calculated according to the previous section. However, some care must be taken if one attempts to solve the two layer Green-Ampt model exactly since it is necessary to account for layer changes during a time step, see listing 3.1.
As previously mentioned, when updating the water heights we make sure to both also have an error term, which we call the pending runoff, as well as make sure that the water depths never go below zero. The full algorithm is seen in listing 3.2, which also updates the globalWaterVolume which checks that the shallow water solver being volume conserving.

Listing 3.1: Accounting for change of layers during the time step.

```
def solveGreenAmpt(F0, params, dt):
    Depth = F0/params.crust.porosity
    if Depth > params.crustThickness:
        return solveTwoLayerGaSoil(params, F0, dt);
    else:
        crustCapacity = params.crustThickness * params.crust.porosity
        tBoundary = solveCrustT(params.crust, crustCapacity, F0);
        if (dt > tBoundary ):
            F0New = params.crustThickness * params.crust.porosity
            return solveSoil(params, F0New, dt - tBoundary);
        else:
            return solveCrust(params.crust, F0, dt);
```

Listing 3.2: Accounting for infiltration in the shallow water solver.

```
def add_infiltration():
    infiltration = solveGreenAmpt(cumulativeInfiltrain, parameters, dt)
    pendingRunoff += getPrecepitation(t) - infiltration
    negativeHeight =  delta < 0:
    if negativeHeight:
        newH = 0;
```

```
waterDepthChange = newH - h;
pendingRunoff -= waterDepthChange;
if negativeHeight:
    pendingRunoff = 0;
globalWaterVolume += waterDepthChange
h = newH
```

3.4 Implementation of Optimization

3.4.1 Optimization Node

A new node was created that, given a list of input and its corresponding outputs, creates a new sample that should be evaluated. Since there are a vast number of optimization algorithms available in python, instead of picking one and implementing it in c++ directly, it was decided that a better approach would be to run python directly from the c++ application. This was achieved by creating a new node, which given a python script, pipes all the input to the node into python, runs the scripts and read back the data again to c++ and send it through to the rest of the Visdom application. While this certainly represents some overhead, some nodes will simply not be on the performance critical path. In the case of the optimization node this is the case since the vast majority of the time will be spent running the water simulation. Finding a new parameters to evaluate will not take up a significant amount of time.

3.4.2 AutoTracks

For running simulations, the Visdom application uses the concept of tracks. Usually these tracks are created by the users with the desired parameters. Instead these tracks must now also be possible to create from the nodes directly. To solve this problem a new node called the AutoTracks node was created, this node is given a name of the new track as well as a list of floats representing the new parameter values to be tested. It then creates a new track where it changes the setting of a connected values node to represent the input it got.
The Visdom application only executes lazily, ie. it does not run a track unless the user requests it. To make sure that the new track is executed, the auto tracks node also updates a data series node which makes a data series that includes the newly created track, finally the auto tracks node request itself to be run again, but now since its input has changed a new simulation is run. The flow diagram from visdoms design view can be seen in fig 3.3.

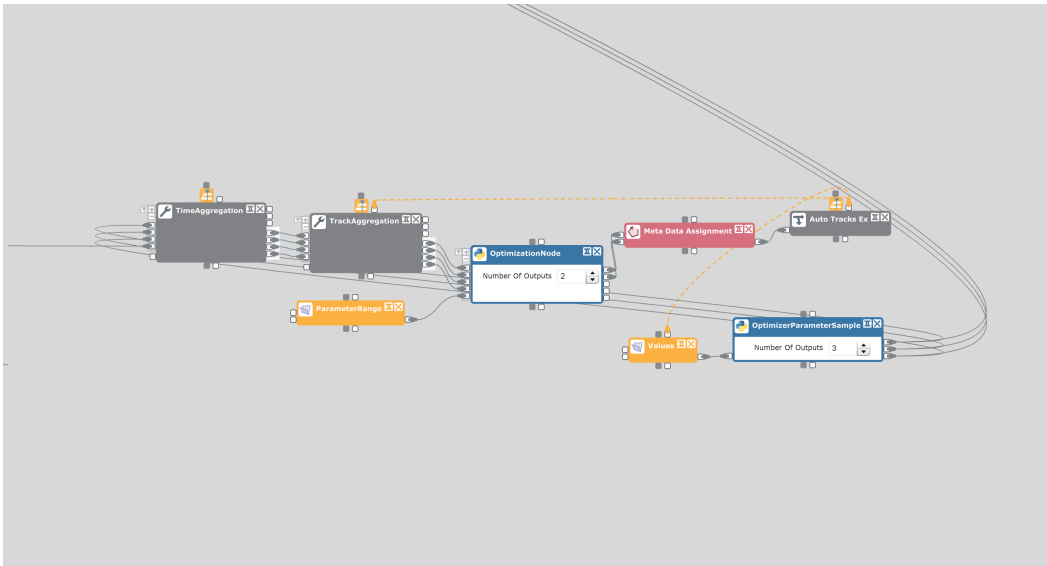


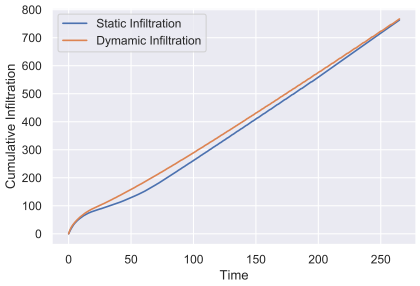
Figure 3.3: The flow diagram in the Visdom design view of the optimization.

4

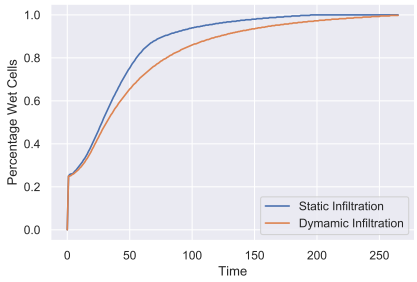
Results

4.1 Infiltration

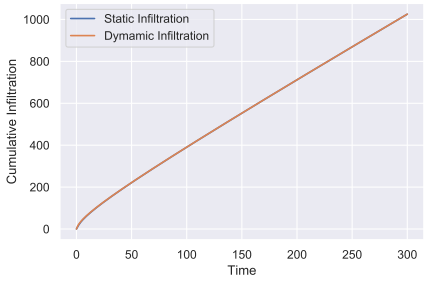
4.1.1 Dynamic vs Static Infiltration



(a) Cumulative Infiltration



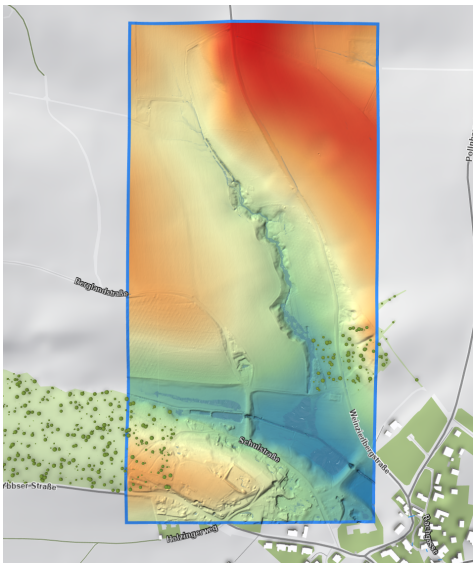
(b) Percentage of cells that are wet.



(c) Cumulative Infiltration with constant precipitation.

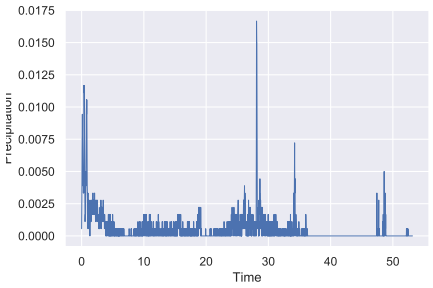
Figure 4.1: Comparison of the static and dynamic infiltration models for a slope with constant inclination.

To evaluate the importance of the dynamic infiltration model we test it on a simple scenario that clearly illustrates the problems with the static infiltration model. A sloped plane is created so that the water movement has a large effect, then precipitation is added at the top of the slope. The infiltration problem cannot be solved without first simulating the surface flow. If we use the approach we detailed before, where we assume there to be constant precipitation everywhere the results are not correct. See figure 4.1, in figure 4.1a, we see the cumulative infiltration which appears to be quite similar for both models although still differing. However, if we

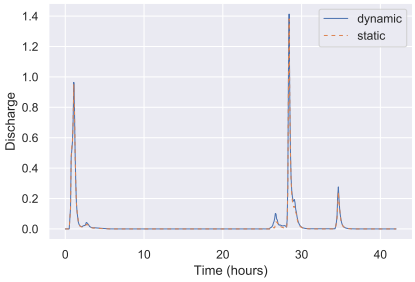


(a) The heightfield of the simulation domain.

(b) A frame form the simulation showing where the water accumulates during the periods of high precipitation.



(c) The measured precipitation of a storm event in july 2016.



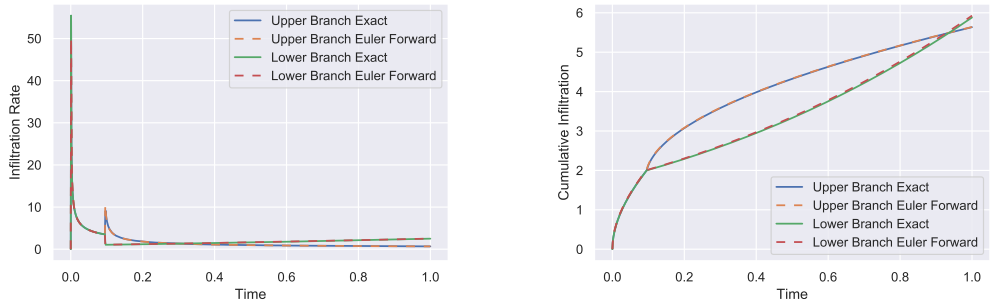
(d) The simulated discharge over the storm event for the static and dynamic infiltration model with the same parameters.

Figure 4.2: Comparison of the static and dynamic infiltration models for a storm event from july, 2016 in Petzenkirchen.

look at the percentage of the cells in the simulation which are wet, figure 4.1b, we see a large difference. Since the number of wet cells in this simulations describes how far along the slope the water has flowed this constitutes a large difference in the behavior of the simulation. Further, this demonstrates that only observing the infiltration may be deceiving since differences in the surface flow may hide the difference in subsurface flow. Finally, in 4.1c we see the reference if the infiltration happened over the full area, where both the static and dynamic infiltration model behave almost identically.

We also evaluate the the static and dynamic infiltration model on a real world scenario with data gathered from Petzenkirchen in lower Austria during a storm event in july 2016. In figure 4.2 we can see the setup as well as the result of the experiment, in particular, in 4.2d we see that while there are differences between the discharge of the simulation with static and dynamic infiltration models they are small. The quality of the simulation will not be drastically improved in this case by using the dynamic infiltration model since both the precepitation model and the infiltration model is constant over space.

4.1.2 Euler Forward vs Exact



(a) Infiltration Rate **(b)** Cumulative infiltration

Figure 4.3: Comparison between the different solver, and the solutions they produce for two different, Two Layer Green-Ampt model. Both branches share the same crust, but the soil properties differ.

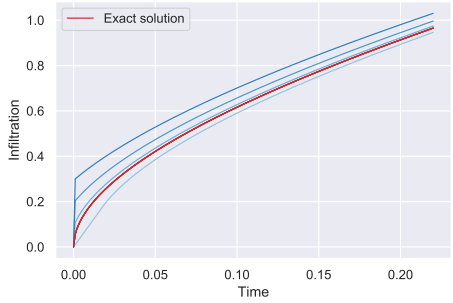


Figure 4.4: The effect of varying the maximum infiltration rate for the Euler-Forward solver, compared to the solution given by the exact solver.

To test the differences between the euler forward and exact solver we set up another simple scenario, which is flat and has fixed water depths. There, two different soils are simulated, sharing the same crust. This is done both for the exact solver and for the euler forward solver, see fig 4.3. Both solvers show good agreement. However, to get this good agreement the maximum infiltration rate must first be tuned see fig 4.4 for an even simpler one layer example of this. In 4.4 we see that changing the maximum infiltration rate can have large effects on the cumulative infiltration. However, in some real world cases this is not as much of an issue as the infiltration rate is also limited by the available water. In real world cases we will likely have some precipitation on all areas before a sufficiently large body of water gets there for this to cause an issue. However if this is not the case and large amount of water quickly moves into dry areas, the simple euler forward solver will not preform well. In such cases even if one attempts to tune the maximal infiltration rate it may not be sufficient as different rates may be needed to be provided at different locations for it to behave correctly. At that point, simply solving the model exactly might be the easier solution.

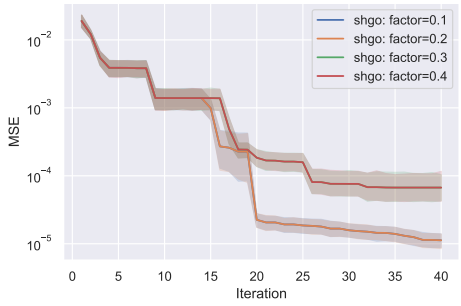


Figure 4.5: The performance of the SHGO algorithm on the test function while varying the fraction used in the global part of the optimization. Each experiments is evaluated 100 times. Here the lower branch corresponds to the factor of 0.1 and 0.2 and the upper branch corresponds to 0.3 and 0.4. The reason for the overlap is that the number of iterations in the first phase is only treated as a hint, so in this example both 0.1 and 0.2 will almost always do the same exact thing and so will 0.3 and 0.4.

4.2 Optimization

To test the optimization of the infiltration parameters a simple test case was chosen such that the simulation can repeated many time to give significance to the differences between different approaches. Further it is important that the test case resembles the function over which we will run the final parameter optimization. To this end the test chosen was to solve the inverse problem of the Green-Ampt infiltration. That is to say, finding the parameters which gives rise to a particular infiltration curve. The Green-Ampt parameters to find are chosen randomly and uniformly from a range of possible parameter values. For each set of Green-Ampt parameters the inverse problem is solved and the mean squared error of the resulting infiltration curve compared to the correct infiltration curve is computed. A new set of parameters is then chosen and the process repeated. The parameters are chosen uniformly in the following domain: $K \in (0.01, 2)$, $\Psi \in (0, 10)$, $\theta \in (0.01, 1)$ and the infiltration curve is computed for twenty equally spaced timesteps in the interval $t \in (0, 1)$.
For the SHGO algorithm, described in section 2.6.6, the implementation which we use from [29] does not include a way to limit the total number of iterations, but it does include a way to approximately limit the number of iterations that is spent in the first part of the optimization problem where it attempts to find different regions which are approximately convex. To be able to fairly compare it with other methods we treat the fraction of the iterations which is used in the first optimization phase as a hyper parameter, the results can be seen in figure 4.5. For future testing when comparing SHGO with the other algorithms we will set this fraction to 0.15.

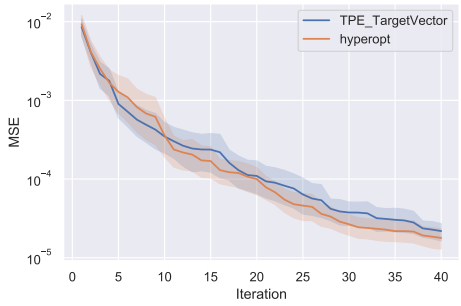


Figure 4.6: Comparison of the default TPE provided by hyperopt, and an implementation of the method modified for target vectors which was described in 2.6.5.2.

In section 2.6.5.2 a version of TPE for target vectors was derived, in figure 4.6 we see the results of this model. The vanilla TPE algorithm provided by hyperopt preforms better than the version that also takes multiple outputs into account, therefore this method will be abandoned. One likely reason why this did not work is that to get good estimates for l(x) and g(x), γ should be set such that the y is approximately split into half, however if all components are on the same scale this means that we are optimizing for the expected improvements over the average value per component. This is approximately the number of components that we use times smaller than what would like it to be, which means that we only search for values that are much

smaller than what has previously found, ie we are exploring too much. Another possible reason, caused by the γ_ith percentile of y_i needing to be the same for all components is that we only have accurate estimates for $l(x)$ and $h(x)$ for a few components that the γ_ith percentile of y_i splits roughly equally. A possible avenue for future exploration for target vector TPE would be to try and derive a method that allows arbitrary splits for each component.

Finally the performance of different methods are compared on the previously described problem, the result of which can be seen in figure 4.7. The bayesian optimization method using gaussian processes and the $\ell 2$-norm, 'bo-l2', which has previously been described in detail performs much better than all other algorithms. As such we will focus only on this method.

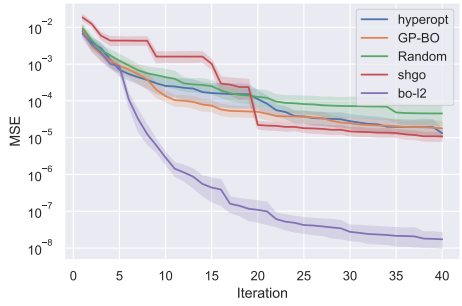
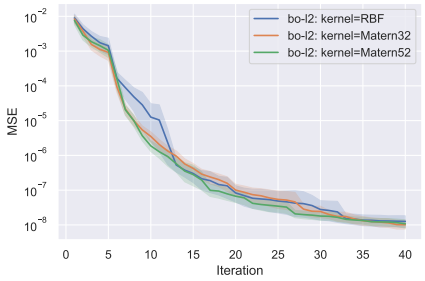


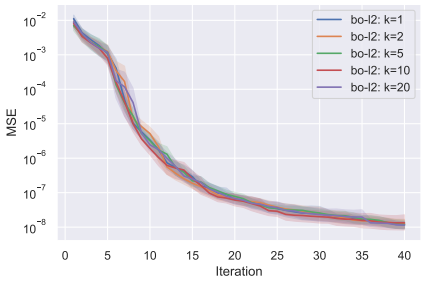
Figure 4.7: Comparison of a different optimization methods that has previously been discussed. 'hyperopt' is the implementation of TPE form the popular python library hyperopt, 'GP-BO' is Bayesian Optimization with Gaussian Processes here using the expected improvement acquisition function. Random, is a baseline of the performance from choosing random points, 'SHGO' is the SHGO algorithm previously discussed, and finally bo-l2 is the bayesian optimization algorithm using a gaussian processes to model each component and the acquisition function is the expected improvement of their $\ell 2$-norm as discussed previously.

For BO-l2 we run the experiment for the different kernels and find that the Matern kernels appear to perform slightly better for very low number of iterations but that they all perform approximately equivalently, see fig 4.8a. We chose the Matern-52 in future experiments.

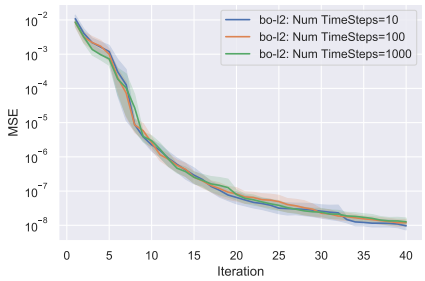
In section 2.6.5.1 it was asserted that the number of degrees of freedom in the non-central chi-square distribution used to estimate the expected improvement, may be to high. To this effect we test if changing the number of degrees of freedom has a large effect on the performance of the algorithm see fig. 4.8b. We also test the how the number of timesteps used affects the performance in 4.8c, We conclude that the number of degress of freedom does not have a large effect on the performance of the algorithm. As such we will continue to use the same number of degrees of freedom as there are timesteps as was done originally in [26].



(a) The performance of different kernels on the test function. The five first samples are chosen randomly which account for the sudden increase in performance thereafter. Each experiments is evaluated 100 times.



(b) The performance of different number of degrees of freedom used in the acquisition function.



(c) The performance of different number of timesteps for the l2-bo algorithm. With the number of degrees of freedom equal to the number of timesteps as in [26]

Figure 4.8: Performance of the Baysian Optimization For Target Vector optimization method for the different parameters.

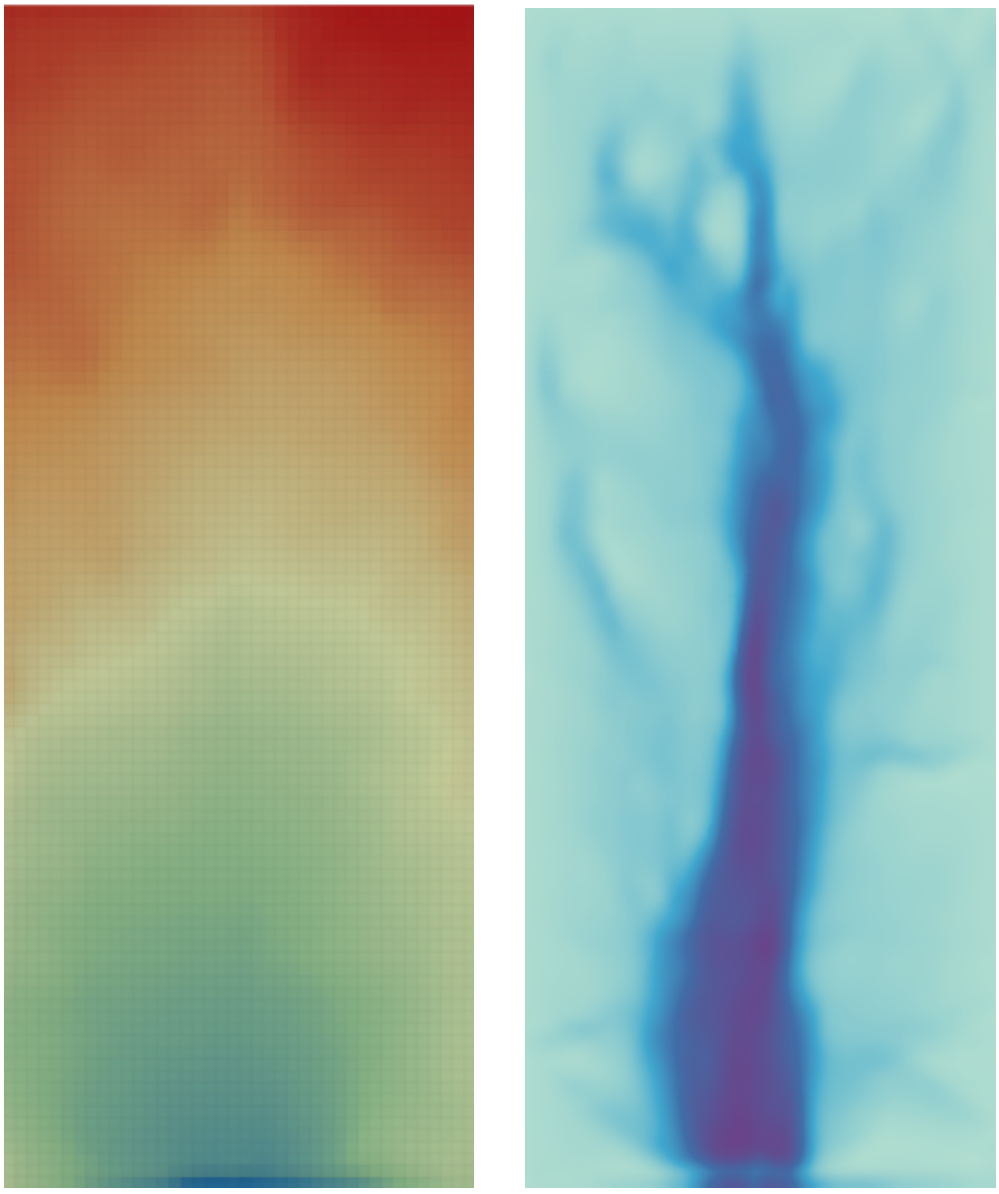
4.2.1 Inverse Problem for Runoff in Thiès

The optimization procedure, previously outlined, is evaluated on a small real world case, captured in Thiès, Senegal from [30]. The test case consists of a small 10m by 4m simulation plot. It is originally captured to calibrate roughness models, as such they provide the velocities of the water at several points in time and space, however we do not have access to the runoff information from for example hydrographs. A plot of the heightfield as well as the typical fluid flow from a simulation can be seen in figure 4.9.

Nevertheless we use the height-field provided, along with the roughness parameters that they recommend. The parameters that the algorithm should find is $K = 0.005$, $\Psi = 30$, $\theta = 0.3$. The search interval is $K \in (0, 0.01)$, $\Psi \in (0, 1000)$, $\theta \in (0, 1)$. Which includes the parameters for most naturally occurring elements.

The results can be seen in figure 4.10d. Parameters matching very closely to the

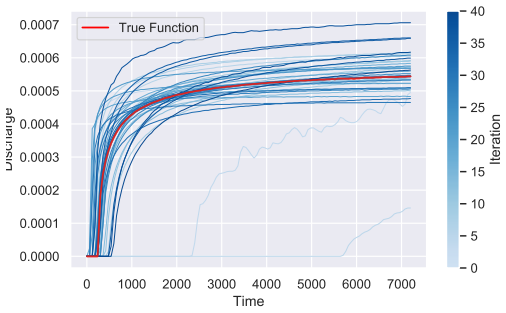
ones used in the original simulation is found in 29 iterations.



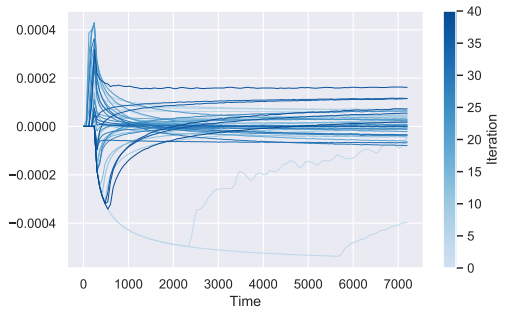
(a) The bed elevation from the Thiès dataset, per simulated cell

(b) The resulting water depths after running the water simulation for 2 simulated hours.

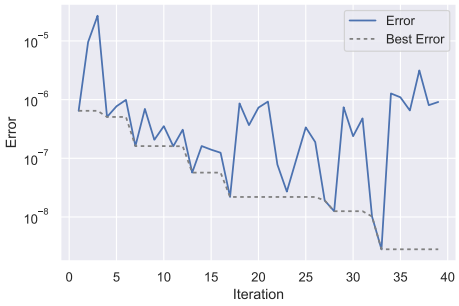
Figure 4.9: Images captured from running the simulation on the thies data set.



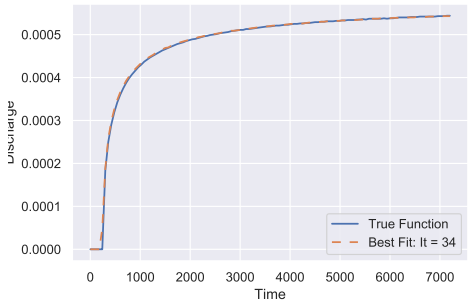
(a) All of the tested parameters.



(b) The error of all of the tested parameters.



(c) How the error of the best fit evolves over the optimization run.



(d) The best fit found during the optimization, compared with the ground truth function.

Figure 4.10: The progress of a optimization run over the infiltration parameters of the Thiès dataset.

5
Closure

5.1 Conclusion

The aim of this thesis was two fold. Firstly to develop a fast algorithm for a sufficiently accurate infiltration model and integrate into the existing visdom system. Secondly to create a procedure to optimize the infiltrator parameters in such a way that it is feasible ot do even for large scale optimization problems.
For the first part of he aim, two algorithms where implemented, one exact solution that iteratively solves the Green-Ampt model, and one simple euler forward solver. Both model show good agreement except for when the euler forward solution diverges when the cumulative infiltration is zero. In an application where this scenario is unlikely, the simple euler forward approach is likely sufficiently accurate, in particular if the true infiltration parameters are unknown.
For the second part of the aim many different optimization algorithms where evaluated and the Bayesian Optimization using Gaussian Processes with a target vector performed much better than anything else. Using this algorithm it seems possible to find good infiltration parameters in around twenty simulations, which is feasible even for large scale simulations.

5.2 Future Work

5.2.1 Acquisition Functions

More work can be done to improve the optimization algorithm, the main algorithm used here from Uhrenholt and Jensen [26] was published fairly recently and more improvements can likely be made. In particular only two acquisition functions were evaluated in the original paper, the expected improvement and the lower confidence bound, however for many types of problem better alternatives exist. In particular EI has been shown to explore too little and, for example TTEI, has for many functions shown to preform better. To this effect one might hope to be able to derive atleast an approximate solution to this acquisition function. However finding an exact solution might be hard since this amounts to calculating the truncated mean for the distribution generated by taking the difference of two scaled and correlated non random variates distributed according to the non central chi square distribution. The difference between random, uncorrelated and unscaled variates from the non central chi square distribution are known to be distributed according to the variance gamma distribution. The distribution the scaled and correlated case must therefore

be able to represent both the non central chi-square distribution as well as the variance gamma distribution. However, as previously noted we must only be able to solve this problem approximately, as such as long as we have sufficiently good approximations we could still get an improvement over the Expected Improvement that we use here.

5.2.2 Infiltration Parameters

We only support infiltration parameters that are constant in space for the optimization procedure, while the infiltration supports parameters varying in space. Finding good ways to represent spatial variability while still keeping the number of parameters needed to be estimated low deserves some more work. Previous work has achieved this through splitting the space into multiple regions each with constant parameters, however there might be other parametrization that work better.
Further, increasing the number of parameters might change which optimization procedure work best, since many algorithms struggle when the dimensionality increases, and some might be able to handle this better than others.

Bibliography

[1] O. Delestre, F. Darboux, F. James, C. Lucas, C. Laguerre, and S. Cordier, "Fullswof: Full shallow-water equations for overland flow," 2017.

[2] J. Fernández-Pato, D. Caviedes-Voullième, and P. García-Navarro, "Rainfall/runoff simulation with 2d full shallow water equations: Sensitivity analysis and calibration of infiltration parameters," *Journal of Hydrology*, vol. 536, pp. 496–513, 2016.

[3] Y. Wang, G. Sang, C. Jiao, Y. Xu, and H. Zheng, "Flood simulation and parameter calibration of small watershed in hilly area based on hec-hms model," *IOP Conference Series: Earth and Environmental Science*, vol. 170, p. 032 093, Jul. 2018. DOI: 10.1088/1755-1315/170/3/032093.

[4] J. Dennis and D. J. Woods, "Optimization on microcomputers: The nelder-mead simplex algorithm," *New computing environments: microcomputers in large-scale computing*, vol. 11, pp. 6–122, 1987.

[5] C. Zhang, R.-b. Wang, and Q.-x. Meng, "Calibration of conceptual rainfall-runoff models using global optimization," *Advances in Meteorology*, vol. 2015, 2015.

[6] Q. Duan, S. Sorooshian, and V. K. Gupta, "Optimal use of the sce-ua global optimization method for calibrating watershed models," *Journal of hydrology*, vol. 158, no. 3-4, pp. 265–284, 1994.

[7] Q. Duan, S. Sorooshian, and V. Gupta, "Effective and efficient global optimization for conceptual rainfall-runoff models," *Water resources research*, vol. 28, no. 4, pp. 1015–1031, 1992.

[8] Y. Ding, Y. Jia, and S. S. Wang, "Identification of manning's roughness coefficients in shallow water flows," *Journal of Hydraulic Engineering*, vol. 130, no. 6, pp. 501–510, 2004.

[9] H. Darcy, *Fontaines Publiques de La Ville de Dijon*. Libraire des Corps, 1856.

[10] W. Green and G. Ampt, "Studies on soil physics, 1. the flow of air and water through soils," *J. Agric. Sci*, vol. 4, no. 1, pp. 1–24, 1911.

[11] D. Barry, J.-Y. Parlange, L Li, D.-S. Jeng, and M. Crapper, "Green–ampt approximations," *Advances in Water Resources*, vol. 28, no. 10, pp. 1003–1009, 2005.

[12] M. Esteves, X. Faucher, S. Galle, and M. Vauclin, "Overland flow and infiltration modelling for small plots during unsteady rain: Numerical results versus observed values," *Journal of hydrology*, vol. 228, no. 3-4, pp. 265–282, 2000.

[13] W. Rawls, D. Brakensiek, and K. Saxton, "Estimation of soil water properties," *Trans. Asae*, vol. 25, no. 5, pp. 1316–1320, 1982.

[14] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "Sympy: Symbolic computing in python," *PeerJ Computer Science*, vol. 3, e103, Jan. 2017, ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. [Online]. Available: https://doi.org/10.7717/peerj-cs.103.

[15] W. Kahan, "Pracniques: Further remarks on reducing truncation errors," *Communications of the ACM*, vol. 8, no. 1, p. 40, 1965.

[16] M. Stein, *Interpolation of Spatial Data: Some Theory for Kriging*. Springer, 1999.

[17] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005, ISBN: 026218253X.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] C. G. Broyden, "The convergence of a class of double-rank minimization algorithms: 2. the new algorithm," *IMA journal of applied mathematics*, vol. 6, no. 3, pp. 222–231, 1970.

[20] C. Schenck and D. Fox, "Spnets: Differentiable fluid dynamics for deep neural networks," *arXiv preprint arXiv:1806.06094*, 2018.

[21] C. Qin, D. Klabjan, and D. Russo, "Improving the expected improvement algorithm," in *Advances in Neural Information Processing Systems*, 2017, pp. 5381–5391.

[22] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the lipschitz constant," *Journal of optimization Theory and Applications*, vol. 79, no. 1, pp. 157–181, 1993.

[23] K. Kandasamy, K. R. Vysyaraju, W. Neiswanger, B. Paria, C. R. Collins, J. Schneider, B. Poczos, and E. P. Xing, "Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly," *arXiv preprint arXiv:1903.06694*, 2019.

[24] T. G. authors, *GPyOpt: A bayesian optimization framework in python*, `http://github.com/SheffieldML/GPyOpt`, 2016.

[25] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

[26] A. K. Uhrenholt and B. S. Jensen, "Efficient bayesian optimization for target vector estimation," in *Proceedings of Machine Learning Research*, K. Chaudhuri and M. Sugiyama, Eds., ser. Proceedings of Machine Learning Research, vol. 89, PMLR, 2019, pp. 2661–2670. [Online]. Available: `http://proceedings.mlr.press/v89/uhrenholt19a.html`.

[27] S. C. Endres, C. Sandrock, and W. W. Focke, "A simplicial homology algorithm for lipschitz optimisation," *Journal of Global Optimization*, vol. 72, no. 2, pp. 181–217, 2018.

[28] O. Delestre, F. Darboux, F. James, C. Lucas, C. Laguerre, and S. Cordier, *Fullswof: A free software package for the simulation of shallow water flows*, 2014. arXiv: `1401.4125 [math.AP]`.

[29] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, bibinitperiodI. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: `https://doi.org/10.1038/s41592-019-0686-2`.

[30] C Mügler, O. Planchon, J Patin, S. Weill, N. Silvera, P. Richard, and E Mouche, "Comparison of roughness models to simulate overland flow and tracer transport experiments under simulated rainfall at plot scale," *Journal of Hydrology*, vol. 402, no. 1-2, pp. 25–40, 2011.