NTNU – Trondheim
Norwegian University of
Science and Technology

A PDE Based Approach to Path Finding in Three Dimensions

Solving a Path Finding Problem for an
Unmanned Aerial Vehicle

Stian Engebretsen

Master of Science in Physics and Mathematics
Submission date: June 2014
Supervisor: Einar Rønquist, MATH
Co-supervisor: Kristin Paulsen, KONGSBERG

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

This thesis presents a general three-dimensional method for pathfinding, based on a partial differential equation. The method relies on a grid with hazard-values, describing the risk associated with every point in the domain. Analogous to a fluid flow problem, we construct an artificial permeability based on the hazard-values, and we use this to calculate streamlines that constitute the potential paths from a starting point α to the target β. We investigate the different parameters and ways to manipulate the problem to yield sufficiently flyable streamlines.

The method is geared towards finding a terrain-following, flyable path for an unmanned aerial vehicle(UAV) through a hostile terrain. In special, we consider the potential for a program implementation to run on-board the UAV during mission flight. For this application, the available memory and processor resources can be restricted. This sets strict requirements on the pathfinding algorithm. Particularly fast solvers exist for solving PDE's discretized using finite differences on regular grids. We implement a multigrid method for the resulting linear set of equations, with optimal memory usage, linear complexity and a potential for parallelization.

Sammendrag

I denne masteroppgaven presenteres en generell tredimensjonal metode for ruteplanlegging, basert på en partiell differensialligning. Metoden er avhengig av et grid med hasardrater, som beskriver risikoen assosiert ved hvert punkt innenfor domenet. Analogt med et fluidproblem konstruerer vi en kunstig permeabilitet basert på hasardratene, og denne brukes til å beregne strømlinjer mellom punkt α og punkt β. Blant strømlinjene velges en bane som løsning. Vi undersøker nærmere de ulike parametre og innfallsvinkler for å manipulere problemet slik at vi kan finne tilstrekkelig flybare strømlinjer.

Metodens mål er å finne en flybar bane for en drone gjennom et fiendtlig terreng. Spesielt vurderer vi potensialet for å implementere et program som kan kjøres på dronen under oppdrag. For denne problemstillingen kan det være begrenset ledig minne- og prosessorkapasitet, og dette setter strenge føringer på ruteplanleggeren. Det finnes raske løsere for differensialligninger som er diskretisert med endelig differansemetoder på regulære grid. Vi implementerer en multigridmetode som løser det resulterende ligningssettet med optimal minnebruk, lineær kompleksitet og et potensial for parallelisering.

Preface

This thesis concludes my master's degree in Industrial Mathematics in the field of numerics at the Norwegian University of Science and Technology(NTNU). The work was performed in the spring of 2014 as a continuation of a project started during the fall of 2013. The thesis was supervised under Professor Einar Rønquist(NTNU) and Kristin Paulsen(KONGSBERG)

I would like to thank KONGSBERG for giving me the opportunity to work with an exciting real-world application, and, in special, Kristin Paulsen for giving me feedback on my text and source code. Thanks to Einar Rønquist for taking on the role as supervisor along side the position as Head of the Mathematical Institute at NTNU.

I would also like to compliment Tim Wendelboe on roasting the best possible coffee, and Ben Harper for the musical inspiration he provided through the final stages of this thesis.

Stian Engebretsen
NTNU, Trondheim
June 10, 2014

Contents

Abstract i

Sammendrag iii

Preface v

Notation xiii

List of Figures xvi

1 Introduction 1
1.1 Motivation . . . 1
1.2 Overview of the Thesis . . . 4

2 Theory 5
2.1 Problem Statement . . . 5
2.2 Minimization Formulation . . . 6
2.3 Hazard . . . 7
2.4 PDE Approach . . . 9
2.5 Partial Differential Equations . . . 9
2.6 Solving PDE's . . . 11
2.6.1 Discretization . . . 11
2.6.2 Finite Difference Discretization . . . 13
2.7 Solving Linear Sets of Equations . . . 14
2.7.1 Direct Methods . . . 15
2.7.2 Iterative Methods . . . 15
2.8 The Physical Analogy Method . . . 24
2.8.1 The PAM Equation . . . 25
2.8.2 Discretization . . . 25
2.8.3 Modified Hazard . . . 26

2.8.4 Extracting a Path from the Solution . . . 27
2.8.5 Boundary Conditions . . . 28
2.8.6 Source Term vs. Boundary Condition . . . 29
2.8.7 Exclusion Zones . . . 30
2.8.8 Optimality of the Solution . . . 30
2.8.9 Computational Domain . . . 31
2.9 Graph Algorithm . . . 34
2.9.1 Dijkstra on Cost Grid . . . 35
2.9.2 Preprocessed Graph . . . 36
2.10 Post-Processing . . . 36
2.11 Code Environment . . . 36

3 Implementation 39
3.1 Linear Solver in MATLAB . . . 40
3.2 The Multigrid Method . . . 40
3.2.1 Matrix-Free Relaxation . . . 40
3.2.2 Conjugate Gradient . . . 42
3.3 Path Evaluation . . . 42
3.4 Dijkstra on Regular Grid . . . 43

4 Results 45
4.1 The PAM with Homogeneous Neumann BC . . . 45
4.1.1 Isotropic Permeability . . . 45
4.1.2 Anisotropic Permeability . . . 46
4.1.3 Processed Permeability . . . 46
4.1.4 Evaluating Paths . . . 47
4.1.5 Optimality Tests . . . 51
4.1.6 Exclusion Zones . . . 51
4.1.7 Permeability-Mapping . . . 51
4.1.8 Permeability Cut-Off . . . 53
4.2 The PAM with Homogeneous Dirichlet BC . . . 56
4.2.1 The Multigrid Method . . . 56
4.2.2 Avoiding the Boundary . . . 61
4.3 Dijkstra's Algorithm on Regular Grid . . . 63
4.4 Preprocessed Graph . . . 64
4.5 Post Processing . . . 67

5 Discussion 69
5.1 Streamline Properties . . . 69
5.1.1 Optimality . . . 69
5.1.2 Boundary Conditions . . . 70
5.1.3 Evaluating Paths . . . 70

5.2 The Multigrid Solver 71
5.2.1 Parallelization 71
5.3 Extension to 3D 71
5.4 Error from Grid Simplification 72
5.5 Runtime 72

6 Conclusion **73**

Bibliography **75**

Notation

(ξ, η) Reference variables.

α Starting point.

β Target location.

Δ Error associated with the computational domain.

κ The thermal conductivity.

λ Represents a general eigenvalue of a matrix.

$\mathcal{F}$ A mapping from Ω to the reference domain $\hat{\Omega}$.

$\mathcal{J}$ The jacobian of a transformation.

$\mathcal{K}$ A Krylov subspace.

$\mathcal{O}(g)$ We say $f = \mathcal{O}(g)$ as $x \to x_0$ provided there exists a constant C such that $|f(x)| \leq C|g(x)|$ for all x sufficiently close to x_0.[4]

μ The viscosity of a fluid.

ν The number of streamlines calculated. In the multigrid algorithm we use ν_1 and ν_2 to represent the number of pre- and post-smoothings.

Ω The domain all paths between α and β can be in.

ω Parameter used in the weighted Jacobi method.

$\partial\Omega$ The boundary of the domain Ω.

Φ The pressure in a fluid.

ϕ The porosity in a porous medium.

ρ The density of a fluid.

σ Source function for the heat equation.

$\boldsymbol{n}$ Normal vector pointing out of Ω.

$\boldsymbol{u}$ A velocity vector field.

ζ The curvature of a path.

A Represents a matrix for a linear set of equations.

$C^n(\Omega)$ Space of continuous functions over Ω such that the 1st through nth order derivatives are continuous.

COL Cut-off-level.

e The error in a numerical solution.

f A general source function in a PDE.

$G(V, E)$ A graph with a set of vertices V, and a set of edges E. G is also used to denote an iteration matrix.

H Scalar three dimensional cost grid containing risk or hazard data.

I_{2h}^{h} An intergrid operator, $2h$ and h can also be switched.

K An artificial permeability based on the cost grid H.

k The permeability of a fluid.

K_c Cut-off-value.

L Length in general. L_x, L_y and L_z represents the length scales of the domain Ω.

M A pre-conditioner.

m The total number of unknowns used in general.

N Even number for the size of the grid used in the multigrid algorithm.

P An arbitrary path through Ω.

$P^\star$ The optimal path.

P_{opt} Approximation to the optimal path $P^\star$.

r The residual.

$S(P)$ Accumulated hazard over the path P.

s_i A set of seed points for the streamlines.

$S_N(P)$ Numerically summed hazard over the path P.

T The temperature in the heat equation.

u Symbolizes the solution of an equation in general.

v Used in general to denote a vector. In the context of graphs, we use it as a particular vertex $v \in V$.

w A weight function for the graph G.

List of Figures

1.1 A simple graph with seven vertices and ten edges 2
1.2 The Oslo fjord; a part of the terrain used in this thesis 3

2.1 A regular grid with varying step size in z-direction 7
2.2 The topography and corresponding cost-grid with hazard values . . 8
2.3 Two unstructured finite element grids 13
2.4 Two-dimensional central difference stencil 14
2.5 A grid transfer operation: interpolation from Ω_{2h} to a finer grid Ω_h 20
2.6 A scheme illustrating the V-cycle iteration 23
2.7 The domain illustrated with two exclusion zones 30
2.8 The mapping between physical domain Ω and the reference domain $\hat{\Omega}$. 33
2.9 One-dimensional plot of the elevation 34
2.10 The stencil for coupling neighbouring nodes in the Dijkstra implementation 35
2.11 Smoothing the altitude component of a three-dimensional path . . 37

4.1 The PAM with uniform permeability 46
4.2 The PAM with unprocessed permeability 47
4.3 The PAM with processed permeability 48
4.4 Varying the step length for the streamline integration 49
4.5 The PAM with different streamline selection strategies 50
4.6 Plot of altitude for paths with different streamline selection strategies 50
4.7 Optimality test: test of sub path 51
4.8 Optimality test: is $P_{\alpha\to\beta} = P_{\beta\to\alpha}$? 52
4.9 The PAM with an exclusion zone 52
4.10 Testing of the permeability-mapping 54
4.11 Testing the cut-off-level 55
4.12 Numerical solution of a Poisson test problem 57
4.13 Multigrid convergence: homogeneous test problem 58

4.14 The discretization error for the test problem 58
4.15 Multigrid convergence: inhomogeneous test problem 59
4.16 Multigrid runtime 60
4.17 Testing the PAM with homogeneous Dirichlet boundary conditions 62
4.18 Contour plot of three problems solved with the PAM and Dijkstra 63
4.19 Comparison of the PAM and Dijkstra including elevation 64
4.20 Dijkstra runtime 65
4.21 The edges in the graph found with the PAM 66
4.22 3D visualization of a terrain-following flyable path 67
4.23 Large 3D visualization 68

Chapter 1

Introduction

1.1 Motivation

The task of revealing the shortest path between two points, or pathfinding is a very common problem in many fields of application. Although we talk about the shortest path, this does not have to represent physical distance. In general, shortest path-algorithms can be used to solve problems where we want to minimize a quantity between two points. In practice, many problems can be transformed into a shortest path-problem, and we enjoy the benefits of these types of applications on a daily basis:

The obvious application of the shortest path-problem is for transport route planning. Websites such as google maps use specialized shortest path-algorithms to find the best route between two destinations. In other applications shortest path algorithms can be used to solve sub-problems of a bigger problem, such as when you compile a TEX-document, shortest path-algorithms are used to typeset the text[11]. This means the typesetting of this thesis has at some level been done with a shortest path algorithm.

Numerous algorithms have been invented with the aim of solving shortest path-problems on graphs. In graph theory, we define the graph $G(V, E)$ where V is the set of all vertices, and E is the set of all edges between the vertices. The graph has an associated weight function $w : E \to \mathbb{R}$, mapping edges to real-valued weights. The path trough G which minimizes the sum of all the weights along its edges is the solution to the shortest path problem. In figure 1.1 we see a simple example of a graph where the shortest path between vertex a and g is indicated by the red weights.

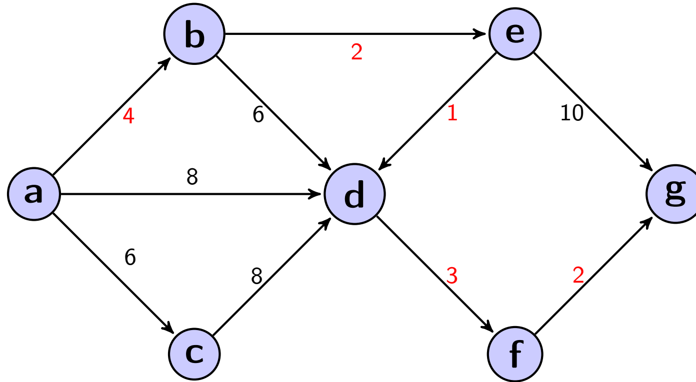


Figure 1.1: A simple graph with 7 vertices and 10 edges. The shortest path between a and g is indicated with red weights

In 1959, Dijkstra invented his famous algorithm, which carries his name. Dijkstra's algorithm solves the single-source shortest path problem on a weighted directed graph $G = (V, E)$ for the case in which all edge weights are non negative[10]. There are many versions of Dijkstra's algorithm and some can achieve a very good runtime complexity. However, we must be able to represent the problem as a graph.

In this project, we want to solve a path finding-problem; taking an Unmanned Aerial Vehicle(UAV) with minimal risk from point α to point β. The path should exploit the topography for stealth and avoid known threats to minimize the risk. Furthermore, the UAV must follow a time- and fuel schedule, and the path must take must-fly-zones and exclusion zones into account. In addition, the path must satisfy the UAV's manoeuvrability requirements. The total problem is complex and has many technical challenges.

The aim of this thesis is to find an aerial path in three dimensions - this is a continuous problem by nature. Traditionally, path finding problems are solved with popular algorithms such as Dijkstra's and $A^{\star}$, but we need an algorithm that can run on a UAV with limited computational power in real-time. Parallel shortest path-algorithms exist, but it can be hard to achieve good speedup with these. Hence, there is a need for an efficient algorithm that scales satisfactory in parallel.

To make the task parallelizable we can exploit the fact that our problem is continuous by nature, by modelling it as a Partial Differential Equation(PDE). The motivation for using a PDE comes from its analogies to physics. One analogy is to fluid flow, and its ability to find a path of least resistance. In a porous media the fluid will tend to flow through the high permeability zones. Similarly,

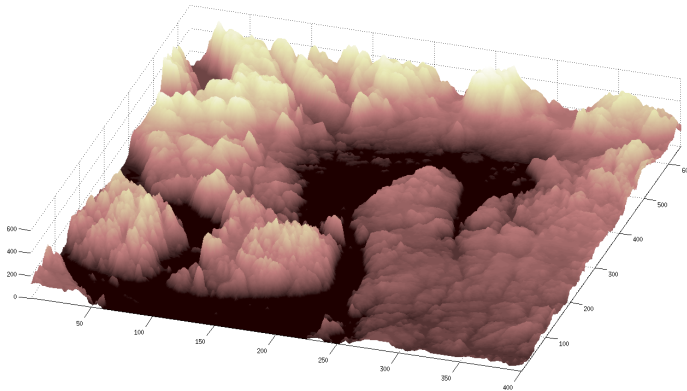


Figure 1.2: Image of an example terrain around Oslo which we will use when calculating shortest paths.

in a heat transport-problem, the energy will move in the zones of high thermal conductivity.

Due to applications in the oil industry and technology in general, a great amount of time and effort has been spent on optimizing the solution methods for fluid problems. Because of this, highly parallelizable algorithms with good complexity exist for solving such PDE's.

For our path finding-problem we are given a cost grid H, containing weights that symbolize the UAV's risk or hazard. The cost grid is based on topographical data, as seen in figure 1.2. Using information from this cost grid as an analogy to permeability or conductivity, we will model the problem analogously to a fluid or heat problem. The goal of this thesis is to solve the pathfinding problem, such that a terrain following, flyable path can be found using a fast and memory efficient algorithm.

1.2 Overview of the Thesis

Chapter 2

We cover the main theory that is relevant for the thesis. The problem is presented in detail before we cover general theory on PDE's and how to solve them. The method we have developed is presented and finally we cover some aspects of Dijkstra's algorithm.

Chapter 3

We comment on our program implementation and we list some of the essential parts of the source code.

Chapter 4

The results from our methods and implementations are presented. The results mainly revolve around testing different aspects of the pathfinding algorithm, but another big part is related to the specialized linear equation solver, the multigrid algorithm.

Chapter 5

We make some further comments on the results from the previous chapter. We discuss the properties of the set of streamlines which the path is chosen from. We point out weaknesses and suggest some improvements for future work.

Chapter 6

We present the conclusions of our method for the pathfinding problem.

Chapter 2

Theory

In this chapter, our focus will be on describing the theory behind our path finding algorithm and relevant background. The mathematics we rely on is mainly a means to do satisfactory modelling of the problem, however we will try do be as mathematically exact as possible.

2.1 Problem Statement

The key aspects which constitutes our problem is described in more detail in this section. The main aim is to find a path from α to β with minimal risk, through a hostile environment. The path will also have to fulfill the following requirements:

- **Avoid terrain:** The UAV must avoid crashing into the terrain and other physical obstacles.
- **Minimize threats:** To minimize the risk of being detected, the path should make use of the topography. Hiding in the contours of the topography, the UAV is less likely to be "seen" by radar and other devices. The path should also avoid known threats, such as missile defence systems.
- **Exclusion zones:** Some zones are to be avoided completely and the expressions exclusion zone and no-flight-zone is used interchangeably in this report to refer to these zones. An airport is one example of an exclusion zone.
- **Must-Fly-Zones:** The path must be coerced to go via certain locations we call must-fly-zones. In some cases communication with the UAV can force us to introduce a must-fly-zone along the path.

- **Time- and fuel schedule:** The UAV has limited amounts of fuel which sets restrictions on path length. In some cases the time could also be set explicitly to have an upper bound. If time is a critical element, one must sometimes take a less optimal path with respect to risk.

- **UAV Manoeuvrability:** The UAV can only accelerate, respond and turn at a certain level, and these restrictions in manoeuvrability must be considered in the path finding algorithm. This imply that an optimal path must satisfy some smoothness and curvature requirements reflecting the UAV's capabilities.

2.2 Minimization Formulation

This path finding problem can be formulated as an optimization problem. We want to find the path with minimal hazard along it, that also satisfies the problem requirements defined above. Mathematically, we do not know if there exists a unique solution, but we will assume that an optimal solution $P^\star$ of our problem exists.

For a given path $P \subset \Omega$ we define the accumulated hazard along this path as

$$S(P) = \int_P H \mathrm{d}s. \tag{2.1}$$

This line integral is a means to quantify the total risk along a given path, P. Using equation (2.1) we can find $P^\star$ by solving

$$\begin{aligned} \min_{P \in C^d(\Omega)} \quad & S(P) \\ \textit{subject to} \quad & L(P) \leq L_{max} \\ & \zeta(P) \leq \zeta_{max} \\ & \vdots \end{aligned} \tag{2.2}$$

where L is the length of P and ζ is the curvature. We are minimizing over the space of d-times continuously differentiable functions, the constant d is depending on the manoeuvrability requirements. The distance, L_{max} indicates that the path length should be below a certain maximum, both with consumption of time and fuel in mind. The length L_{max} is coupled with the curvature, since the UAV will burn fuel fast when accelerating rapidly. The ζ_{max} indicates the maximum curvature the UAV can follow, so we should expect this is intertwined with the smoothness requirement.

Problem (2.2) becomes increasingly complex to solve when we add many path requirements and constraints.

Our intuition suggests that fluids will find the path of least resistance from α to β, and this can be used as an analogy to minimize the hazard. This problem can be formulated as a PDE. However, we cannot assume that the path P_{opt} found by the PDE will yield the optimal path $P^{\star}$, which solves the problem defined in equation (2.2).

2.3 Hazard

The physical domain we are working on has a discrete representation with an associated hazard, $H(x, y, z)$ that represents the risk in each point. The domain is a box in three dimensions, where the bottom represents the physical ground. We have access to the hazard which is based on the geographical data and is preprocessed by KONGSBERG, it is represented as a regular grid with layers of varying step size in the vertical direction, and stored as a three dimensional array of hazard values. The array or matrix, is also referred to as the cost grid.

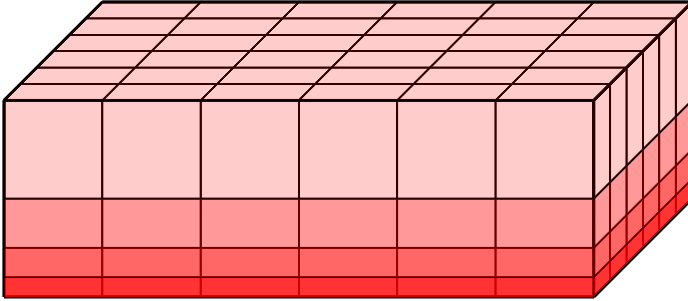
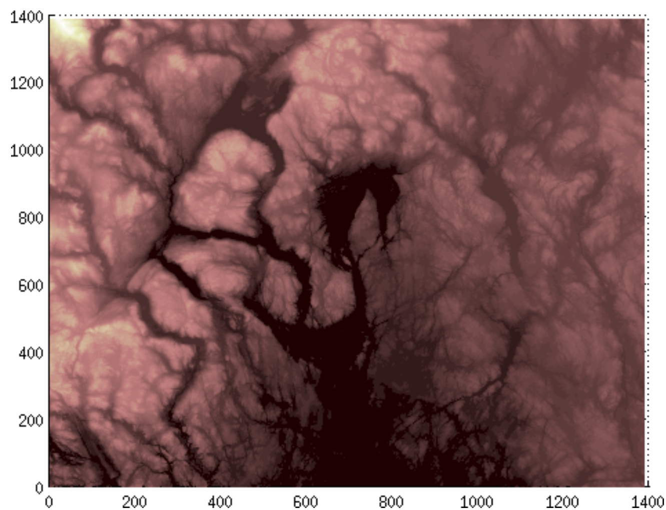


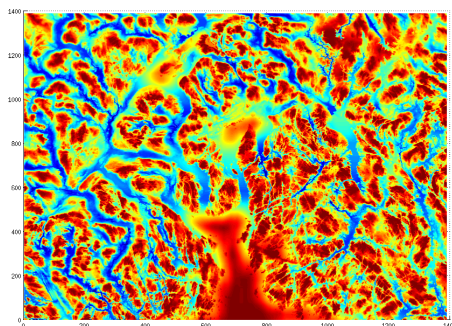
Figure 2.1: The cost grid H is a regular grid with varying step size in the vertical direction.

In figure 2.2 we have plotted the topography map and the bottom layer of the associated cost grid for the eastern part of Norway around the capital. The hazard in a point (x, y, z) is based on the visibility from the neighbouring terrain; in general $H(x, y, z)$ will increase when z increases. This is reflected in the layers in z-direction of H. The first layer in H has mean of 8.1×10^{-6}, while the second layer has a mean of 1.1×10^{-5}. Notice in figure 2.2c that large areas with open water is not very attractive, since the UAV is very visible, the valleys however, represents potentially attractive areas. Since low level flight yields the least risk, it is sufficient to only consider the first layer in the cost grid, and a

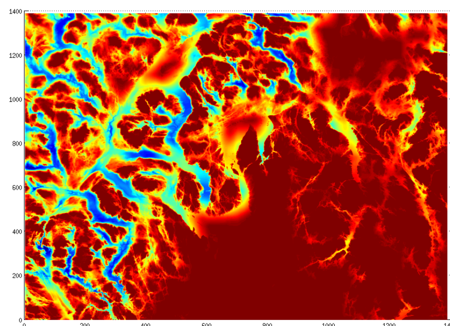
two-dimensional model. Extension to three dimensions is not complicated, but it will result in more computational time for little reward.



(a) The topography map.



(b) First layer in the cost grid.



(c) Second layer in the cost grid.

Figure 2.2: On the top, the topography is visualized where brighter colors represent greater altitude. The first two layers in the cost grid is visualized. The first layer in H has mean of 8.1×10^{-6}, while the second layer has a mean of 1.1×10^{-5}. The areas of the least risk are the blue regions in the first layer of the grid.

2.4 PDE Approach

Above we have briefly mentioned the motivation for modelling the problem by a PDE. The idea is to exploit the analogy to physics and its well developed solution methods. However, we must incorporate the above mentioned requirements into the formulation.

Initially, we expect to avoid terrain, minimize the risk, go through must-fly-zones and avoid exclusion zones simply by solving our PDE, given a sufficient formulation. Consider the problem of going from α to β via the must-fly-zone $\tilde{\beta}$. This can be split into two separate problems that can be solved successively as α to $\tilde{\beta}$, then $\tilde{\beta}$ to β. There might be problems with the smoothness of incoming and outgoing flight directions from the point $\tilde{\beta}$. An exclusion zone can be implemented by increasing the hazard for every point in the exclusion zone.

The only requirements that is quite different in nature from the rest are those on time and fuel. Ideally our solution will be a path that minimizes the hazard sufficiently, but when we introduce a time schedule we can be forced to take a more hazardous path. In general, the PDE might impose smoother solutions then graph-algorithms depending on the implementation, but the drawback can be the optimality of the solution. Depending on the specific hazard of a problem, the paths extracted from the solution may need post-processing to satisfy manoeuvrability requirements. Before we go into the details of the path finding algorithm we will present some relevant theory concerning the PDE's and how to solve them.

2.5 Partial Differential Equations

Solving PDE's numerically is a central topic in computational science. Problems in chemistry, biology and all of physics give rise to a vast number of different PDE's - these equations can be solved with high accuracy with the help of mathematics and computer science. In this section we will describe some relevant partial differential equations for this thesis.

Poisson's Equation

One of the most famous equations is without a doubt Poisson's equation. This simple equation is often used as a test equation for numerical methods, and in many cases it has an analytical solution. The equation can be presented on the following form:

$$-\nabla^2 u = f \qquad in \qquad \Omega, \tag{2.3}$$

where Ω in general denote the mathematical domain in which the equation govern. As we will see in the equations below, they also have a second order derivative,

similar to the left hand side of equation (2.3). When the source function is zero $f = 0$, in Poisson's equation, it is referred to as the Laplace equation, and the term ∇^2 is often referred to as the laplacian.

Flow in Porous Media:

By mass conservation we get the continuity equation in a porous medium;

$$(\phi\rho)_t + \nabla(\rho\boldsymbol{u}) = f \qquad in \qquad \Omega \tag{2.4}$$

where ϕ is the porosity, ρ is the density, $\boldsymbol{u}$ is the velocity and f is the source function. From the continuity equation we can derive the flow equation using Darcy's law [2]

$$\boldsymbol{u} = -\frac{k}{\mu}\nabla\Phi \qquad in \qquad \Omega, \tag{2.5}$$

where k is the permeability, μ is the viscosity and Φ represents pressure. We insert (2.5) into (2.4) and look at the steady state to obtain

$$-\nabla \cdot (\frac{\rho k}{\mu}\nabla\Phi) = f \qquad in \qquad \Omega. \tag{2.6}$$

Darcy's law is valid for creeping, incompressible flows.

The Heat Equation:

Based on conservation of energy and Fourier's Law, the anisotropic heat equation is derived. For a full derivation, see [3]. We skip the steps and show the resulting equation:

$$-\nabla \cdot (\kappa\nabla T) = \sigma \qquad in \qquad \Omega. \tag{2.7}$$

Here κ is the thermal conductivity, T is the temperature and σ is the heat source function.

Common for both the heat equation and the porous medium flow equation above is the fact that when $\frac{\rho k}{\mu}$ and κ are constants, they are both reduced to Poisson's equation, although the solution represents different physical quantities.

In the equations above we have not included *boundary conditions*, but in order for a PDE to have a solution, boundary conditions must be defined in addition do the governing equation. The governing equation is usually defined within a domain Ω, while the boundary conditions are defined at the boundary $\partial\Omega$ of the domain.

2.6 Solving PDE's

Partial differential equations are often used to model a physical process within a given domain Ω. Some equations have continuous analytical solutions, but for the most practical use, we are dependent on numerically approximated solutions.

A computer can only process discrete problems, this means the continuous solution of a PDE must be approximated by a finite amount of points. The continuous PDE, which holds for an infinite number of points in Ω must be converted into a finite linear system of equations. The process of reducing a PDE into a linear system of equations is called discretization.

2.6.1 Discretization

The most common forms of discretization techniques for solving PDE's numerically are; finite differences, finite elements, finite volume and spectral methods.

The Finite difference method is applicable for simple geometries such as rectangles and cubes, where the internal data can be well represented on a regular grid. This is the case if the values in the grid vary in a relatively smooth manner. Finite difference stencils couples information from the neighbouring points together, but there is no direct connection between data points further away. This means that many parallel algorithms are well suited for solving the resulting equation sets from finite difference discretization. These sets of equations typically result in a matrix which exhibits the following qualities:

- **Positive definiteness:** We say a matrix is positive definite if:

$$v^T A v > 0 \quad \forall v \in \mathbb{R}^m, \tag{2.8}$$

  where $A \in \mathbb{R}^{m \times m}$.

- **Diagonally dominance:** A matrix A is diagonally dominant if

$$|a_{jj}| \geq \sum_{i=1, i \neq j}^{i=m} |a_{ij}|, \qquad j = 1, ..., m. \tag{2.9}$$

- **Symmetry:** A is symmetric if $A = A^T$.
- **Sparsity:** A sparse matrix consists of primarily zeros.
- **Banded:** The non-zero elements is located on diagonal bands through the matrix.

If a matrix A is both symmetric and positive definite, we say A is *SPD*. There are several very effective algorithms that can be used to solve the resulting set of equations possessing the characteristics mentioned above, among them is the multigrid algorithm. For these reasons the Finite Difference (FD) method will be used in this thesis. Discretization using FD is described in more detail at further down in this section.

All of the discretization methods mentioned above will produce a linear system of equations to be solved. Depending on the method used, the system of equations will have different qualities. Every discretization method mentioned, except for spectral methods, will tend to produce sparse matrices. For m unknowns the discretized system will be represented on the form

$$Ax = b, \tag{2.10}$$

where $A \in \mathbb{R}^{m \times m}$ is a matrix and $x, b \in \mathbb{R}^m$ are vectors.

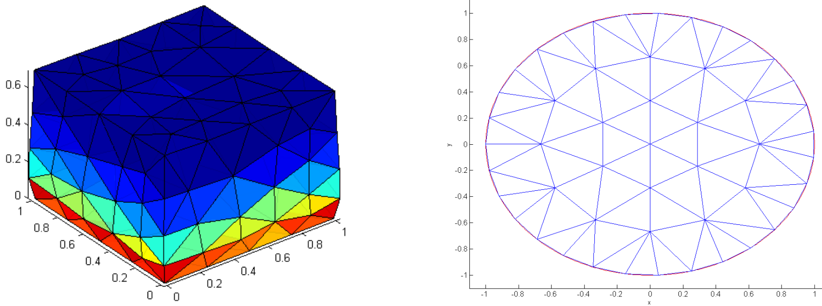
When we have defined an equation, discretized it and represented as in (2.10), the rest is a matter of linear algebra. There are many algorithms for solving linear systems, but this can only be done if the system has a solution. We can mention two cases for which there exist a solution of (2.10) [9]:

1 If the matrix A is nonsingular, then there exist a unique solution given by $x = A^{-1}b$.

2 If the matrix A is singular, then there exist solutions if $b \in Ran(A)$. If $Ax_0 = b$, then $A(x_0 + v) = b$ for all $v \in Ker(A)$. The nullspace $Ker(A)$ is at least one-dimensional when A is singular, this means (2.10) has an infinite amount of solutions.

For the sake of context we will mention some alternative discretization techniques that are commonly used. The finite element method (FEM) is commonly used in engineering applications with complex geometry. In this method, the physical domain is approximated by a tessellation consisting of a finite number of elements. The elements are often triangles in two dimensions and tetrahedrons in three dimensions. The method produces a linear system of equations for the unknowns in the vertices where the elements are tied together. An example of a finite element grid and the solution is plotted in figure 2.4. Figure 2.4b shows how the element method can be used to solve problems on geometries with curved edges, this would require a much more complex problem definition on the boundary if we wanted to use finite differences. Figure 2.4a shows the solution of the linear elasticity equation for a cubical region with tetrahedral elements. The solution is the displacement of the vertices, and the stresses indicated by the colour of the elements is calculated from the displacement.

The finite volume method, like the finite element method, has the advantage that it can solve problems with complex geometries. The finite volume method is



(a) Solution of the linear elasticity equation. (b) Two dimensional triangulation of a circle.

Figure 2.3: *Finite element* solution of the linear elasticity equation from continuum mechanics. The solution is calculated on a course tetrahedral grid, showing the displacement and stresses for a cube affected by gravity.

the preferred method for solving conservation laws [5], and the method is widely used in computational fluid dynamics.

Finally we mention the concept of spectral methods, where the solution of a PDE can be represented as a sum of special basis functions. These methods are not as suited for complex geometries like the FEM-method is, but for certain problems, spectral methods can exhibit excellent convergence rates.

2.6.2 Finite Difference Discretization

As mentioned, FD will be used in this thesis and there is many variations of finite difference approximations, but we will only use the central difference in this thesis. The finite difference method approximates the partial derivatives locally with a low order Taylor series expansions [5]. As an example we derive a very common approximation for the second order derivative, the central difference. We expand $u(x)$ with a Taylor series to the left and right of x.

$$
\begin{aligned}
u(x+h) &= u(x) + h\frac{\partial u(x)}{\partial x} + \frac{h^2}{2!}\frac{\partial^2 u(x)}{\partial x^2} + \frac{h^3}{3!}\frac{\partial^3 u(x)}{\partial x^3} + \frac{h^4}{4!}\frac{\partial^4 u(x)}{\partial x^4} + \dots \\
u(x-h) &= u(x) - h\frac{\partial u(x)}{\partial x} + \frac{h^2}{2!}\frac{\partial^2 u(x)}{\partial x^2} - \frac{h^3}{3!}\frac{\partial^3 u(x)}{\partial x^3} + \frac{h^4}{4!}\frac{\partial^4 u(x)}{\partial x^4} + \dots
\end{aligned}
$$

Where h is the step length between the grid points. When we sum together the two equations above, the terms with odd powers in h will cancel and we are left

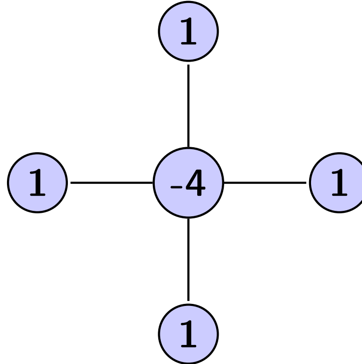


Figure 2.4: The *central difference* stencil for a two dimensional uniform grid.

with an approximation of the second order derivative.

$$u(x+h) - 2u(x) + u(x-h) = h^2 \frac{\partial^2 u}{\partial x^2} + \mathcal{O}(h^4) \tag{2.11}$$

The big O notation $\mathcal{O}(h4)$ simply means a function that decreases asymptotically as h^4 when $h \to 0$. We divide the equation by h^2 and rearrange to get the central difference formula:

$$\frac{\partial^2 u}{\partial x^2} = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + \mathcal{O}(h^2). \tag{2.12}$$

This approximation is second order accurate, seen by the term $\mathcal{O}(h^2)$. If we halve the step size h, the error should be reduced by a factor of four. Higher order schemes for the Laplacian may be used to improve accuracy.

The approximation uses information from neighbouring points straight to the left and right. We can use the central difference formula in multiple spatial directions, approximating functions in two and three dimensions as well. In figure 2.4 the two dimensional central difference stencil indicates how the approximation couples information from the neighbouring points together.

2.7 Solving Linear Sets of Equations

A big part of this thesis revolves around solving the the linear system of equations efficiently. Because of this we will spend some time introducing important theory on linear equation-solvers. We split the methods into two categories; *direct* methods and *iterative* methods. Direct methods will produce a solution after a

number of floating-point operations(FLOPs) that is known in advance[6]. Direct methods are robust and reliable, but this comes at the price of lower efficiency, and they are not so easily parallelized.

Iterative methods will iterate a procedure on an initial approximation until the satisfied accuracy of the solution is reached. Specialized effective iterative methods can be tailored for a given problem, but in general they are not as robust as direct methods.

When the number of unknowns in the coefficient matrix increases, the asymptotic runtime becomes increasingly important. For sparse systems, iterative methods can yield better runtime complexities and require less memory, in addition they are easy to parallelize - this is why iterative methods is preferred for problems with a large number of unknowns.

2.7.1 Direct Methods

The most commonly used direct algorithms are *Gauss elimination*, *LU-factorization* and *Cholesky's* method. Gauss elimination and LU-factorization will find the solution given nonsingular matrix A. Cholesky's method requires that the matrix is SPD in addition to non-singular. Cholesky's method is the fastest, but all of the three direct methods mentioned have a complexity of $\mathcal{O}(m^3)$ in general.

For the Poisson problem there exist methods with very good complexity in the order of $\mathcal{O}(m \log(m))$[9]. This can be done by utilizing an analytic expression for the eigenvalues of A and the discrete sine transform(DST) to calculate the solution. Fast direct Poisson solvers are possible because of the simple matrix structure, but this is a very special case. If we want effective solvers for more complex problems, iterative methods can be the best choice.

2.7.2 Iterative Methods

For large sparse linear systems, iterative methods can be very attractive due to the computational efficiency and low memory requirement, as well as being parallelizable. We cover the basic details of some iterative solvers, since this will be implemented in this thesis. We define the error with respect to equation (2.10) as

$$e_k = x - x_k \tag{2.13}$$

where x_k represents the approximation to the solution x after k iterations. In situations where we know x, we can measure the error e directly, and this is used to make sure our implementations behave as expected.

To measure the quality of the approximation when x is unknown, the *residual* is used:

$$r_k = b - Ax_k. \tag{2.14}$$

Many iterative methods such as relaxation use information from the residual since $r = 0$ imply that $x_k = x$. When $r \to 0$, usually $x_k \to x$. Finally we define the residual equation which is defined from the error and residual as

$$Ae = r. \tag{2.15}$$

The residual equation is used in the multigrid method[7] which we cover in section 2.7.2. We can distinguish between two classes of iterative methods; *relaxation*- and *projection* methods.

Relaxation Techniques

These methods modify the components of a given approximation x_k until convergence of (2.10) is reached[9]. Separately these techniques do not yield very good convergence, but great solvers can be built which make use of relaxation methods. The most famous relaxation methods are the *Jacobi* iteration and *Gauss-Seidel* iterations. To understand the Jacobi iteration, we will look at the components of an approximation x_k, denoted by $\xi_i^{(k)}$ for the ith component. We want to annihilate the components of the residual vector r_k in order to get the next approximation x_{k+1}. On component form we can write

$$(b - Ax_{k+1})_i = 0$$

which leads to the following iteration for the ith component

$$\xi_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^{m} a_{ij} \xi_j^{(k)} \right), \qquad i = 1, ..., m. \tag{2.16}$$

The Jacobi iteration can be written on vector form if we split the matrix A into two. We let D be the diagonal entries of A, and we define $\bar{A}$ as

$$\bar{A} = A - D. \tag{2.17}$$

which yields

$$x_{k+1} = D^{-1}(b - \bar{A}x_k). \tag{2.18}$$

Jacobi iterations can be implemented from both equation (2.16) and (2.18), but we will focus on the latter. Vector operations are implemented faster in MATLAB than point-wise iterations using for-loops.

Algorithm 1 Jacobi iteration on vector form

```
1: procedure Jacobi( A, b, x_0)
2:     for k = 0, 1, ..., until convergence do
3:         x_{k+1} := D^{-1}(b − Āx_k)
4:     end for
5: end procedure
```

The Jacobi iteration uses information from the kth iterate to build the next approximation x_{k+1}. The Gauss-Seidel iteration is similar to the Jacobi iteration, but utilizes the newly updated solution to get better convergence per iteration. On component form the iteration is described:

$$\xi_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} \xi_j^{(k+1)} - \sum_{j=i+1}^{m} a_{ij} \xi_j^{(k)} \right), \qquad i = 1, ..., m. \tag{2.19}$$

In algorithm 2 we can see that $\xi_i^{(k+1)}$ is updated and used to calculate the next components of x_k.

Algorithm 2 Gauss-Seidel iteration on component form

```
 1: procedure Gauss_Seidel( A, b, x_0)
 2:     for k = 0, 1, ..., until convergence do
 3:         for i = 1, ..., m do
 4:             σ := 0
 5:             for j = 1, ..., m do
 6:                 if i ≠ j then
 7:                     σ := σ + a_ij ξ_j
 8:                 end if
 9:             end for
10:             ξ_i := (b_i − σ)/a_ii
11:         end for
12:     end for
13: end procedure
```

The convergence rate of Jacobi and Gauss-Seidel is considerably reduced for inhomogeneous systems, but both methods can be improved by utilizing *Block Relaxation Schemes*, also called *line relaxation*. The solution vector x will often be a global representation of the unknowns in a grid, and the block relaxation or line relaxation scheme will update a sub-vector of the solution vector at a time. It is common to split the solution vector column-major or row-major for a two dimensional grid. When the solution vector has been split into sub vectors the coefficient matrix A will consist of blocks A_{ii} associated with each sub vector. The block Jacobi iteration is showed in pseudo code in algorithm 3. For each iteration with the block Jacobi we must do one matrix-vector product for the global vector x_k, and we solve a smaller set of equations for every block. In algorithm 3, W_i is a projection operator for the ith block and V_i represents an extension operator of the ith block into the global solution.

Algorithm 3 Block Jacobi iteration

```
1: procedure Block_Jacobi( A, b, x_0)
2:     for k = 0, 1, ..., until convergence do
3:         for i = 1, ..., number of blocks do
4:             Solve A_ii σ_i = W_i^T (b − A x_k)
5:             x_{x+1} := x_k + V_i σ_i
6:         end for
7:     end for
8: end procedure
```

With the block Jacobi iteration expressed, it is only a small change needed to get the block Gauss-Seidel iteration. The block Jacobi updates the new global vector for every block, but it does not use the updated solution to calculate the next block. In block Gauss-Seidel we update the solution and perform the matrix-vector product Ax for every block per iteration.

Relaxation methods are guaranteed to converge to a solution of (2.10), if the matrix is diagonally dominant, but we may get convergence without this being satisfied also.

We split projection methods into two classes; the *orthogonal*- and *oblique* projections. Two of the most common projection methods are the *conjugate gradient*(CG) method and the *generalized minimal residual*(GMRES) method. These two methods are projection processes onto *Krylov subspaces*, and represent respectively an orthogonal- and an oblique projection method. The Krylov subspace is defined as:

$$\mathcal{K}_k(A, r_0) = span\{r_0, Ar_0, A^2r_0, ..., A^{k-1}r_0\}. \tag{2.20}$$

GMRES is the most general method of the two, requiring that the matrix A is positive definite, where the CG in addition require that A is symmetric. We will present the conjugate gradient algorithm, since this is implemented in our multigrid algorithm for this project.

Conjugate Gradients

For the CG algorithm we search for the approximate solution $x_k \in x_0 + \mathcal{K}_k$ of problem (2.10) by imposing the Petrov-Galerkin condition[9]

$$b - Ax_k \perp \mathcal{K}_k. \tag{2.21}$$

We simply state the best known formulation of CG for the general case. In section 2.7.2 we take a look at a more specialized formulation of CG.

Algorithm 4 Conjugate Gradient

1: **procedure** **CG**(A, b, x_0)
2: **Calculate** $r_0 := b - Ax_0,\ p_0 := r_0$
3: **for** $j = 0, 1, ...,$ *until convergence* **do**
4: $\alpha := (r_j, r_j)/(Ap_j, p_j)$
5: $x_{j+1} := x_j + \alpha_j p_j$
6: $r_{j+1} := r_j - \alpha_j Ap_j$
7: $\beta_j := (r_{j+1}, r_{j+1})/(r_j, r_j)$
8: $p_{j+1} := r_{j+1} + \beta_j p_j$
9: **end for**
10: **end procedure**

We state the convergence result for the conjugate gradient, see [9] for a detailed derivation. Given the solution x and approximation x_k we get the following bound

$$||x - x_k||_A \leq 2 \left[\frac{\sqrt{\frac{\lambda_{max}}{\lambda_{min}}} - 1}{\sqrt{\frac{\lambda_{max}}{\lambda_{min}}} + 1} \right]^k ||x - x_0||_A, \tag{2.22}$$

where λ_{max} and λ_{min} represent the largest and smallest eigenvalue of A and the A-norm of x is defined as

$$||x||_A = (Ax, x)^{1/2}. \tag{2.23}$$

From (2.22), we see that the convergence of CG is dependent on $Eig(A)$. We know A is SPD, such that all eigenvalues are positive. The ratio $\frac{\lambda_{max}}{\lambda_{min}}$ is referred to as the condition number, and a smaller condition number means faster convergence with CG. As a standalone solver, the CG-algorithm is seldom used in the form stated in algorithm 4, but in combination with an effective preconditioner M, conjugate gradient can be a powerful solver. Suppose M is SPD and can be split into

$$M = M_L M_R. \tag{2.24}$$

Then we can change variables and solve the following problem:

$$M_L^{-1} A M_R^{-1} x = M_L^{-1} b, \qquad \tilde{x} \equiv M_R^{-1} x. \tag{2.25}$$

Preconditioning is about reducing the condition number to decrease the number of iterations needed to reach convergence. There is much more to preconditioning then what we have mentioned, but our intention is to note the idea.

The Multigrid Method

As previously mentioned one can utilize relaxation techniques to build faster methods. Multigrid methods are examples of specialized algorithms that can

achieve extraordinary runtime complexities. Parallel implementations of the multigrid method are often scalable, meaning that the speedup does not deteriorate fast when increasing the number of processors. The low runtime of multigrid methods comes at the cost of less generality. The methods mentioned above have in common that they solve general system of equations. Multigrid methods were initially designed to solve discretetized elliptic PDE's[9].

Multigrid methods employ a hierarchy of grids with varying resolution to get optimal convergence using relaxation techniques. Grid transfer operations are essential for multigrid methods since these methods rely on transferring information between different grids. We start by looking at the simple case with two grids where the finest Ω_h have step size h and the coarsest grid Ω_{2h} have double the step size of Ω_h. Moving from a coarse grid to a fine grid, is referred to as prolongation. Restriction is used when we go from a fine grid to a coarse grid. In the following we assume the grid is of size $m+1$ in every spatial direction, where m is even. More details can be found in [9]. When we prolong from a

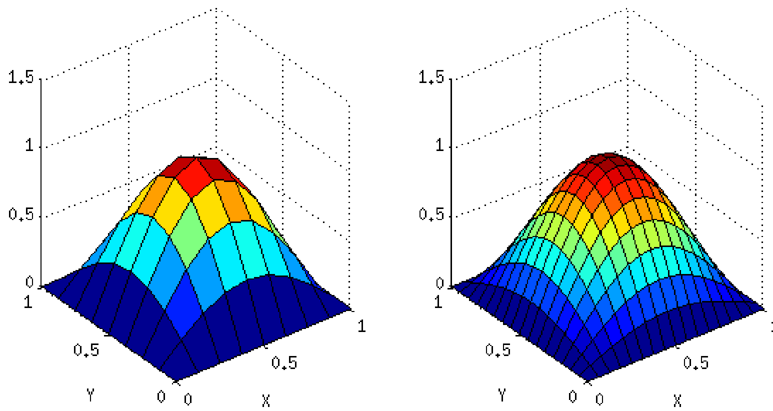


Figure 2.5: A grid transfer operation: interpolation of a surface from the coarse grid Ω_{2h} to the finer grid Ω_h.

coarse to a fine grid we use an interpolation operator. In figure 2.5 we illustrate the interpolation operator on a surface in MATLAB.

The Interpolation Operator

We denote interpolation as $I_{2h}^{h} : \Omega_{2h} \rightarrow \Omega_h$. The grid is doubled in size for each spatial direction and I_{2h}^{h} finds the new function values in the half steps of the

previous grid Ω_{2h} by linear interpolation. For a given vector $v^{2h} \in \Omega_{2h}$ we get $v^h = I_{2h}^h v^{2h}$ in Ω_h. For the one dimensional case I_{2h}^h is defined as:

$$\begin{cases} v_{2i}^h &= v_i^{2h} \\ v_{2i+1}^h &= (v_i^{2h} + v_{i+1}^{2h})/2 \end{cases} \quad , i = 1, ..., \frac{m}{2} + 1.$$

We move up to two-dimensions and the interpolation becomes

$$\begin{cases} v_{2i,2j}^h &= v_{ij}^{2h} \\ v_{2i+1,2j}^h &= (v_{ij}^{2h} + v_{i+1,j}^{2h})/2 \\ v_{2i,2j+1}^h &= (v_{ij}^{2h} + v_{i,j+1}^{2h})/2 \\ v_{2i+1,2j+1}^h &= (v_{ij}^{2h} + v_{i+1,j}^{2h} + v_{i,j+1}^{2h} + v_{i+1,j+1}^{2h})/4 \end{cases} \quad for \begin{cases} i &= 1, ..., \frac{m}{2} + 1 \\ j &= 1, ..., \frac{m}{2} + 1 \end{cases}$$

The two-dimensional case can be described as the tensor product between the one-dimensional interpolation in y- and x-direction:

$$I_{2h}^h = I_{y,2h}^h \otimes I_{x,2h}^h.$$

The stencil for the two dimensional case is

$$\frac{1}{4} \left] \begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix} \right[$$

The open brackets represent the fact the we prolong the previous grid by calculating the points in between.

The Restriction Operator

Defined by $I_h^{2h} : \Omega_h \rightarrow \Omega_{2h}$, the restriction operator have two common variations. Since the grid is reduced with half the step size in each direction, we can simply inject the values at every other grid point. The injection operator will simply remove every other line in each direction. A better restriction that require a bit more calculating is called full weighting(FW). In one spatial direction the FW is

$$v_i^{2h} = \frac{1}{4}(v_{2i-1}^h + v_{2i}^h + v_{2i+1}^h). \tag{2.26}$$

For a two dimensional problem the full weighting can be expressed as the tensor product of equation (2.26) in x- and y-direction, and the stencil is:

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

With the intergrid operations described we turn the next important concept of smoothing.

Smoothing

The relaxation methods mentioned above can be written in the following form:

$$x_{k+1} = Gx_k + f \tag{2.27}$$

where G is the iteration matrix for a given method. To fully solve a linear system of equations with a relaxation method will require many iterations, but there are components of the error that are damped quickly with few iterations. Residuals in the direction of the eigenvectors of G associated with the large eigenvalues are reduced quickly. The eigenvectors mentioned are called high frequency modes, while the smooth modes associated with the low frequency are slowly damped with a relaxation method. In the context of multigrid, relaxation methods are used to dampen the high frequency modes of the residual with a couple of iterations - this is referred to as smoothing. A key principle is that when we restrict the solution x to a coarser grid, many of the low frequency modes are mapped into high frequency modes, which can be effectively reduced by smoothing.

Multigrid V-Cycle

Knowing that we can smooth different components of the residuals on different grids, the multigrid V-cycle is quite intuitively defined. The residual equation (2.15) is used to smooth the error. The V-cycle starts with an initial guess x_0 in the finest grid Ω_h where it does ν_1 numbers of pre-smoothing iterations with a chosen relaxation method. The residual is calculated and restricted into a coarser grid Ω_{2h}, before it is used to smooth the error e^{2h}. This procedure is recursively done until we solve (2.15) exactly at the coarsest level. The method will update the solution by correcting for the error, do ν_2 numbers of post smoothing iterations, and interpolate back to the original grid. The process of the V-cycle is visualized in figure 2.6.

Algorithm 5 Multigrid V-cycle

```
1: procedure MGV( A_h, x_0^h, b^h)
2:     if (level== maxLevel) then
3:         Solve:                A_{2h} σ^{2h} = r^{2h}
4:     else
5:         Pre-smooth:           x^h := smooth^{ν1}(A_h, x_0^h, b^h)
6:         Calculate residual:   r^h = b^h − A_h x^h
7:         Coarsen               r^{2h} = I_h^{2h} r^h
8:         Recursion:            σ^{2h} = MGV(A_{2h}, 0, r^{2h})
9:         Correct:              x^h := x^h + I_{2h}^h σ^{2h}
10:        Post-smooth:          x^h := smooth^{ν2}(A_h, x^h, b^h)
11:    end if
12: end procedure
```

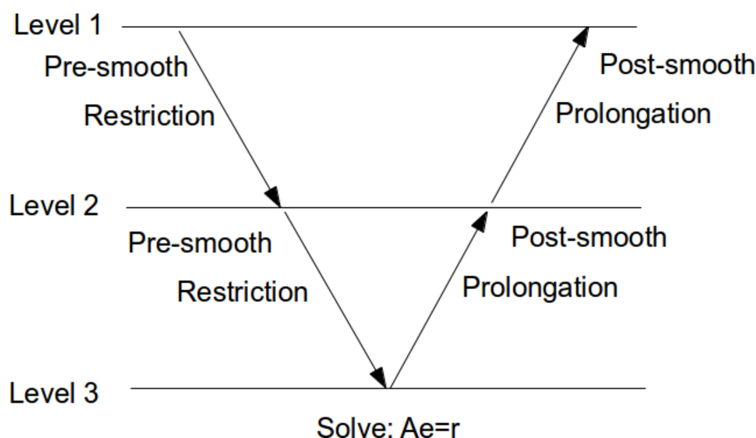


Figure 2.6: The multigrid V-cycle iteration. This shows three levels of the recursive algorithm.

When algorithm 5 is called, it will perform one V-cycle and return the numerical approximation of the solution. In algorithm 6, we put the V-cycle into an iteration which stops when the relative residual-norm reaches a wanted tolerance. In this way we can perform a number of iterations on the V-cycle to get a satisfying solution.

Algorithm 6 The Multigrid Method

1: **procedure** **MG_solver**(A, x_0^h, b^h)
2: $\quad ||r_0|| = ||b - Ax_0||$
3: $\quad x = x_0$
4: $\quad$ **for** $k = 1, ..., \infty$ **do**
5: $\quad\quad x = \mathbf{MGV}(A, x, b)$
6: $\quad\quad ||r_k|| = ||b - Ax||$
7: $\quad\quad$ **if** $||r_k||/||r_0|| < tol$ **then**
8: $\quad\quad\quad$ *stop*
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: **end procedure**

The V-cycle Complexity

For a general coefficient matrix $A_h \in \mathbb{R}^{m \times m}$ there are $c_s m_h$ non zero elements. The constant c_s depends i.e. on the type of discretization used. The V-cycle will smooth and restrict/prolong on each level. The cost of smoothing is $c_s m_h$, and

the cost of the intergrid operation is $c_g m_h$. The total cost at a level with m_h unknowns is given by

$$C(m_h) = (c_s(\nu_1 + \nu_2) + 2c_g)m_h + C(m_{2h}).$$

For a two dimensional problem, we halve the step size in each spatial dimension, yielding $m_{2h} = \frac{m_h}{4}$. As expected from the recursive nature of the algorithm, we get a recurrence relation for the cost:

$$C(m) = cm + C(m/4),$$

where $c = c_s(\nu_1 + \nu_2) + 2c_g$. We let the number of levels go to infinity to obtain an upper bound for the cost:

$$\begin{aligned} C(m) &\leq cm \sum_{k=0}^{\infty} \frac{1}{4^k} \\ &= \frac{4}{3}cm. \end{aligned} \tag{2.28}$$

From equation (2.28) we see that the cost per V-cycle iteration is linear, $\mathcal{O}(m)$. For our problem, the constant $c_s = 5$, since we couple the unknowns according to the stencil in figure 2.4. Using the full weighting operator and the interpolation as defined above will both yield $c_g = 9$. Based on this we can expect a cost per iteration of:

$$C(m) = \frac{4}{3}(5(\nu_1 + \nu_2) + 18)m, \tag{2.29}$$

in our multigrid implementation.

2.8 The Physical Analogy Method

We will now look into the details of the methods we are developing. Initially we make the analogy to a fluid flow problem. Imagine a box containing a porous media, a sink and a source. The source injects water, and the sink ejects water from the box. The source and sink is located in α and β respectively. The box is closed and we inject water in α at the same rate as we eject water in β.

If we could trace the water as it is pumped into the box, then we would see that the water will travel the fastest in the areas of least resistance. That is, the water will find the high permeability zones. This is the analogy we want to utilize, using the hazard data as a measure of the permeability. Setting a source in the starting point α, and a sink in the goal β, we can calculate different paths and then look for the best one with respect to the path finding problem.

We could also choose to use an analogy to heat diffusion since we will qualitatively get the same equations. As seen in (2.6) and (2.7), the only thing that separates the equations are the physical interpretation and constants. For our purpose it makes no difference which analogy we use, since equation (2.6) and (2.7) are essentially the same from a mathematical point of view.

From here on we will refer to the Physical Analogy Method as the PAM and mainly use the analogy to fluid flow.

2.8.1 The PAM Equation

The PAM will be based on the the same type of equation as in porous flow problems except that the physical constants will not be included. We will use a new function K which is based on the hazard H as an analogy to k or κ. The equation will look like this,

$$-\nabla \cdot (K \nabla u) = f \qquad in \qquad \Omega \tag{2.30}$$

where the solution u is the potential we can extract paths from, and K is the modified hazard or permeability. Equation (2.30) is referred to as the PAM equation.

2.8.2 Discretization

Equation (2.30) has only one second order derivative that must be discretized. In three dimensions fully written out, equation (2.30) looks like this:

$$-\left[\frac{\partial}{\partial x}\left(K\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(K\frac{\partial u}{\partial y}\right)\right] = f. \tag{2.31}$$

To discretize we use a second order centered difference formula in each spatial direction [9]. In the x-direction this becomes:

$$\begin{aligned} \frac{\partial}{\partial x}\left(K\frac{\partial u}{\partial x}\right) &= \frac{K_{i+1/2,j}(u_{i+1,j}-u_{i,j}) - K_{i-1/2,j}(u_{i,j}-u_{i-1,j})}{h_x^2} \\ &+ \mathcal{O}(h_x^2) \end{aligned} \tag{2.32}$$

where the indices represent the grid-points such that $u(x_i, y_j) = u_{i,j}$. The step size in the x-direction is denoted h_x and should not be mistaken as the derivative of h. We must use an averaging technique in the half step in K. Since this second order derivative tends to smooth things out, we use a simple average when evaluating half steps:

$$K_{i+1/2,j} = \frac{K_{i+1,j} + K_{i,j}}{2} \tag{2.33}$$

Applying (2.32) to all the terms we get the full set of discretized equations in two dimensions.

$$
\begin{aligned}
f_{i,j} = & - \frac{K_{i+1/2,j}(u_{i+1,j} - u_{i,j}) - K_{i-1/2,j}(u_{i,j} - u_{i-1,j})}{h_x^2} \\
& - \frac{K_{i,j+1/2}(u_{i,j+1} - u_{i,j}) - K_{i,j-1/2}(u_{i,j} - u_{i,j-1})}{h_y^2}
\end{aligned}
\tag{2.34}
$$

for all i and j in our grid. The resulting matrix A will be a sparse band diagonal matrix, with properties such as symmetry and positive definiteness depending on the boundary conditions.

2.8.3 Modified Hazard

In equation (2.6), $\frac{\rho k}{\mu}$ represents the ground and fluid properties of the domain Ω. In our model the hazard will be used as an analogy to the physical values.

The hazard has low values in the attractive areas, whereas $\frac{\rho k}{\mu}$ has high values in the areas the fluid will tend to go. To deal with this we must modify the hazard with the function $K = K(H)$, which is a transformation of the hazard taking high values of H to be low values in K. We will refer to K as the permeability in our problem.

A possible transformation of the hazard is

$$
K = C/H^p, \tag{2.35}
$$

where $C, p \in \mathbb{R}^+$ are constants, and $p \sim 1$. If we set $C = \max(H)$ and $p = 1$ then the resulting permeability will lie in the range $\left[1, \frac{\max(H)}{\min(H)}\right]$.

The hazard is not designed to be converted into permeability. To get useful results with the PAM, further processing of $K(H)$ must be expected. For fluid problems, the permeability must vary with some order to get significant effect on the solutions. We must make sure the $K(H)$ will stop the streamlines from going in areas with high hazard, while still keeping the system of equations as well posed as possible.

To limit the streamlines from entering hazardous areas, we must transform the permeability such that the difference in $K(H)$ between wanted- and unwanted areas is increased. One simple strategy is to set all permeabilities under a certain limit even lower. The level we cut off the permeability at, is referred to as the cut off-level COL. The permeabilities lower than COL is set to a new, lower level K_c.

$$
\begin{aligned}
\forall\ (x, y) \in \Omega :\ & K(x, y) < COL \\
set\ & K(x, y) \rightarrow K_c
\end{aligned}
$$

2.8.4 Extracting a Path from the Solution

To obtain a path we must process the solution of the PAM equation even further. We can calculate the gradient of u in every grid point to get the velocity vector $\boldsymbol{u}$ analogously to equation (2.5). With zero flux boundary conditions, all fluid generated in α should flow into β due to conservation of mass. This means every path from α should end up in β. We can find the streamlines in $\boldsymbol{u}$ and search for the one which minimizes S in equation (2.1).

If we seed a number of ν streamlines near the starting point α, these will follow the velocity field $\boldsymbol{u}$ through Ω and end up in β. The streamlines will be the potential candidates for the approximated optimal path P_{opt} that solves the path finding problem.

Given a vector field $\boldsymbol{u} \in \Omega$ we can use a variety of methods to find a streamline through Ω. With Euler's method we simply follow the vector field in small steps of length h starting in a seed point p_0. The path or streamline is found by iterating this formula:

$$P_{n+1} = P_n + h\boldsymbol{u}(P_n), \qquad P_0 = p_0. \tag{2.36}$$

The explicit Euler method is fast, simple and intuitive, but not the most accurate method. If accuracy is important we can i.e. use the fourth order *Runge-Kutta Method*[6], given by:

Algorithm 7 Explicit Runge-Kutta 4th order

1: **procedure ERK4**($\boldsymbol{u}, p_0, h,$ *maxIt*)
2: **for** $n = 0, 1, ...,$ *maxIt-1* **do**
3: $k_1 = h\boldsymbol{u}(P_n^{(x)}, P_n^{(y)})$
4: $k_2 = h\boldsymbol{u}(P_n^{(x)} + \frac{1}{2}h, P_n^{(y)} + \frac{1}{2}k_1)$
5: $k_3 = h\boldsymbol{u}(P_n^{(x)} + \frac{1}{2}h, P_n^{(y)} + \frac{1}{2}k_2)$
6: $k_4 = h\boldsymbol{u}(P_n^{(x)} + h, P_n^{(y)} + k_3)$
7: $(P_{n+1}^{(x)}, P_{n+1}^{(y)}) = (P_n^{(x)}, P_n^{(y)}) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$
8: **end for**
9: **end procedure**

In algorithm 7 we have explicitly written out the components, such that there is no confusion that $P_n = (P_n^{(x)}, P_n^{(y)})$. By solving the PAM equation and finding ν streamlines, we have reduced the total search space of possible solutions to a manageable number of possible paths. Given a sufficient formulation in the PAM, the paths we have found so far will already tend to go in high permeable zones; in other words: low hazard zones. To find the path that suits our problem statement and requirements mentioned in section 2.1, we can employ different strategies when selecting P_{opt} in $\mathbf{P} = \{P_1, P_2, \ldots, P_\nu\}$. We will now mention

some concepts for eliminating paths that can be used in combination to find P_{opt}.

Minimal Accumulated Hazard:

We can evaluate the cost grid along the candidate paths in **P** by interpolation, and then use this to find the path of least accumulated hazard. We denote the accumulated hazard over a path P by $S_N(P)$

$$S_N(P) = \sum_k H(P_k) \tag{2.37}$$

The path is represented by two vectors with x and y components and we quantify S_N by simply summing up the hazard along P. We can also put restrictions on the maximum hazard at any point along the path.

Shortest Path:

In addition to minimizing the hazard we can also base our selection on the total length of the path. It is easy to calculate and compare the different path lengths $|P_i|$, for $i = 1, 2, \dots, \nu$. If the amount of fuel is a bottleneck, it might be useful to choose among the shortest paths.

Energy Efficiency:

We can make a more advanced fuel consumption model if we evaluate the centripetal acceleration on the UAV, and the work performed against gravity. Eliminating the paths that erratically varies in height can help us find the most economical paths with respect to fuel. This concept might also be used to find paths that tend to go near the shoreline.

Manoeuvrability:

If the paths calculated contains jagged edges, post processing might be the best solution to ensure the UAV can follow the path. The problem statement does however tend to generate smooth paths when there is not extreme variations in the permeability.

2.8.5 Boundary Conditions

Zero flux boundaries means that $\frac{\partial u}{\partial \boldsymbol{n}} = 0$ on $\partial\Omega$, this is also called homogeneous Neumann boundary conditions. The streamlines of u, which are the paths we seek, will be forced to go parallel to $\partial\Omega$ when they are sufficiently close to it.

The full path finding PDE when we use homogeneous Neumann conditions is then given by:

$$
\begin{aligned}
-\nabla \cdot (K \nabla u) &= f \qquad in \qquad \Omega \\
\frac{\partial u}{\partial \boldsymbol{n}} &= 0 \qquad on \qquad \partial\Omega.
\end{aligned} \tag{2.38}
$$

If we discretize equation (2.38), the coefficient matrix A will be singular. The resulting solution u from such a problem will only be defined up to a constant, and $u + D$ will also be a solution, for some constant D. This can be seen by the fact that $\frac{\partial(u+D)}{\partial \boldsymbol{n}} = \frac{\partial u}{\partial \boldsymbol{n}}$ in the Neumann condition. For the strategy mentioned above, we can still use homogeneous Neumann condition because the gradient of u is used to derive the paths. Homogeneous Neumann conditions on the whole boundary will be practical in order to evaluate paths from α to β, but this will not result in nice properties for the linear system of equations.

When we set a the solution directly to a certain value, we refer to this as Dirichlet conditions. For many modelling cases, the solution is known at the boundary, then the Dirichlet boundary conditions look like this:

$$
u = g \qquad on \qquad \partial\Omega,
$$

where g is a function describing the values along $\partial\Omega$. If we utilize Dirichlet conditions along the boundary $\partial\Omega$ the streamlines of the solution u can go in or out of Ω, this means that not every streamline will end up in B. This can restrict the method, but it can also yield matrix properties such as symmetry and positive definiteness.

2.8.6 Source Term vs. Boundary Condition

Point α and β can be implemented through the boundary conditions or with the use of a source function, f. We can choose to omit f from the equation and simply use Dirichlet boundary conditions on the points α and β. This means we set the solution to certain values $u(\alpha)$ and $u(\beta)$.

Qualitatively we do not expect the streamlines to be different whether we use a source function or Dirichlet boundary, but this can have an effect on the linear system of equations when we discretize. To implement two Dirichlet nodes inside Ω we have to change this in the matrix A, but if we use the source function, A remains the same.

Matrix properties such as symmetry, positive definiteness and diagonally dominance will greatly affect which methods that can be used to solve the linear system. Boundary conditions will often affect these matrix properties.

2.8.7 Exclusion Zones

The exclusion zones can be treated by simply increasing the hazard values for every grid point inside the exclusion zone. If the permeability in the exclusion zones are sufficiently low, the paths will be forced to avoid these zones. In figure 2.7 we have illustrated the terrain with two exclusion zones as examples.

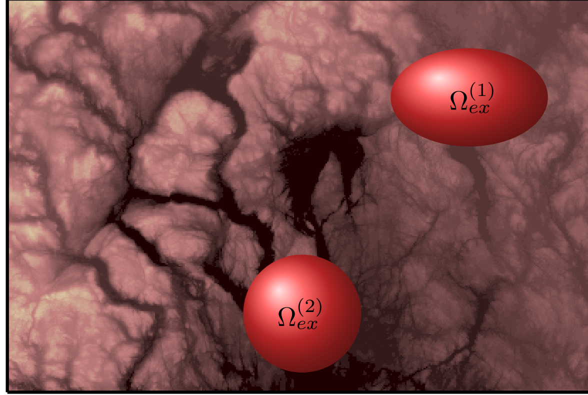


Figure 2.7: The domain Ω seen from above with two exclusion zones Ω_{ex}^1 and Ω_{ex}^2 in the interior.

2.8.8 Optimality of the Solution

As was mentioned in the introduction, we do not know if the minimization formulation (2.2) can be represented by the PAM equation or a partial differential equation in general. The analogy to fluid flow and physics is merely a means to see if we can solve the path finding problem to a satisfactory extent. We consider some properties of the streamlines to see how the PAM set restrictions on the solution.

The paths obtained with the PAM are all based on the streamlines of ∇u. The streamlines are tangential to ∇u at every point in Ω and can only have one value, implying that none of the streamlines are intersecting. The fact that no streamlines intersect can be restricting for the purpose of finding the optimal path.

Suppose we have a streamline that corresponds to the optimal path through a part of the domain, then there is no guarantee that this streamline will go through the optimal remaining part of the domain. Since no streamlines intersect, a streamline that is optimal in a sub route of α to β, can be forced to go through a low permeable zone on the remaining route to β. Based on this we have a

concern that the PAM will find less optimal solutions when the distance between α to β increases.

To test if the method finds the optimal solution we can inspect sub routes of the solution P_{opt}. The method finds a path $P_{opt} = P_1 + P_2$ from α to β. If the PAM really finds the optimal solution, then the sub routes should also be optimal. We can check if the method calculates the same path when we only test P_1 or P_2.

In addition, it is easy to check if the pathfinder is conservative, in the sense that it will find the same path from α to β as from β to α. If $P_{\alpha\rightarrow\beta} \neq P_{\alpha\rightarrow\beta}$, the pathfinder clearly does not find an optimal solution for this problem.

2.8.9 Computational Domain

The computational grid is represented as a regular grid, uniform in the xy-plane and of varying step size in z-direction. By regular we mean that the grid is composed of rectangular blocks as seen in figure 2.1. Our plan is to calculate the optimal path on a regular grid, then add the elevation to the z-component of P_{opt}. This approach may be erroneous, as we will demonstrate below, but as long as the error do not compromise the entire method we will gain simplicity from using a regular grid. The alternative would be to use FEM on an unstructured grid which would represent the physical domain, but this will complicate the task of building a fast solver.

Depending on on how strongly the topography varies we will introduce an error when we build the final solution by adding the z-coordinate to P_{opt}. To address the error we will consider the weak formulation of the Poisson problem:

$$\begin{aligned} -\nabla^2 u &= f \qquad in \quad \Omega \\ u &= 0 \qquad on \quad \partial\Omega. \end{aligned} \tag{2.39}$$

We will use (2.39) as an example to evaluate the error we introduce by not using the exact physical domain in the calculations.

The weak form is derived by first multiplying equation (2.39) with an arbitrary *test function* v. Then we utilize *Green's formula* for the Laplacian[8]:

$$-\int_{\Omega} \nabla^2 u v \mathrm{d}\Omega = \int_{\Omega} \nabla u \cdot \nabla v \mathrm{d}\Omega - \int_{\partial\Omega} \frac{\partial u}{\partial \boldsymbol{n}} v \mathrm{d}\gamma, \tag{2.40}$$

The weak form of problem (2.39) is equivalent to finding a solution $u \in H_0^1(\Omega)$ such that

$$\int_{\Omega} \nabla u \cdot \nabla v \mathrm{d}\Omega = \int_{\Omega} f v \mathrm{d}\Omega \qquad \forall v \in H_0^1(\Omega). \tag{2.41}$$

The function space H_0^1 is a *Sobolev* space defined as[8]:

$$\begin{aligned} H_0^1(\Omega) &= \{v \in H^1(\Omega) : v = 0 \quad on \quad \partial\Omega\} \\ H^1(\Omega) &= \{v : \Omega \to \mathbb{R} \;\; s.t \;\; v \in L^2(\Omega), \; \frac{\partial v}{\partial x}, \frac{\partial v}{\partial z} \in L^2(\Omega)\} \\ L^2(\Omega) &= \{v : \Omega \to \mathbb{R} \;\; s.t \int\limits_{\Omega} v^2 \mathrm{d}\Omega < +\infty\}. \end{aligned}$$

We can transform this problem and do the calculations in a reference domain, $\hat{\Omega}$. See figure 2.8. To represent a function in the reference domain we use the following notation; $u(x,z) = u(x(\xi,\eta), z(\xi,\eta)) = \hat{u}(\xi,\eta)$, where (ξ,η) is a point in $\hat{\Omega}$. With this notation we can express the gradient operator in terms of reference domain variables

$$\nabla u = \begin{pmatrix} \xi_x & \eta_x \\ \xi_z & \eta_z \end{pmatrix} \hat{\nabla}\hat{u} \tag{2.42}$$

where

$$\hat{\nabla}\hat{u} = \begin{pmatrix} \hat{u_\xi} \\ \hat{u_\eta} \end{pmatrix}. \tag{2.43}$$

We use (2.42) to express $\nabla u \cdot \nabla v$ in terms of ξ and η:

$$\nabla u \cdot \nabla v = (\hat{\nabla}\hat{v})^T \begin{pmatrix} \xi_x & \xi_z \\ \eta_x & \eta_z \end{pmatrix} \begin{pmatrix} \xi_x & \eta_x \\ \xi_z & \eta_z \end{pmatrix} \hat{\nabla}\hat{u}. \tag{2.44}$$

We define $\mathcal{J}$, the Jacobian of the transformation.

$$\mathcal{J} = x_\xi z_\eta - x_\eta z_\xi \tag{2.45}$$

With the Jacobi transformation the matrix-matrix product in equation (2.44) can be written as

$$\boldsymbol{G} = \frac{1}{\mathcal{J}} \begin{pmatrix} (x_\xi^2 + z_\eta^2) & -(x_\xi x_\eta + z_\xi z_\eta) \\ -(x_\xi x_\eta + z_\xi z_\eta) & (x_\xi^2 + z_\xi^2) \end{pmatrix}. \tag{2.46}$$

The reformulation yields; find $u \in H_0^1(\Omega)$ such that:

$$\int\limits_{\hat{\Omega}} (\hat{\nabla}\hat{u})^T \boldsymbol{G} \hat{\nabla}\hat{v} \mathrm{d}\hat{\Omega} = \int\limits_{\hat{\Omega}} \hat{f}\hat{v}\mathcal{J}\mathrm{d}\hat{\Omega} \qquad \forall v \in H_0^1(\Omega) \tag{2.47}$$

The mapping in figure 2.8 is of the form

$$\mathcal{F} : \begin{array}{l} x(\xi) = \frac{L_x}{2}(\xi + 1) \\ z(\xi,\eta) = \frac{L_z}{2}(\eta + 1) + g(x(\xi)) \end{array} \tag{2.48}$$

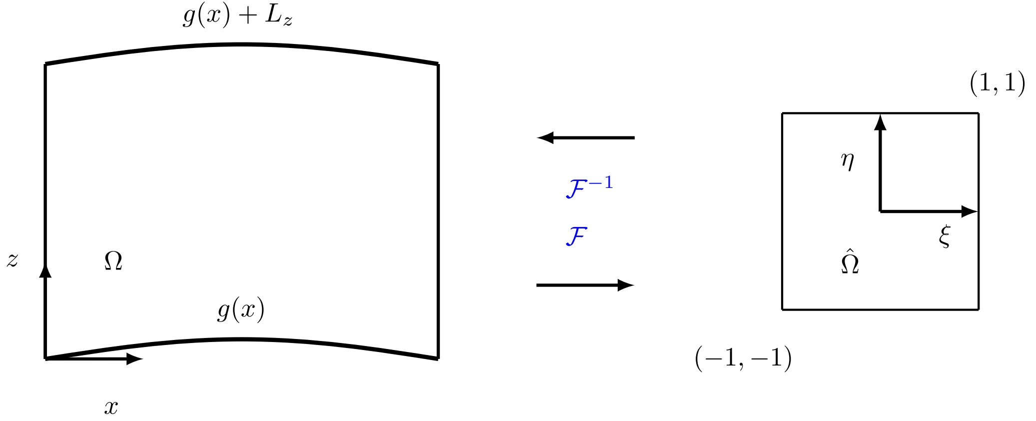


Figure 2.8: The mapping between physical domain Ω and the reference domain $\hat{\Omega}$.

where $g \in C^1(\Omega)$ is a function representing the geographical profile. So by inserting (2.48) into $\mathcal{J}$ and $\boldsymbol{G}$ we can rewrite our integrand. After some calculations we get

$$(\hat{\nabla}\hat{u})^T \boldsymbol{G} \hat{\nabla}\hat{v} = \frac{L_z}{L_x}\hat{u}_\xi \hat{v}_\xi - g'(x(\xi))(\hat{u}_\eta \hat{v}_\xi + \hat{u}_\xi \hat{v}_\eta) + \frac{L_x}{L_z}(1 + g'(x(\xi))^2)\hat{u}_\eta \hat{v}_\eta \quad (2.49)$$

thus indicating that g gives a contribution to the integral. The size of the contribution depends on the function g and the scales of the domain, L_x and L_z.

In comparison, we can consider the case where $g(x) = 0$, representing the domain if the ground was completely flat. Then the integrand would be

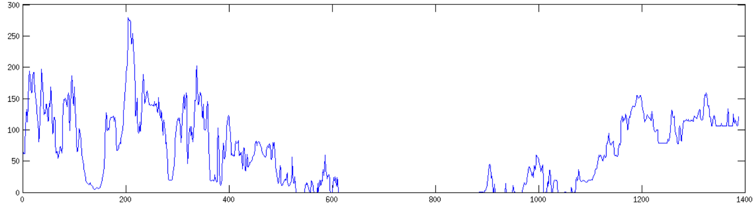
$$(\hat{\nabla}\hat{u})^T \boldsymbol{G_{g=0}} \hat{\nabla}\hat{v} = \frac{L_z}{L_x}\hat{u}_\xi \hat{v}_\xi + \frac{L_x}{L_z}\hat{u}_\eta \hat{v}_\eta. \quad (2.50)$$

We compare the difference between the two integrands by defining

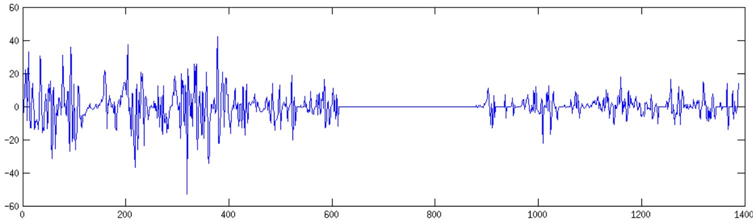
$$\begin{aligned} \Delta &= (\hat{\nabla}\hat{u})^T \boldsymbol{G} \hat{\nabla}\hat{v} - (\hat{\nabla}\hat{u})^T \boldsymbol{G_{g=0}} \hat{\nabla}\hat{v} \\ \Delta &= -g'(x(\xi))(\hat{u}_\eta \hat{v}_\xi + \hat{u}_\xi \hat{v}_\eta) + \frac{L_x}{L_z} g'(x(\xi))^2 \hat{u}_\eta \hat{v}_\eta. \end{aligned} \quad (2.51)$$

Based on equation (2.51) it is reasonable to expect that $\Delta \to 0$, as $g'(x(\xi)) \to 0$. This will certainly be the case when $g \equiv 0$, then $g' \equiv 0$ and the methods are the same. In those cases where g varies throughout Ω we can expect Δ to be small in the areas where the g' is small, and in general $g' \gg 1$ will yield $\Delta > 1$.

In figure 2.9, we have plotted the section of our given example elevation that yield the least variance in z with respect to x. This is an example of how g would look like in the mapping (2.48) and error (2.51). The plot in 2.9b show the g' will have large variations contributing to the error Δ.



(a) Plot of the elevation, $g(x)$ for a constant y.



(b) The derivative: $g'(x)$. In this case $||g'||_\infty = 53$.

Figure 2.9: The elevation data for a constant y in the test data. This is the row with the least variance. This can be considered as $g(x)$ for the error discussion.

2.9 Graph Algorithm

As mentioned in the introduction, there exist various shortest path algorithms for weighted graphs. We can find a path that minimizes the hazard between α and β applying a graph algorithm such as Dijkstra on the regular cost grid H. We expect Dijkstra's algorithm to find

$$P^\star = \min_{P \in \Omega} \int_\alpha^\beta H(P) \mathrm{d}s \tag{2.52}$$

which is a global optimal solution. Dijkstra's algorithm is a serial algorithm by nature, and we require many unknowns to get a sufficiently smooth path using a regular grid for a graph, considering this we expect a poor performance on a multi-core processor, but this will serve as a reference for the PAM.

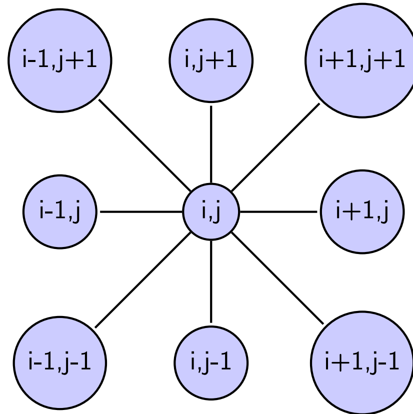


Figure 2.10: Similar to a nine point stencil for a two dimensional uniform grid, we couple each node to eight other nodes in this Dijkstra implementation.

2.9.1 Dijkstra on Cost Grid

We are solving the shortest path problem with Dijkstra on the regular cost grid H. The cost grid will now represent the set of vertices or nodes V. Even though we are using Dijkstra we will not go into algorithmic details. We will simply describe the context around the algorithm and how we incorporate it to the path finding.

The graph $G(V, E)$ is built by connecting each node to 8 neighbours, similar to what is known as the nine-point stencil, shown in figure 2.10. With m grid points in the two dimensional cost grid, the size of V is $|V| = m$ nodes. Connecting the graph using the rule in figure 2.10 we will get roughly $|E| = 4m$ edges, not considering the boundary effects. The edge weights is given by the associated weight function w which simply is the average value of the hazard for the connected vertices. For the edge E_1 consisting of vertices v_1 and v_2, the weight function will yield $w(E_1) = 1/2(H(v_1) + H(v_2))$. After we assemble the graph G and calculate all the weights, we can find the shortest path from a start point α to the target β using Dijkstra's algorithm.

There are different ways to implement Dijkstra and some methods are more efficient than others. Simple implementations have a runtime complexity of $\mathcal{O}(|V|^2)$ while more efficient versions are $\mathcal{O}(|V| \log |V|)$.

2.9.2 Preprocessed Graph

An alternative approach to the path finding problem can be found by combining a graph algorithm with the PAM. If we can reduce the cost grid to a reasonably small graph with nodes placed on the right places, then the PAM can be used to find the edges with corresponding weights and i.e. Dijkstra to find the optimal combination of sub paths. In other words we create a hybrid algorithm where the PAM is used to pre-process the cost grid into a graph G where the weights and edges are stored. Then a graph algorithm can solve the problem efficiently because of the reduced size of G compared to the full grid.

There is a challenge with the incoming and outgoing flight directions from a node with this approach. With different paths connected in a node, the combination of paths for a total solution might not be sufficiently smooth in the nodes. In [1] the author uses a similar approach to solve a pathfinding-problem. A graph is preprocessed by a random process, placing nodes in an unstructured fashion throughout the domain, then feasible paths are constructed with polynomials satisfying the path requirements. Since the superposition of edges into a path will satisfy every requirement, the UAV can simply solve a graph optimization problem.

A hybrid approach of this mentioned form, seems as a viable approach for solving pathfinding problems of this nature, but in our case it is difficult to make the PAM satisfy flight directions toward a target location.

2.10 Post-Processing

After we have calculated a two dimensional path $P^{(x,y)}$ with the PAM or Dijkstra, we need to find the horizontal component $P^{(z)}$ of the total path P. We interpolate the horizontal components $P^{(x,y)}$ onto the terrain and get $P_T^{(z)}$, which is the same as the projection of P onto the terrain. A simple way to calculate $P^{(z)}$ is adding a constant height to $P_T^{(z)}$ and remove the oscillations with a simple moving average operator. This is done as an example in figure 2.11 where $P_T^{(z)}$ is plotted in the blue line. The green line $P^{(z)}$ is shown lying above the terrain.

2.11 Code Environment

The path finding algorithm is meant to run on the UAV during flight, continuously calculating a path. New threats can appear and the UAV must be able to adjust to a dynamic scenery. This has implications for the path finding algorithm. We assume the UAV has a multi-core processor to utilize parallelization, but with limited available computational power. Due to the limited available resources,

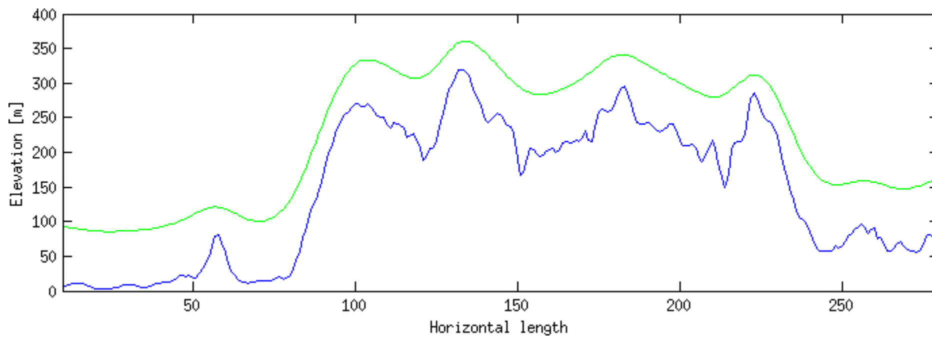


Figure 2.11: The vertical data for a path P. The green path $P^{(z)}$ is calculated by averaging away the local variations of $P_T^{(z)}$ and translating the whole path up from the terrain.

the UAV needs a cost efficient algorithm, preferably running close to real-time. The data-handling and memory usage can also be a critical factor for the total performance, and this should be addressed in an efficient implementation.

Chapter 3

Implementation

In this chapter we will go through some details of our implementation. We will present the PAM algorithmically as well as some details in the implementation of algorithms mentioned in section 2.

We have presented the theory behind the PAM, and in algorithm 8 we state the steps of the method symbolically. When we solve the PAM equation (2.38) in step 3 of algorithm 8, we do not actually form the inverse coefficient matrix A^{-1}, this is solved with MATLAB's built in function or our multigrid implementation. The multigrid implementation is presented in section 3.2. Step 4 and 5 is calculated easily with the built in functions for gradient and streamlines, but for function *evaluatePaths* we will present some more details. We start by presenting some details concerning the calculation of u, with both the direct solver in MATLAB and our multigrid implementation.

Algorithm 8 The PAM

1: **procedure PAM**(H, α, β)
2: **Process Permeability:** $K := K(H)$
3: **Solve PAM equation:** $u = A^{-1}b$
4: **Calculate gradient:** $\boldsymbol{u} = \nabla u$
5: **Calculate streamlines:** $[P_1, ..., P_\nu] = streamline(A)$
6: **Evaluate streamlines:** $P_{opt} = evaluatePaths(P_1, ..., P_\nu)$
7: **end procedure**

When we solve the linear set of equations with the built in solver in MATLAB we need to construct the matrix A explicitly, since this is input along with the right hand side vector b. The matrix is sparse, consisting of five bands, which is roughly $5m$ floating point numbers for a problem with m unknowns. Memory usage is also a problem when solving big problems, and in the multigrid

implementation we get away with not forming the matrix A at all.

3.1 Linear Solver in MATLAB

The two dimensional grid is of size $(N+1)\times(N+1)$ including the boundary points. This configuration of the grid is used to make the multigrid implementation as simple as possible. The boundary is excluded and we get a problem of size $m = (N-1)^2$. When solving $Au = b$ with a direct solver, the solution u and right hand side b must both be vectors. We use a global representation for the unknowns; that u and b is stored row after row, in a global vector of length $(N-1)^2$. Once b and the matrix A is assembled we can solve the system and reshape the vector into a local representation:

```
%% Solve AU=b
U = A\b;
U=reshape(U,N-1,N-1);
```

The MATLAB solver is called **mldivide**, and it uses an appropriate direct method depending on the properties of the matrix A.

3.2 The Multigrid Method

As mentioned we do not form the matrix in our multigrid implementation. We use the multigrid V-cycle presented in algorithm 5 to solve $Au = b$, but in the places where we need to apply the matrix-vector product we use the stencil the matrix A represents. The simple structure that comes from the central difference discretization makes it possible to perform the operation directly on the grid without forming A. We remember from the V-cycle, discussed in algorithm 5, that A is involved when we pre- or post-smooth and when we solve the equations for the coarsest level, but we will avoid storing the matrix A in our implementation. Since the matrix-free operations is an essential part of our code, we will look at the specific implementations of relaxation and the CG iterations.

3.2.1 Matrix-Free Relaxation

We utilize a local representation in the multigrid implementation, which means the unknowns is stored as a matrix or array of numbers. The implementation is based on grids with size $(N+1)\times(N+1)$ where $N = 2^d$ and $d \in \mathbb{N}$. This makes restriction operations simple since we can halve N, d times.

When we do an iteration with GS and Jacobi, each unknown will use four adjacent neighbours in a pattern to update the solution. In the following MAT-

LAB code-snippet, we show one iteration with the Jacobi method, in a fashion that does not require a stored matrix, A.

```
%% Jacobi iteration
index = 2:N;
U(index,index) = (   U(index-1,index) + U(index+1,index) + ...
                     U(index,index-1) + U(index,index+1) + ...
                     h^2*b(index,index)   )/4
```

Remember from equation equation (2.17) that the Jacobi method performs one matrix-vector operation per iteration. In our implementation, this is performed with vectors instead of for-loops. There is also a convolution-function in MATLAB that can apply a stencil on a matrix U really fast, and with this, the Jacobi iteration becomes:

```
%% Jacobi iteration with convn()
% The stencil
Jacobi = [   0 -1   0;
            -1  0  -1;
             0 -1   0  ];
U=h*h*b-convn(U,Jacobi,'same'))/4
```

Gauss-Seidel is similar to Jacobi, but we loop through the unknowns with a double for-loop, and this will update the solution continuously during the iteration. The following code-snippet illustrate how GS is applied without ever forming the matrix:

```
%% Gauss-Seidel iteration
for j = 2:N
    for i = 2:N
        U(i,j) = (U(i-1,j)+U(i+1,j)+U(i,j-1)+U(i,j+1)+h^2*b(i,j))/4;
    end
end
```

For the case where we have variable material data, i.e. the cost grid, we have to include some more terms when we apply A. The Gauss-Seidel with the permeability becomes:

```
%% Gauss_Seidel iteration on cost grid
for j = 2:N
    for i = 2:N
        U(i,j)=(((K(i-1,j)+K(i,j))*U(i-1,j)+...
                 (K(i+1,j)+K(i,j))*U(i+1,j)+...
                 (K(i,j-1)+K(i,j))*U(i,j-1)+...
                 (K(i,j+1)+K(i,j))*U(i,j+1))/2+...
                  h^2*b(i,j))/((K(i+1,j)+K(i-1,j)+K(i,j-1)+K(i,j+1)+...
                  4*K(i,j))/2);
    end
end
```

Similar to GS, but must faster per iteration is the Jacobi iteration, implemented with vector operations:

```
%% Jacobi on cost grid
U(2:N,2:N) = (((    K(1:N-1,2:N)+K(2:N,2:N)).*U(1:N-1,2:N) + ...
              (     K(3:N+1,2:N)+K(2:N,2:N)).*U(3:N+1,2:N) + ...
              (     K(2:N,1:N-1)+K(2:N,2:N)).*U(2:N,1:N-1) + ...
              (     K(2:N,3:N+1)+K(2:N,2:N)).*U(2:N,3:N+1))./2+...
                    h^2.*b(2:N,2:N))./((K(3:N+1,2:N) + K(1:N-1,2:N)+...
                    K(2:N,1:N-1) + K(2:N,3:N+1) + 4*K(2:N,2:N))./2);
```

We emphasize that these code snippets are specific implementations for the matrix-vector products arising from the discretization in equation (2.34). We also show how the matrix is applied for the CG iterations.

3.2.2 Conjugate Gradient

In our CG implementation we apply A on the current search direction vector p_j, one time per iteration, as seen in algorithm 4. With vector operations the matrix-vector operation is performed as:

```
%% Matrix vector product A*p performed in CG.
Ap(2:N,2:N)=(-   (K(1:N-1,2:N)+K(2:N,2:N)).*p(1:N-1,2:N) - ...
                 (K(3:N+1,2:N   )+K(2:N,2:N)).*p(3:N+1,2:N) - ...
                 (K(2:N,   1:N-1)+K(2:N,2:N)).*p(2:N,1:N-1) - ...
                 (K(2:N,   3:N+1)+K(2:N,2:N)).*p(2:N,3:N+1) + ...
                 (K(3:N+1,2:N   )+K(1:N-1,2:N)+K(2:N,1:N-1) + ...
                  K(2:N,3:N+1) + 4*K(2:N,2:N)).*p(2:N,2:N))/(2*h^2);
```

In this algorithm p_j is stored in a local representation when we apply A, but we reshape p_j to a global vector to perform the rest of the iteration.

3.3 Path Evaluation

The calculation of the gradient, streamlines and the evaluation of the streamlines is done in one function in our implementation. In algorithm 9 we take the solution u and the two locations α and β as input, then we seed the ν points in a circle around α, and we calculate the gradient vector field ∇u. We set the minimum hazard along a path, to infinity, and then we iterate over all streamlines/paths and save the one yielding the least accumulated hazard. For every seed point s_i we find the streamline corresponding to a path P. We check if P terminates at the target location β and then sum up the hazard $S_N(P)$. If $S_N(P)$ is smaller than the previous iterate we store the path as the best path P_{opt}. Similar to this approach, we can also get the lowest path through the domain, or the shortest.

Algorithm 9 Path evaluation

1: **procedure evaluatePaths**(u, α, β)
2: $[s_1, ..., s_\nu] = seedCircle(\alpha)$
3: $[Gx, Gy] = gradient(u)$
4: $S_{min} := \infty$
5: **for** $i = 1, ..., \nu$ **do**
6: $P = streamline(Gx, Gy, s_i)$
7: **if** $P \rightarrow b$ **then**
8: $S_N = \sum_k H(P_k)$
9: **if** $S_N < S_{min}$ **then**
10: $S_{min} := S_N$
11: $P_{opt} := P$
12: **end if**
13: **end if**
14: **end for**
15: **end procedure**

3.4 Dijkstra on Regular Grid

We store all the nodes in our grid in a global vector $v \in \mathbb{R}^m$, similar to what we have done for the linear solver in MATLAB. With m nodes in the grid, the graph is represented with an adjacency matrix $A_G \in \mathbb{R}^{m \times m}$ where $A_G(i, j)$ represent the weight between node i and node j. The edge weight is calculated as a simple average, as discussed in section 2.9.1, and we compensate for the diagonal edges by scaling with $\sqrt{2}$. The cost of going from node i to j is the same as the cost of going from node j to i, thus $A_G = A_G^T$. The adjacency matrix will be a sparse banded matrix consisting of 9 bands, based on the stencil in figure 2.10.

We use two MATLAB source codes; **dijkstra.m** and **pred2path.m**, to find the optimal path between point α and β. The Dijkstra-code uses **pred2path.m** internally, to return the path represented as a vector of consecutive nodes. For more information and the complete source, see [5]. After assembling A_G we simply use **dijkstra.m** as a black box solver to get P_{opt}:

```
%% Dijkstra
[D,P] = dijkstra(Ag, alpha , beta);
```

In this code-snippet, P is the path from α to β and D is the summed hazard $S_N(P)$.

Chapter 4

Results

In this chapter we present results from the method we are developing. Initially, we present the simple test cases, then we advance and build on top of these. In a more effective implementation, we test the multigrid V-cycle as a solver for the linear system of equations. All of the results presented are based on the two dimensional equations. Every result is run with MATLAB R2013a on a Lenovo Thinkpad with 2.60GHz Intel Core i5 CPU and 5.6 GB of RAM.

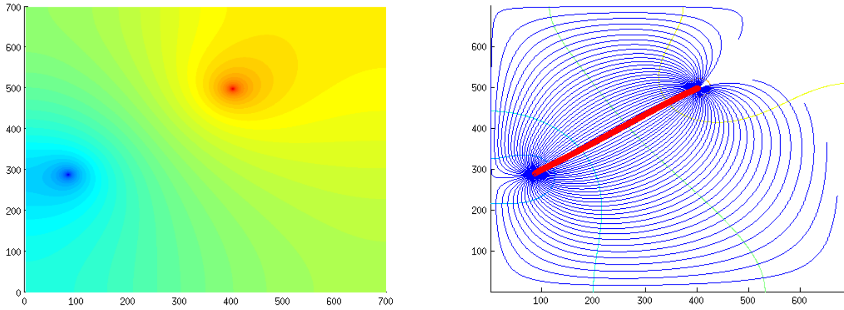
4.1 The PAM with Homogeneous Neumann BC

In section 2.8.5 we covered the Neumann boundary conditions. The results presented in this section are calculated from the PAM-equation described in 2.38. The matrix A is singular, but this does not hinder us from finding the streamlines based on ∇u.

4.1.1 Isotropic Permeability

To begin with, we consider the case where $K = 1$ for all $(x, y) \in \Omega$, then we are left with the Poisson equation as in (2.3). The reason for doing this is to test the PAM for the very simplest case. Intuitively we know that the optimal path from α to β when the risk is constant is the euclidean distance between α and β, a straight line.

The results from the PAM when we have uniform hazard are shown in figure 4.1, the scalar solution is on the left with the paths derived from it to the right. The red path indicated in figure 4.1b shows the shortest of all the evaluated paths. We notice that the homogeneous neumann boundary conditions makes the paths go parallel near the boundary. Figure 4.1 indicates that the idea of the



(a) The solution of Poissons equation, u.

(b) Streamlines of u between α and β.

Figure 4.1: Results from the PAM with uniform cost grid in Ω. The red path shows the shortest path between point α and β. The streamlines representing the possible paths are calculated from ∇u.

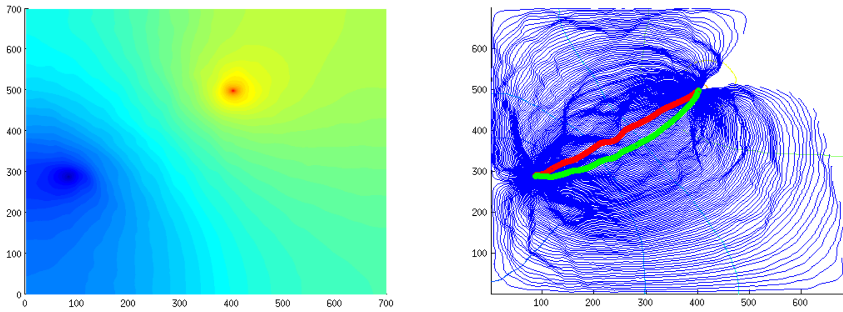
PAM is working in a simple 2D case with constant hazard.

4.1.2 Anisotropic Permeability

The next step is to implement the case when K varies throughout the domain. We use the set of discretized equations in (2.34). From the cost grid around Oslo we define the permeability $K(H)$. After we scale the permeability with the maximal value, K lies within $[1, 42.7]$. Figure 4.2 illustrates the effect a varying K has on the streamlines. The paths tend towards areas where the hazard is low, but only locally. When we compare with the plot of the cost grid in figure 2.2, we see that the paths in figure 4.2b cross the areas with high hazard. To make the paths avoid areas with high hazard, we have to process the permeability further.

4.1.3 Processed Permeability

To force the streamlines of u to avoid areas with high hazard, we cut off all the values of K under a certain level. In figure 4.3 we have set all $K_{i,j}$ below two standard deviations of $\text{mean}(K)$ to 0.01. The results in figure 4.3b and 4.3c clearly show that many areas are avoided, and the paths follow the contours of the terrain. Out of all the evaluated paths in figure 4.3b, the green one has the least sum of hazard along it.



(a) The solution of the PAM equation, u.

(b) Streamlines of u between α and β.

Figure 4.2: Results from the PAM with cost grid. Indicated in red is the physically shortest streamline between point α and β, and the green streamline has the least risk associated with it. The permeability $K(H)$ concentrate the streamlines toward attractive areas in Ω

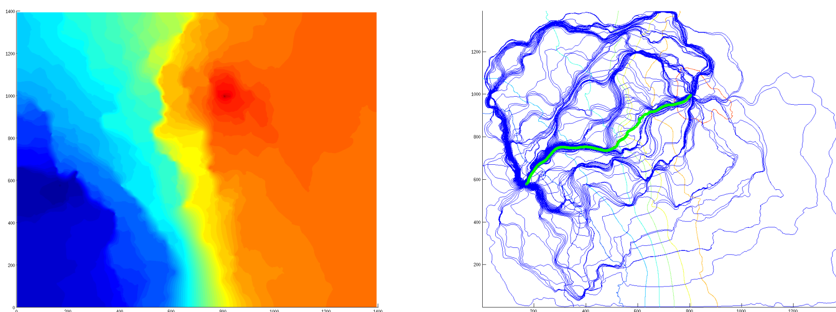
4.1.4 Evaluating Paths

The parameter space for this project is vast, there are many parameters that can be manipulated and tuned to get what we need. We will now present some results for the parameters used to process the paths.

Streamline Step Size:

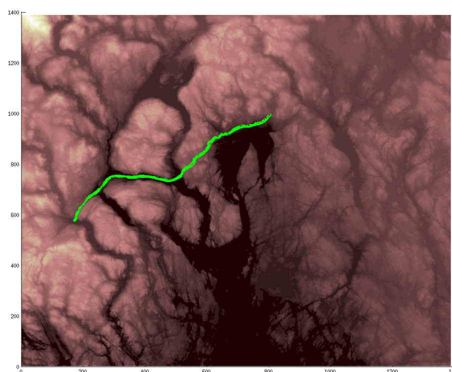
The ν streamlines from α to β is calculated using a built in function in MATLAB that is based on the gradient vector field from the solution u. We can set the maximum number of steps and step length when finding a streamline with the function `stream2`. In table 4.1 we compare the paths calculated between Kongsberg and Sinsen using the standard 2-norm and infinity norm. The results from the table indicate that the difference between the paths decreases for smaller step sizes. The infinity norm is relatively large compared with the 2-norm, thus indicating a local variation between the paths.

The three different paths used in table 4.1 is plotted in figure 4.4 where we have zoomed in on a local variation between the paths. The variation in the paths may seem to be set off by a sharp gradient in the the vector field. In the bottom of figure 4.4b there is an edge that separates the paths, before they converge together at the top of the figure. This effect may be reduced with smaller step sizes.



(a) The solution of the PAM equation, u.

(b) Streamlines of u between α and β.

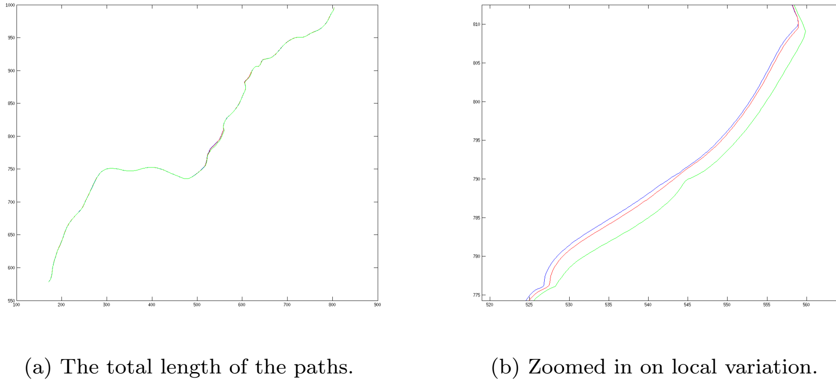


(c) The best path plotted in the terrain.

Figure 4.3: Results of the PAM when we force the paths to avoid regions associated with high risk. The paths are calculated between Kongsberg and Sinsen.

	$\|\cdot\|_\infty$	$\|\cdot\|_2$
$P_{0.1} - P_{0.2}$	1.039	6.960
$P_{0.2} - P_{0.4}$	3.019	23.79

Table 4.1: Comparison of the norm between paths calculated using different step sizes. The subscript on P indicates the step size used when calculating the streamlines. The paths are calculated between Kongsberg and Sinsen with the cut-off value set at $K_c = 0.01$.



(a) The total length of the paths. (b) Zoomed in on local variation.

Figure 4.4: Three approximations to P_{opt} for the same route, calculated with varying step length. $P_{0.1}$ is blue, $P_{0.2}$ is red and $P_{0.4}$ is green.

Streamline Seed Points:

We seed ν streamlines uniformly along a unit circle with point α in the center, such that every direction out from point A is tested. The subscript on P will now refer to the distance between neighbouring seed points.

In table 4.2 we have done three comparisons of the paths calculated with four different seed point densities. The results show that $P_{0.1} = P_{0.05}$ and $P_{0.01} = P_{0.005}$. To find $P_{0.005}$, 20 times as many paths have been checked compared to finding $P_{0.1}$, but there is not a very big difference in the paths.

	$\|\cdot\|_\infty$	$\|\cdot\|_2$
$P_{0.1} - P_{0.05}$	0	0
$P_{0.05} - P_{0.01}$	5.439	30.39
$P_{0.01} - P_{0.005}$	0	0

Table 4.2: Comparison of the norm between paths calculated using different number of seed points. The subscript on the P's indicate the interval between the seed points around the starting point A. The paths are calculated between Kongsberg and Sinsen with the cut-off value at $K_c = 0.01$.

Height Profile:

For the same case as used above in figure 4.3 we will now look at the height profile. Figure 4.5 shows the two paths P_{LP} and P_{BP} found by different selection

strategies. The yellow path P_{LP} has the lowest height profile in the terrain. The summed hazard along the P_{LP} is $S_N(P_{LP}) = 0.0150$, while for the green path P_{BP} the summed hazard is $S_N(P_{BP}) = 0.0146$. P_{BP} represents the path with the least evaluated summed hazard.

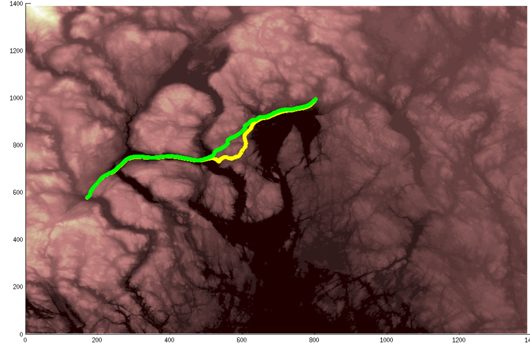


Figure 4.5: Two routes between Kongsberg and Sinsen chosen using different strategies. The green has the least sum of hazard along it and the yellow has the lowest height profile in the terrain.

To see the difference in elevation the two paths seen in figure 4.5 is interpolated onto the topography. In figure 4.6 we see that the two paths P_{LP} and P_{BP} are very similar in the beginning, but differ after the midpoint. When we compare figure 4.5 and 4.6 we see that the main variation between the two paths occur when crossing the terrain between Drammen and Sandvika, which is the part with the highest elevation along the route.

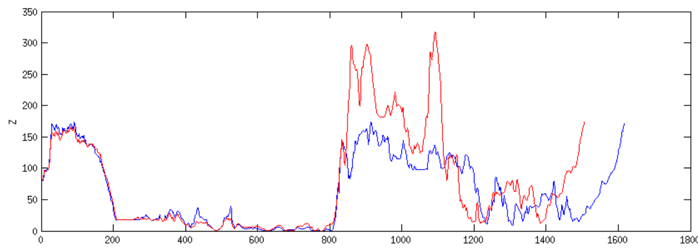


Figure 4.6: Height profiles for the path with the minimal hazard P_{BP}(red) and the path with the minimal height profile P_{LP}(blue).

4.1.5 Optimality Tests

We want to test whether the PAM really finds the optimal solution. For the test case from α to β, Kongsberg to Sinsen, we calculate a candidate for P_{opt}. We select a point lying roughly on the middle of P_{opt} and call this center point β_0. Next we calculate the optimal path in two steps, from α to β_0 and β_0 to β.

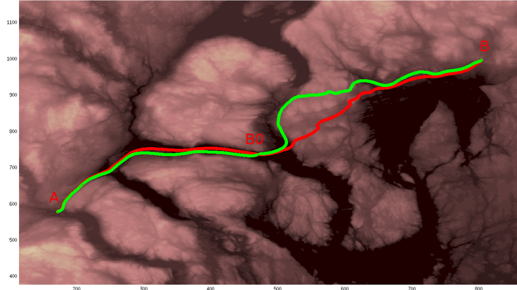


Figure 4.7: The red path $P_{\alpha\to\beta}$ is calculated with the PAM from α to β. The green path $P_{\alpha\to\beta_0\to\beta}$ is calculated with the PAM in two steps, α to β_0, then β_0 to β, where $\beta_0 \in P_{\alpha\to\beta}$.

The summed hazards are $S_N(P_{\alpha\to\beta}) = 0.0146$ and $S_N(P_{\alpha\to\beta_0\to\beta}) = 0.0139$ The results show that we are able to find a better solution when evaluating the sub routes, implying the the PAM does not find the optimal solution.

In figure 4.8 we test the PAM on a problem from α to β, then we test the reverse problem; β to α. We get two different paths, where the upper path has an accumulated hazard of 0.0020 and the lower path has 0.0023.

4.1.6 Exclusion Zones

In chapter 2 we discussed two methods of incorporating exclusion zones into the model. The easiest way with respect to the implementation is to change the permeability K directly. In figure 4.9 we can see how the exclusion zone added in the cyan circle around Gardermoen airport affects the path. The red path is calculated without any exclusion zone. After we change the permeability inside the cyan circle, the new green path avoids the exclusion zone.

4.1.7 Permeability-Mapping

We run a series of simulations to check to what extent the exponent p in the mapping, in equation (2.35) will affect the outcome. In figure 4.10 we vary the

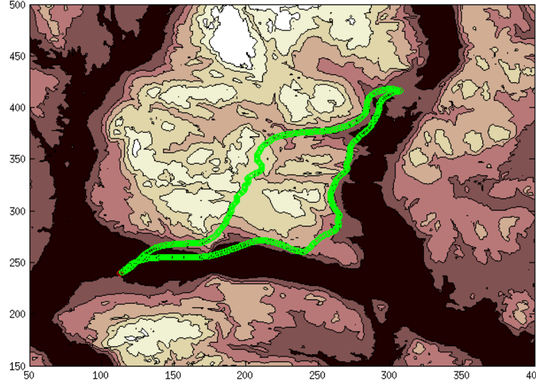


Figure 4.8: We have used the PAM from α to β, then from β to α. An optimal method would find the same path when reversing the problem. $S_N(P_{\alpha\rightarrow\beta}) = 0.0023$ and $S_N(P_{\beta\rightarrow\alpha}) = 0.0020$.

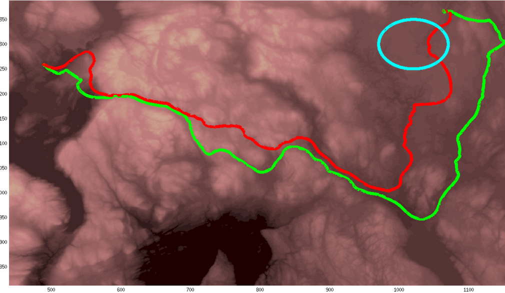
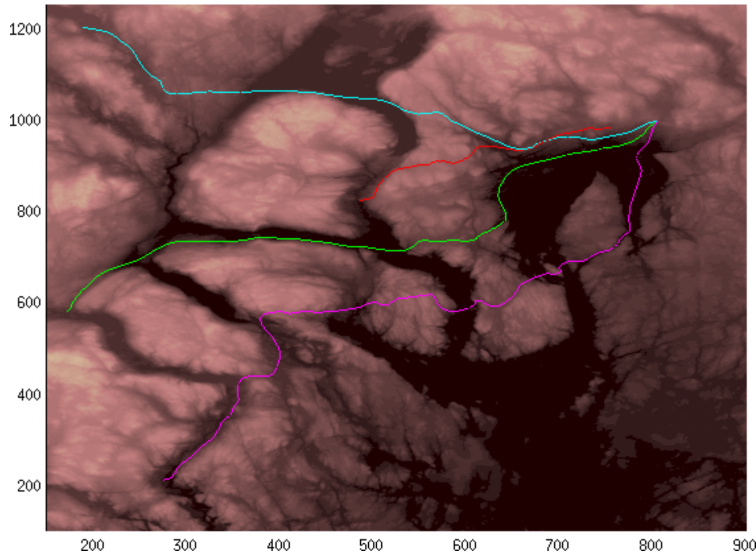


Figure 4.9: Exclusion zone using the permeability. In the area around Gardermoen airport we have adjusted the permeability such that the paths will go around this exclusion zone. When we add the cyan exclusion zone the new optimal path(green) is forced in a different path than the red.

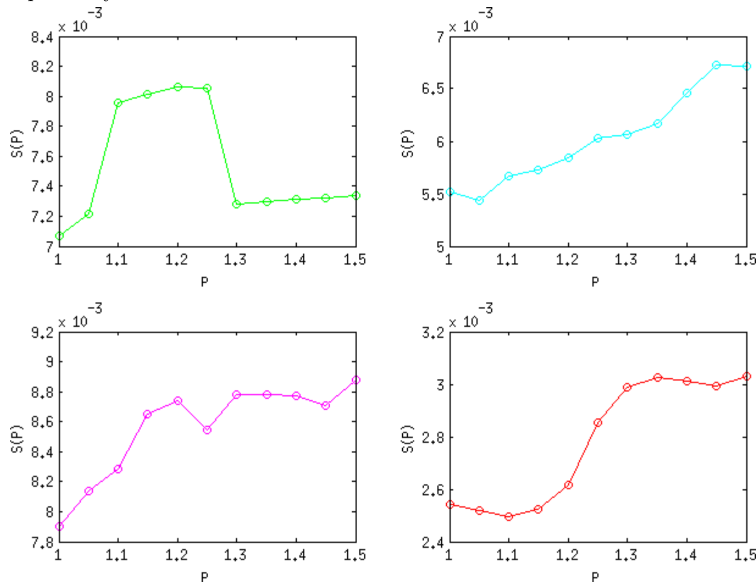
exponent $p \in [1, 1.5]$ and plot the different paths and accumulated hazard. We see that $p \approx 1$ results in the least accumulated hazard for all the four cases.

4.1.8 Permeability Cut-Off

When we cut off all values of K under a certain level we can drastically change the solution u and all streamlines. We run simulations where we vary the level the permeability is cut off. In figure 4.11 we have tested four different set ups with α and β with multiple simulations to choose the best cut off-level.

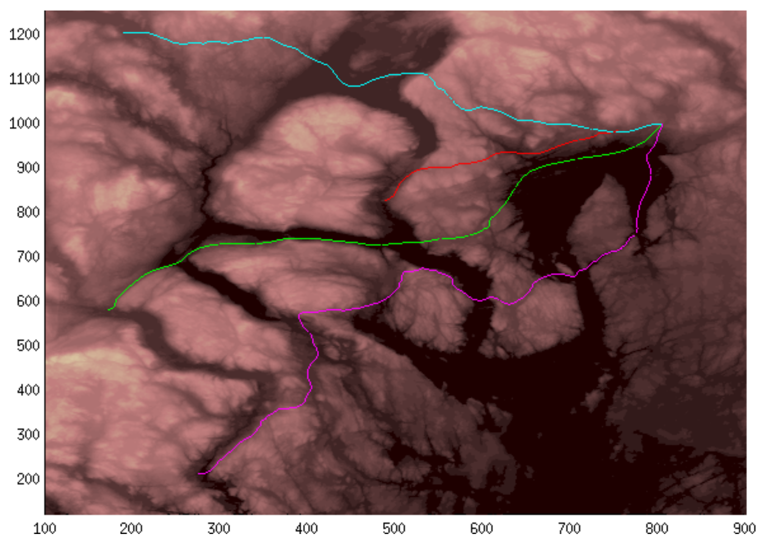


(a) Paths plotted with $p = 1, 1.05, 1, 1.1$ in exponent for colours green, cyan, purple/magenta and red respectively.

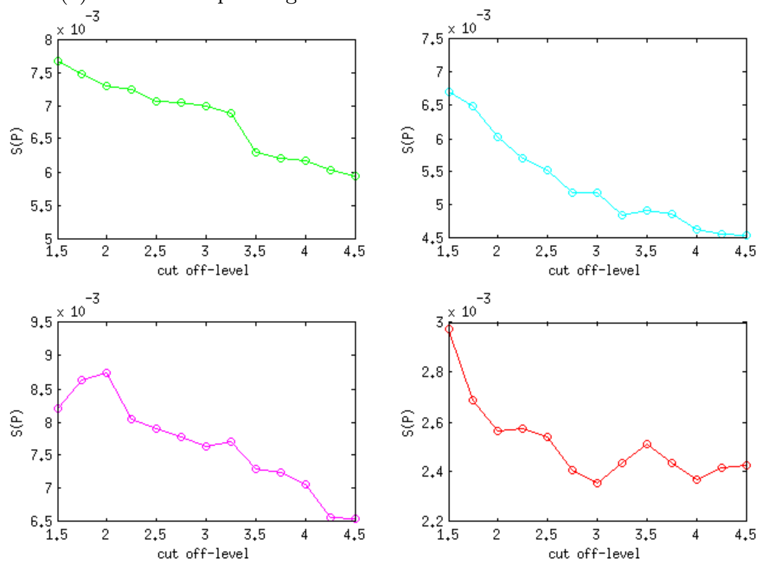


(b) Accumulated hazard $S_N(P)$ for paths calculated with varying exponent p in the permeability mapping from equation (2.35) .

Figure 4.10: We vary the exponent p in the permeability-mapping to obtain the one yielding the least accumulated hazard. All four cases seem to indicate that $p \approx 1$ is a good choice.



(a) Paths corresponding to the best choices of cut off-level below.



(b) The cut off-level is varied to obtain the one yielding lowest accmulated hazard.

Figure 4.11: The cut-off-level is chosen based on the accumulated hazard for the four different problems above. The pathfinder will take short cuts that represents infeasible paths when the cut off-level is high, because of this we have chosen the highest possible cut off-value still yielding a feasible path. Every path visualized above is calculated with a cut off-level at 3 standard deviations from the mean of K, except for the purple/magenta-colored which is at 2 standard deviations.

4.2 The PAM with Homogeneous Dirichlet BC

The boundary conditions determine some properties of the linear set of equations. We use Dirichlet boundary conditions on $\partial\Omega$ such that the matrix A is SPD and we can employ more efficient solvers on $Au = b$.

4.2.1 The Multigrid Method

The multigrid method has a great potential for solving the linear set of equations efficiently. We go through some basic testing of the algorithm before we apply it on the PAM. As a test case for our implementation we start by solving a homogeneous Poisson problem where the analytical solution is known. In this way we make sure all of the tools are working as expected before we tackle the slightly more involved problem that occurs when we introduce the variable permeability. The test problem is defined as:

$$\begin{aligned} -\nabla^2 u &= 2\pi^2 \sin(\pi x)\sin(\pi y) \qquad in \quad \Omega \\ u &= 0 \qquad on \quad \partial\Omega \end{aligned} \tag{4.1}$$

with exact smooth solution $u_{exact} = \sin(\pi x)\sin(\pi y)$. We solve equation 4.1 by iterating on the multigrid V-cycle until convergence is reached.

In table 4.3 we examine the convergence of the V-cycle for various smoothers. We run tests with k number of iterations on the V-cycle, which is stopped when $\frac{||r_k||_2}{||r_0||_2} < 10^{-12}$. The convergence factor r[9] is calculated using the formula

$$r = \exp\left(\frac{1}{k}\log\frac{||r_k||_2}{||r_0||_2}\right). \tag{4.2}$$

Table 4.3 shows that the line relaxation with Gauss-Seidel has a smaller convergence factor than both weighted Jacobi and ordinary Gauss-Seidel. For the weighted Jacobi we use $\omega = 4/5$. The tests are all done with $N = 2^5$ that is 31^2 unknowns and the smooth solution is visualized in figure 4.12. In figure 4.13 we plot the relative residual $\frac{||r_k||_2}{||r_0||_2}$ for every V-cycle until convergence. We use $(\nu_1, \nu_2) = (3, 3)$ and the tolerance is still set at 10^{-12}. Line relaxation with GS reduces the error more efficiently than both weighted Jacobi and GS, they respectively need 6,7 and 8 iteratrions to reach the wanted tolerance.

Figure 4.14 shows the discretization error on a logarithmic scale. We compare the numerical approximation with the exact solution in the infinity norm $||u - u_{exact}||_\infty$. The second order convergence of the central difference approximation is observed, as expected from equation (2.12).

smoother	(ν_1, ν_2)	r_{MG}	(ν_1, ν_2)	r_{MG}
ω-*Jac*	$(0, 1)$	0.501	$(1, 1)$	0.266
	$(0, 2)$	0.262	$(1, 2)$	0.135
	$(0, 3)$	0.132	$(1, 3)$	0.076
GS	$(0, 1)$	0.309	$(1, 1)$	0.107
	$(0, 2)$	0.088	$(1, 2)$	0.044
	$(0, 3)$	0.037	$(1, 3)$	0.016
line-GS	$(0, 1)$	0.269	$(1, 1)$	0.115
	$(0, 2)$	0.092	$(1, 2)$	0.057
	$(0, 3)$	0.016	$(1, 3)$	0.011

Table 4.3: Convergence testing for the multigrid V-cycle when solving Poissons problem with three different smoothers. We measure the different convergence factors. At the coarsest level the conjugate gradient method is used to solve the residual equation. The variables ν_1, ν_2 represent the number of pre- and postsmoothing steps. The restriction operation is done with the full weighting operator.

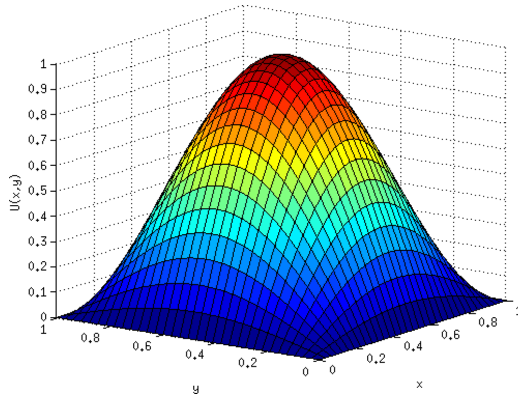


Figure 4.12: The numerical solution of equation (4.1), $u_{exact} = \sin(\pi x)\sin(\pi y)$. This serves as a test for our multigrid implementation.

V-cycle on the PAM

Next we introduce the cost grid and test the convergence of the V-cycle. We do not have an analytical solution to compare the numerical solution with, but we can use the direct solver as a reference. In figure 4.15 we see the convergence of GS is a bit slower than for the case in figure 4.13, but line-GS converges equally fast for the two cases. In addition the jacobi iteration does not converge when

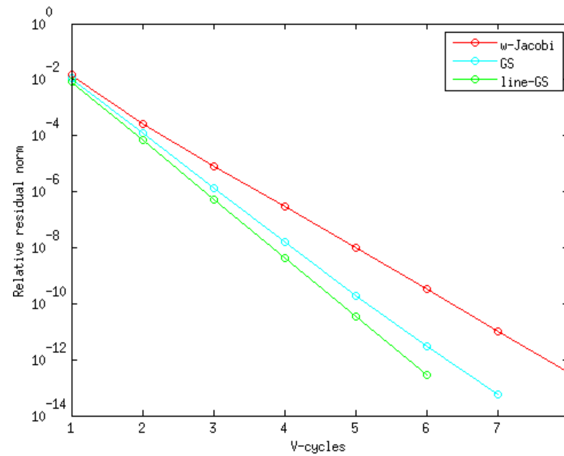


Figure 4.13: The relative residual is plotted on a logarithmic scale for every V-cycle until the tolerance at 10^{-12} is reached. This plot help to visualize the results found in table 4.3. Line relaxation with Gauss-Seidel is the most effective smoother among the three tested with respect to convergence of the V-cycle.

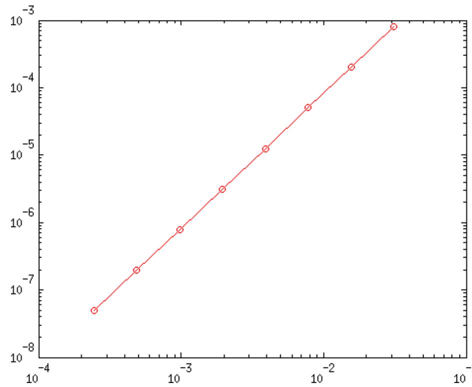


Figure 4.14: The discretization error is measured using the infinity norm $||u - u_{exact}||_\infty$ for various step sizes h. The convergence is of second order as expected from equation (2.12), since the the error term is $\mathcal{O}(h^2)$.

using the cost grid. The convergence testing is done with $N = 2^5$.

In figure 4.16 we present runtime results for the V-cycle. The figure shows the runtime per V-cycle, plotted against the problem size on a logarithmic scale. For each problem size, we run the V-cycle until convergence at $tol = 10^{-6}$. This

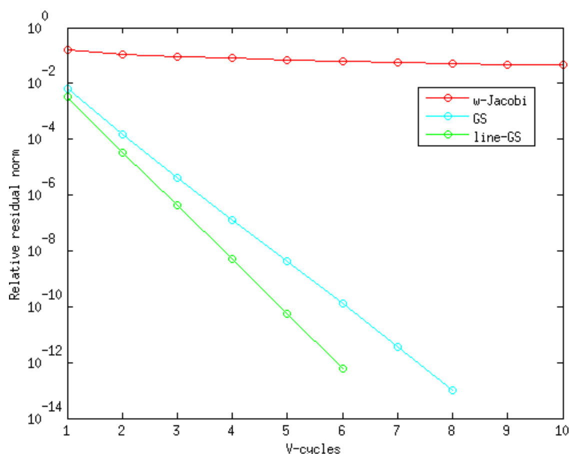


Figure 4.15: The relative residual for the for the V-cycle with cost grid. The tolerance is 10^{-12} and the problem size is $N = 2^5$.

is averaged over 10 simulations and we divide by the number of iterations needed to reach convergence. Because of the variance in the cost-grid associated with the different problem sizes, the number of iterations to get convergence varies, therefore, we look at the runtime per V-cycle to asses the complexity. We also compare the results with the linear runtime $\mathcal{O}(m)$ based on the first runtime. We observe the linear complexity as expected by equation (2.29).

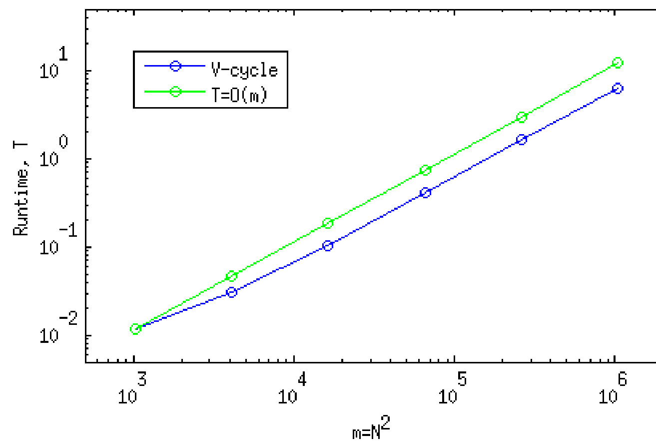


Figure 4.16: The runtime of the multigrid V-cycle is plotted on a logarithmic scale vs. the problem size $m = N^2$. The V-cycle is run and timed until convergence at $tol = 10^{-6}$. We take the average after 10 simulations and divide the total runtime, on the numbers of iterations.

4.2.2 Avoiding the Boundary

Dirichlet boundary conditions impose restrictions on the method because of streamlines that will not terminate in the sink in point β. To test the Dirichlet boundary conditions we run experiments where we measure the percentage of streamlines that go from point α to point β when we vary the position of α. A portion of the grid with size 400×400 is used to test paths starting along the boundary and ending up in the center.

Source Strength

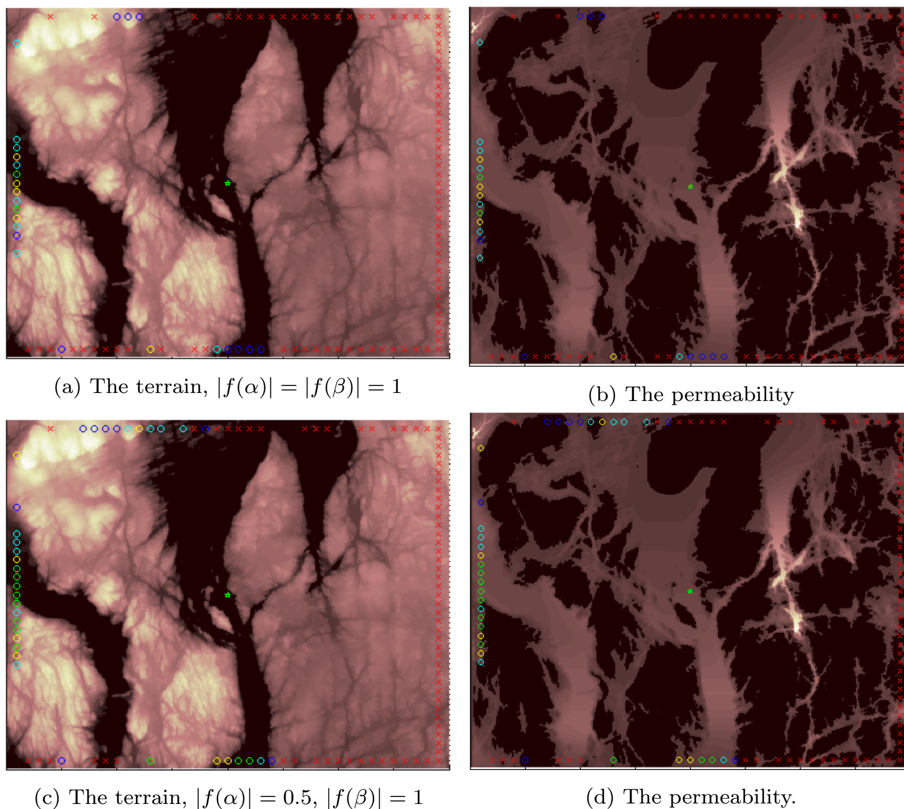
In figure 4.17 a surface plot of the terrain and permeability K is presented. We have varied the starting points α around the boundary, but β is kept constant in the center. The coloured circles in the plot indicate the percentage of streamlines, seeded in that starting point that make it to the destination β.

	0%	> 40%	> 50%	> 60%	> 70%
colour	× *red*	○ *blue*	○ *cyan*	○ *yellow*	○ *green*

Table 4.4: The interpretation of the colours used in figure 4.17.

The effect of varying the strength of the source is seen when comparing the two pairs of plot in 4.17. There is a higher percentage of paths reaching the center in point β when we reduce the source strength compared to the sink. In figure 4.17c and 4.17d the source is $|f(\alpha)| = 0.5$ and the sink is $|f(\beta)| = 1$.

In the permeability plots on the right side of figure 4.17 we see that the method fails for many starting points α. This is because the streamlines fail to reach β before they vanish to the boundary $\partial\Omega$. If there is very low permeability towards the target compared with the direction of the boundary, many streamlines will probably leave the domain. Because of this, the strength of the source and sink alone, might not be enough to force the streamlines toward the target in point β.



(a) The terrain, $|f(\alpha)| = |f(\beta)| = 1$

(b) The permeability

(c) The terrain, $|f(\alpha)| = 0.5$, $|f(\beta)| = 1$

(d) The permeability.

Figure 4.17: Paths from the boundary to the center is tested with Dirichlet boundary conditions. The strength of the source is varied. The colors indicate the percentage of streamlines that end up in $\beta = (200, 200)$ rather than the boundaries. Green: $> 70\%$, yellow: $> 60\%$, cyan: $> 50\%$ and blue: $> 40\%$. The red crosses mean that none of the seeded streamlines ended up in β. The starting points are plotted on top of the terrain on the left and the corresponding permeability to the right.

4.3 Dijkstra's Algorithm on Regular Grid

We consider the performance of Dijkstra's algorithm for this path finding problem. We test the PAM and Dijkstra on a small portion of the grid in $(x, y) \in [427, 777] \times [650, 1000]$, the results are shown in figure 4.19. The green paths shows the solution calculated with Dijkstra's algorithm. The cyan colored path is calculated with the PAM using homogeneous Dirichlet boundaries and a cut off at 1.7 standard deviations. We notice that Dijkstra finds a path that can only step in 45° angles since the edges E in the graph G is built according to the stencil in figure 2.10. Figure 4.18 shows three different problems solved by the PAM and Dijkstra, plotted as they traverse the cost-grid. The pairs of paths differ to a great extent, but they all seem to be reasonable flight paths.

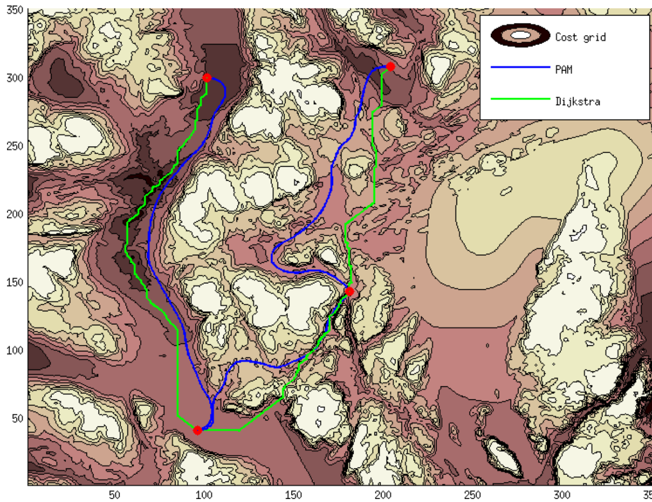
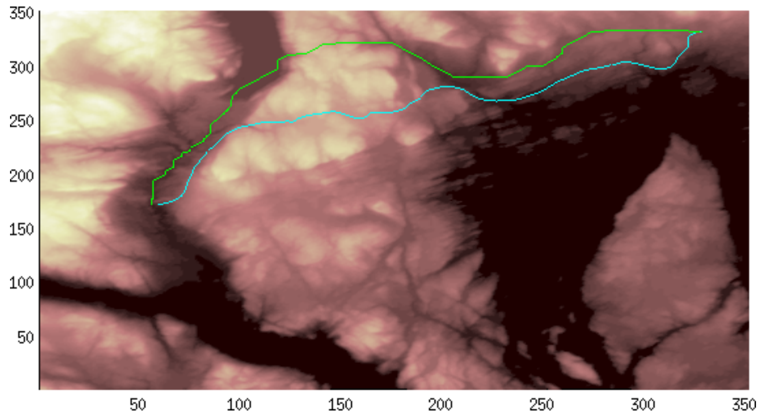
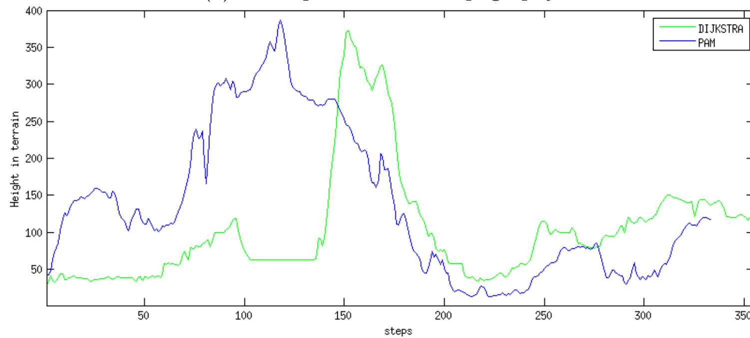


Figure 4.18: Paths for three different problems, solved with the PAM and Dijkstra. The hazard is visualized as a contour plot.

In figure 4.20 we present runtime results for this the Dijkstra-implementation. Because of the varying hazard-rates associated with the grids of different sizes, the runtime for each simulation is hard to measure ideally as m grows. We expect Dijkstra to have a quadratic complexity, and this is reflected in the four biggest simulations in figure 4.20.



(a) Paths plotted in the topography.



(b) Height profiles for the paths found by with Dijkstra and the PAM.

Figure 4.19: We find two different paths with the PAM(blue) and Dijkstra's algorithm(green). We use a small portion of the grid since Dijkstra is not suited for high resolution compared to the PAM. For the PAM we have cut off values below 1.7×standard deviation of mean(K).

4.4 Preprocessed Graph

In figure 4.21, we have found the edges for a small graph with the PAM. The nodes are handpicked and placed at favorable places in this case. The problem regarding incoming and outgoing flight directions, mentioned in section 2.9.2, is evident. The superposition of edges calculated with the PAM will become discontinuous in many of the nodes.

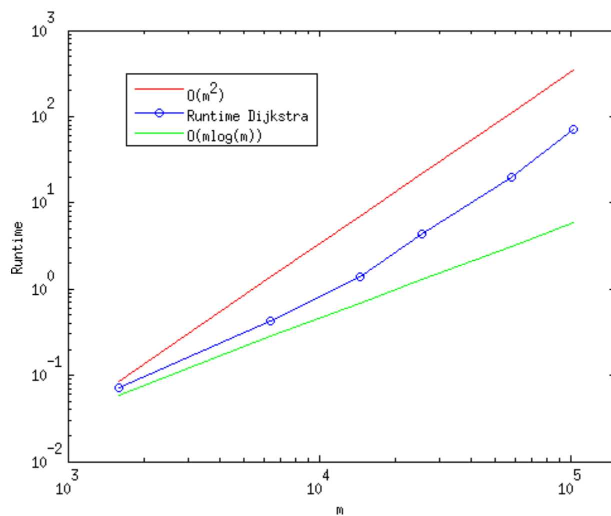


Figure 4.20: The runtime of Dijkstra is compared with $\mathcal{O}(m^2)$ and $\mathcal{O}(m \log m)$. There are $m = N^2$ unknowns in each simulation with $\alpha = (\frac{N}{4}, \frac{N}{4})$ and $\beta = (\frac{3N}{4}, \frac{3N}{4})$.

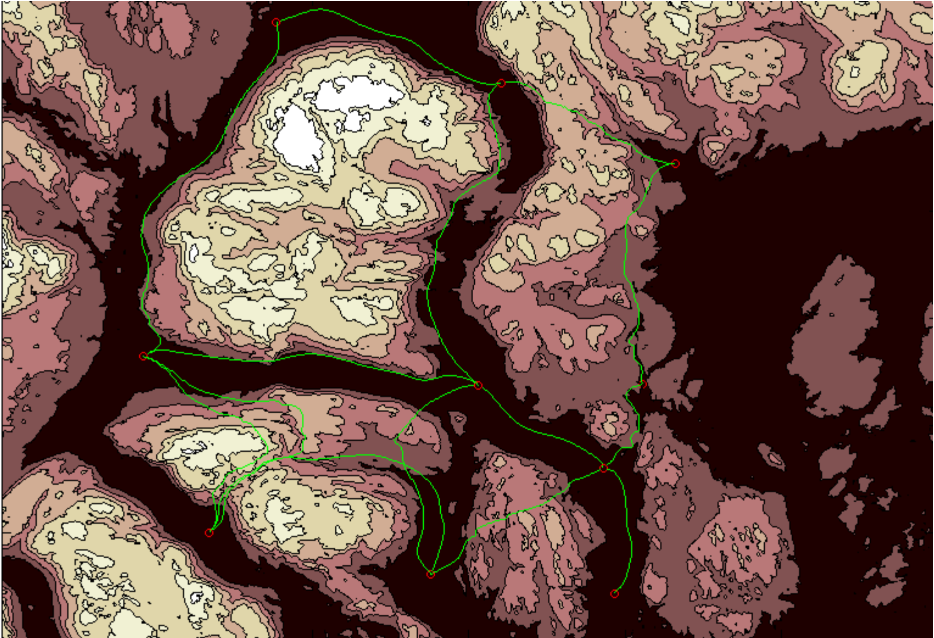


Figure 4.21: Graph with preprocessed edges, suited for a graph algorithm. The paths are non-flyable in many of the nodes.

4.5 Post Processing

In section 2.10 we have mentioned how get the three-dimensional path P from the two-dimensional path. In figure 4.22 we have visualized the path(green) after it has been lifted up from the terrain and smoothed. The red path P_T in figure 4.22 shows the terrain straight underneath P; we notice that P is smooth compared to P_T.

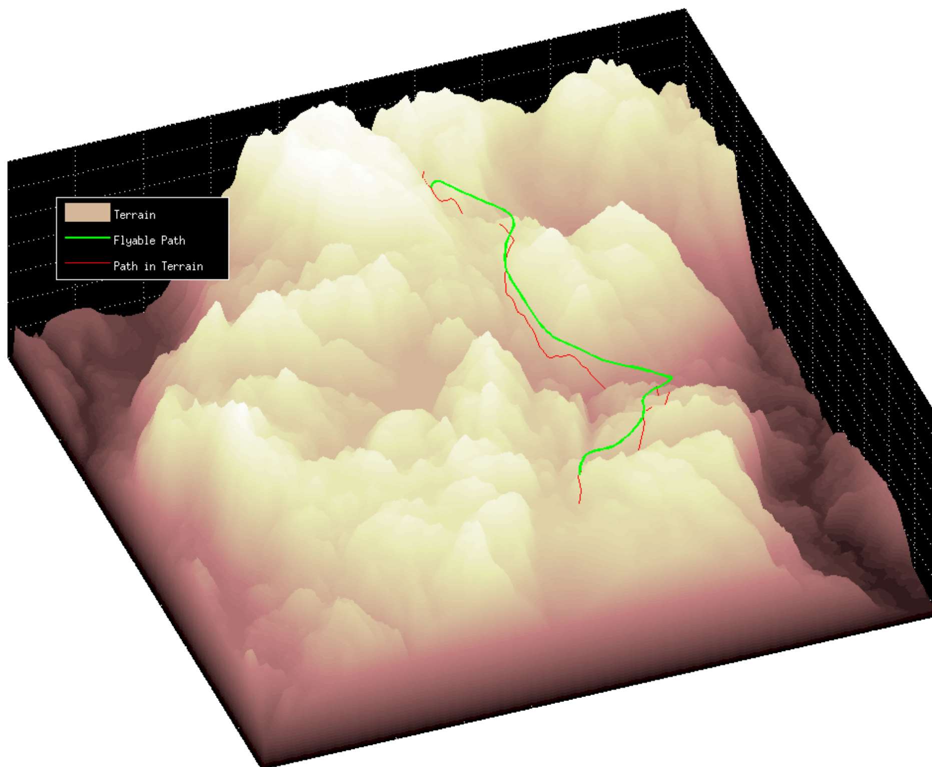
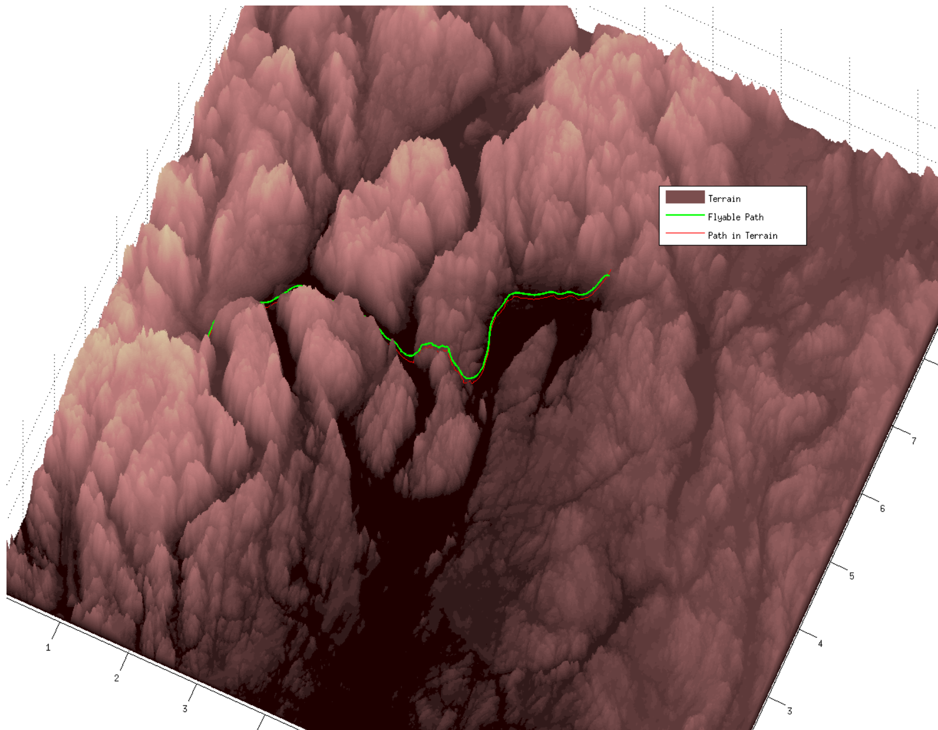
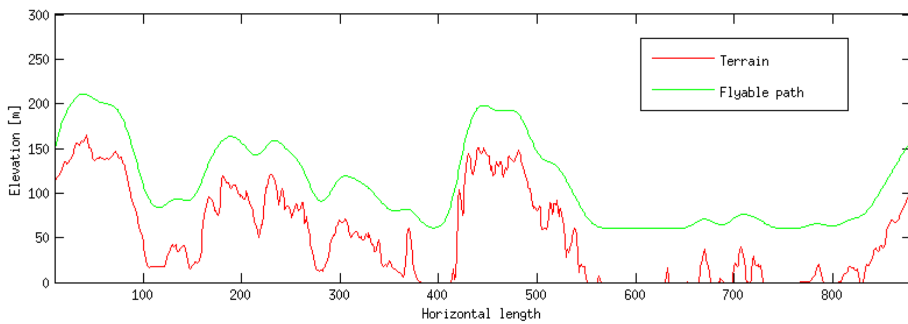


Figure 4.22: The processed terrain-following path is vizualized in the terrain. The red path can be viewed as a projection of P onto the topography.

In figure 4.23 we have visualized the results for a bigger problem, compared with 4.22. It is hard to see the green path in relation to the red path, but the elevation is plotted in figure 4.23b for the same problem.



(a) The topography with the solution P in green. The projection of P in the terrain is plotted in red.



(b) The vertical component of the path is plotted over the terrain.

Figure 4.23: Resuts from the pathfinding problem from Kongsberg to Oslo.

Chapter 5

Discussion

In this chapter we discuss the results found in previous chapter and we will try to shed some light on implications of the results.

5.1 Streamline Properties

The PAM consists of many different variables that can be tuned to yield a terrain-following, flyable path. Throughout section 4.1 we have investigated how different parameters affect the streamlines and the total ability to find a satisfactory path.

The most important parameter seems to be the COL - the processing of K that obstructs the streamlines from entering regions with high hazard. When the COL is too high or too low, the streamlines will not follow the terrain. Since we cut off values based on the mean and standard deviation of the hazard for the whole cost grid, we can not distinguish properly between attractive and unattractive areas at every scale in H. An improvement would be to iterate over segments of H, reducing the permeability locally based on the hazard values relative to each segment. The customization of K is key to getting good results from the PAM.

5.1.1 Optimality

We have shown that the Physical Analogy Method(PAM) does not find a global optimal path. The results in figure 4.7 and 4.8 both dismiss the PAM as an optimal pathfinder. In figure 4.7, we see that when we test a sub problem, it will find a new path than the original. In figure 4.8, we see that the direction we search for a path affects the result. Both of the plots seem to suggest that the placement of the source will affect the outcome to a great extent.

However, from what we can see in the different combinations of α and β we have run, the PAM will often find paths that seem sufficiently smooth and locally optimal for our purpose. Since the paths found with the PAM is relatively well behaved with respect to flight path-requirement, the method should not be totally dismissed based on the lack of optimality. A relatively good path can i.e. be post processed and smoothed in certain regions to yield a sufficient flyable path.

5.1.2 Boundary Conditions

We have looked at two different ways of implementing the boundary conditions; in section 4.1 we use Neumann boundaries in the simulations, and in section 4.2 the Dirichlet boundaries enables faster solvers such as multigrid. The Neumann boundaries forces every streamline to terminate in the destination β; however, this compromises the possibility of utilizing the multigrid or other parallelizable algorithms. The Dirichlet boundaries will restrict the method, but this is necessary, in order to satisfy the runtime requirements.

Dirichlet Boundary

The problem originating from the Dirichlet boundary conditions, is remedied to a certain extent with adjusting the source and sink strength. However, we have seen in figure 4.17 that the permeability of a certain problem in combination with the source placement will be an important factor for the streamlines ability to escape away from the boundary. Based on the results presented, a good practice is to avoid placing the source too close to the boundary $\partial\Omega$. The Dirichlet boundary is what enables an efficient implementation, but it is also the thing that compromises the reliability of the method.

5.1.3 Evaluating Paths

We have used three different methods to evaluate the streamlines in this thesis. We sum the hazard along the streamline, calculate the length and sum the elevation along the path. The three methods result in paths with different characteristics that might all be useful in different contexts. The calculation and evaluation of streamlines is linear $\mathcal{O}(m)$ in complexity for m unknowns.

Further work can be done to improve the evaluation of streamlines. We can accumulate the curvature along paths for a more general fuel consumption model, and we can combine these different methods.

The calculation- and evaluation-process of the ν streamlines is intrinsically parallel. We can simply distribute a portion of the ν seed-points to each of the processes, each finding a suggestion for the best path, which is then compared after the processes finish.

5.2 The Multigrid Solver

The results presented in section 4.2.1 are twofold. We start by presenting a test case for the mutigrid method on a two-dimensional homogeneous problem, where the analytic solution is known. The results indicate the expected discretization error, and we observe the different convergence factors for the different smoothers. With the multigrid method working for the homogeneous test case, we extend and customize the implementation for the inhomogeneous case with cost-grid.

With the cost-grid included in the implementation, the convergence of the method is reduced in general. The weighted Jacobi does not converge, while the convergence rate of Gauss-Seidel decreases slightly. Line-GS converges with the same number of iterations in the homogeneous and inhomogeneous case.

Despite the reduced performance for Jacobi and GS, the method is fully functionable with GS and line-GS. The multigrid method have optimal use of memory, which is not the case for many solvers, and this is very important in some settings where there is restrictions on the memory usage.

5.2.1 Parallelization

We have not utilized parallelization at any level in our implementation, but the smoothing processes can be parallelized with different methods, i.e. multicolouring and domain decompositions, for a specific fast implementation.

5.3 Extension to 3D

We have only considered path optimization in the horizontal plane with the PAM and Dijkstra in this thesis. However, since the cost grid is based on the topography, the three-dimensional properties of the terrain is reflected in the two-dimensional grid. Because of this, a two-dimensional approach is sufficient and it is faster than a three-dimensional approach for the equations. When the 2D path is lifted up from the terrain and smoothed, the final product is a 3D path traversing the terrain.

An extension to three-dimensional numerical model would be more attractive for a FEM-implementation. With the topography represented as an unstructured grid, we would have a more accurate model of the problem, and get a streamline representing a flyable path without post-processing. With FEM-discretization we would have to implement a different solver for the linear set of equations since we would lose the simple banded structure of the matrix. Algebraic Multigrid(AMG) methods are suitable for discretizations on unstructured grids.

5.4 Error from Grid Simplification

In section 2.8.9 we presented a theoretical result on the error we introduce when not explicitly using the elevation in the domain. The error Δ between using the physical domain and our method, which is a flat domain, will be dependent the gradient of the topography and the scales of the domain. In equation (2.51), g' represents the gradient of the topography, and if $|g'|$ is large, we assume $|\Delta| > 0$.

We have not directly been able to measure the effect of Δ in any of our simulations. With a FEM-implementation, the Δ might be seen implicitly as a change in the streamlines compared to our FD-implementation.

5.5 Runtime

The bottleneck for a pathfinding algorithm based on the PAM will be the linear solver of $Au = b$. The computations related to finding the gradient ∇u, streamlines and the evaluation of streamlines are all of linear complexity, for constant ν streamlines.

As seen in the theoretical complexity for the V-cycle in equation (2.29), and backed by the results in figure 4.16, the runtime for the V-cycle scales as $\mathcal{O}(m)$. The number of V-cycle iterations to yield convergence will depend on the problem, but it is not proportional with m, meaning that the total multigrid solver is also linear $\mathcal{O}(m)$.

In an efficient implementation, every level of a serial code should be optimized to yield the best possible performance for a parallel implementation. When tailoring a program for a specific computer architecture; the compilation of code, memory usage and data handling can be crucial for getting the best possible performance. This thesis has not been geared towards the implementation of a fully optimized code, we have only considered the problem algorithmically to investigate the potential for an efficient implementation.

Chapter 6

Conclusion

This thesis explores the use of a PDE for solving a 3D pathfinding problem, via an analogy to fluid flow. The method is called the Physical Analogy Method(PAM) and it relies on hazard rates, describing the probability of survival in the domain, which is converted to an artificial permeability. The method explores streamlines in a velocity vector field to find low level, terrain-following, flyable paths through a hostile terrain. The selection process of these streamlines can be weighted towards finding the shortest, or the lowest path while keeping the probability of survival high. The streamlines tend to be relatively smooth and they can also avoid no-fly zones.

An important aspect of the thesis is the potential for solving the problem efficiently on-board an unmanned aerial vehicle during flight. Due to the optimal memory usage, linear complexity and potential for a parallel implementation, the multigrid method was chosen for solving the linear set of equations. Homogeneous Dirichlet boundary conditions are used to enable the multigrid method, but this also restricts the method because streamlines may not terminate in the target location. For a high performance implementation, the method must guarantee a flyable path on request, so this problem should be further addressed in future work. The PDE discretization is based on a two dimensional equation, but the final path is found by extrapolation of the 2D streamlines and the topography in the domain.

The PAM does not find a global optimal solution for the path finding problem, as many graph algorithms will for a shortest path problem. However, the streamlines will tend to favour paths through the terrain with low hazard, and in this way we find attractive aerial paths. The transformation of hazard rates into permeability is essential, and the characteristics of the streamlines are set by this transformation. A better handling of the permeability is interesting with

respect to future work. The method is not capable of enforcing incoming- and outgoing flight directions from a location. This will cause discontinuous first derivatives when calculating paths from α to β via a must fly zone. Because of discontinuous flight directions when calculating paths in multiple steps, a hybrid implementation of the shortest path problem, on a graph preprocessed with the PAM is dismissed.

Bibliography

[1] Luitpold Babel. Three-dimensional Route Planning for Unmanned Aerial Vehicles in a Risk Environment. *J Intell Robot Syst*, 2012.

[2] Donald A. Nield Adrian Bejan. *Convection in Porous Media, Fourth Edition.* Springer, 2013.

[3] John Rozier Cannon. *The One-dimensional Heat Equation, Fourth Edition.* Camgridge, 1984.

[4] L.C Evans. *Partial Differential Equations, Second Edition.* American Mathematical Society, 2010.

[5] Michael G. Kay. dijkstra.m and pred2path.m. `http://code.google.com/p/bnt/source/browse/trunk/graph/?r=59`, 2001. Version 5.

[6] Erwin Kreyszig. *Advanced Engineering Mathematics, Ninth Edition.* Wiley, 2006.

[7] William L. Briggs Van Emden Henson Steve F. McCormick. *A Multigrid Tutorial, Second Edition.* Siam, 2000.

[8] Alfio Quarteroni. *Numerical Models for Differential Problems.* Springer, 2009.

[9] Yousef Saad. *Iterative Methods for Sparse Linear Systems, Second Edition.* SIAM, 2003.

[10] R. Rivest T. Cormen, C. Leiserson and C. Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, Cambridge, Massachusetts, 2009.

[11] Hàn Thê Thành. *Micro-typographic extensions to the TEX typesetting system.* PhD thesis, Masaryk University Brno, 2000.