MASTER THESIS

NUMERICAL DIFFERENTIATION ON IRREGULAR GRIDS USING FINITE POINTSET METHODS

Written by:
Manh Hong Duong

Supervisors:
Prof. Dr. Axel Klar
Dr. Sudarshan Tiwari
Dipl. Math. Stephan Martin

Technische Universiteit Eindhoven
Technische Universitaet Kaiserslautern

Germany, 2010
Acknowledgments

At first, I express my sincere gratitude to my supervisors Prof. Dr. Axel Klar, Dr. S. Tiwari and Dipl. Math. Stephan Martin for the constant help and guidance throughout the course of this thesis. Without their help, this thesis would not have come out as this present shape.

Further I am thankful to people in the international relations office (STU) in TU Eindhoven and in the international school for graduate study (ISGS) in TU Kaiserslautern for their support during my studying in Eindhoven and Kaiserslautern.

Last but not least, I would like to thank my beloved family.
Declaration

I, hereby, declare that this thesis has been written only by the undersigned and without any assistance from third parties. Furthermore, I confirm that no sources have been used in the preparation of this thesis other than those indicated in the thesis itself.

Kaiserslautern, 05 August 2010.

Manh Hong Duong
# Contents

1 Introduction

2 Finite Pointset Method for approximation of derivatives
   2.1 Literature Overview ........................................... 9
   2.2 The moving least square method ................................ 10
   2.3 The finite pointset method .................................... 10
       2.3.1 Neighborhood searching .................................. 13
       2.3.2 Point cloud management ................................ 13
       2.3.3 Weight functions ........................................ 14
   2.4 General FPM algorithm for approximation of derivatives ...... 15

3 Mathematical model for the fibre lay-down process in the non-woven production
   3.1 Introduction .................................................... 17
   3.2 Stochastic model for the fibre lay-down process ............... 19
   3.3 The deterministic ODEs system ................................ 21

4 Application of the FPM in the fibre lay-down process .......... 24
   4.1 Generating the point cloud .................................. 24
   4.2 Neighborhood Selecting ....................................... 25
   4.3 Filtering the points ......................................... 31
   4.4 Choosing weight functions ................................... 32
   4.5 Algorithm ...................................................... 32
   4.6 Error estimations ............................................ 34

5 Numerical Implementation and Examples .......................... 38
   5.1 Testing Examples ............................................. 38
   5.2 Conclusions of the numerical results ........................ 55

6 Summary
   Bibliography ...................................................... 57
List of Figures

3.1 Felt: a non-woven fabric cloth .......................... 18
3.2 melt blown process ........................................ 18
3.3 Fiber on the non-moving conveyor belt ................... 19
3.4 The exponential function .................................. 22
4.1 The point clouds with 40 and 100 energy levels ............ 25
4.2 A point has not enough neighborhoods ........................ 27
4.3 A point has too many neighborhoods ...................... 28
4.4 neighborhoods are (nearly) in one line .................... 30
4.5 The point now has only 12 neighborhoods ................ 32
4.6 the domain formed by the inner and outer curves .......... 36
5.1 regular grid (left) with $\delta_x = 0.025$, and irregular grid (right) with n=50 . . 39
5.2 Testing function: $f(x, y) = x^2 y$ .......................... 40
5.3 $L^2$-norm of errors for regular grids. Testing function: $f(x, y) = x^2 y$ .... 41
5.4 $L^2$-norm of errors for irregular grids. Testing function: $f(x, y) = x^2 y$ .... 42
5.5 errors with boundary (green), without 5 outmost curves (red) and without 10 outmost curves (black). Testing function: $f(x, y) = x^2 y$ ............... 43
5.6 Testing function: $f(x, y) = \sin(x) \cos(y)$ .................. 44
5.7 $L^2$-norm of errors for regular grid. Testing function: $f(x, y) = \sin(x) \cos(y)$ 45
5.8 $L^2$-norm of errors for irregular grid. Testing function: $f(x, y) = \sin(x) \cos(y)$ 46
5.9 errors with boundary (green), without 5 outmost curves (red) and without 10 outmost curves (black) ........................ 47
5.10 $f(x, y) = \exp \left( -\frac{(x-1)^2+(y-\pi/2)^2}{4} \right)$ ......................... 48
5.11 $L^2$-norm of errors for regular grid. Testing function: $f(x, y) = \exp \left( -\frac{(x-1)^2+(y-\pi/2)^2}{4} \right)$ 49
5.12 $L^2$-norm of errors for irregular grid. Testing function: $f(x, y) = \exp \left( -\frac{(x-1)^2+(y-\pi/2)^2}{4} \right)$ 50
5.13 errors with boundary (green), without 5 outmost curves (red) and without 10 outmost curves (black). Testing function: $f(x, y) = \exp \left( -\frac{(x-1)^2+(y-\pi/2)^2}{4} \right) 51$
5.14 $\exp(x + y)$ .................................................... 52
5.15 $L^2$-norm of errors for regular grids. Testing function: $f(x, y) = e^{x+y}$ .......... 53
5.16 $L^2$-norm of errors for irregular grids. Testing function: $f(x, y) = e^{x+y}$ ...... 54
5.17 errors with boundary (green), without 5 outmost curves (red) and without 10 outmost curves (black) (for \(n = 100, 120, 150, 180, 200\)). Testing function:

\[ f(x, y) = e^{x+y} \]
Chapter 1

Introduction

Numerical differentiation plays an important role in numerical mathematics. The most common method to approximate derivatives is the Finite Difference Method (FDM). In the FDM, the computational domain is discretized into mesh grids and the differential operators are approximated based on these grid points. The FDM has been used and shown advantages when the geometry of the problem is regular, e.g. rectangular in 2D. However, it often runs into problems when mesh cells/elements deteriorate because of changes of the geometry. For instance, in a Lagrangian type of computations, one may experience mesh distortion. In this situation, more computational effort need to be done; otherwise, it can result in drastic deterioration of accuracy. To overcome the disadvantage of grid-based methods, many mesh-free approaches have been developed. The Finite Pointset Method (FPM), which has been introduced by Kuhnert ([6]), goes along this direction.

In the FPM, the computational domain as well as the boundaries are represented by a finite number of particles (pointset) and the spatial differential operators at each particle position are approximated from its surrounding clouds of particles with the help of the weighted moving least squares method. FPM is suitable to handle, for examples, flow problems with complicated and rapidly changing geometry ([8]), free surface flows ([13]) and multiphase flows ([16]), simulation of the liquid-liquid flow field in an extractor([3]). The FPM has advantage over the grid-based methods since it does not require mesh discretization. In addition, compared to other mesh-less methods, e.g., the Smoothed Particle Hydrodynamics method, the FPM is more flexible in handling the boundary particles and conditions. In the FPM, boundary conditions can be implemented in a natural way just by placing the particles on boundaries and prescribing the boundary conditions on boundary particles ([6]).

Mathematical modelling and simulation of the lay-down of flexible fibers onto a conveyor belt play an important role in the non-woven production. The understanding of the process is of great importance for quality improvements, design of new machines and optimization of production processes. In [18], the authors presented a simplified stochastic model for the fiber lay-down process. The model is based on a stochastic differential equation taking
into account the motion of the fiber under the influence of turbulence.

The purpose of this thesis is to apply the FPM for approximation of derivatives of a function where point cloud and data values arise from the model in ([18]). The content of the thesis is as follows.

In chapter 2, we start by introducing the moving least square method since it is the main ingredient of the FPM. Afterwards, the FPM for approximation of derivatives is introduced. Some techniques that need to be done in the framework of FPM such as neighbor searching, point cloud management and choosing weight functions are also discussed.

In chapter 3, we introduce the mathematical model for the lay-down process presented in ([18]). After that, we consider numerical solutions to the deterministic system of the model since the point cloud that the thesis is dealing with arises from that system.

In the next chapter, we apply the FPM to approximate derivatives of a function with the point cloud coming from the model presented in chapter 3. Due to a special feature of the point cloud, neighbors selecting and filtering need to be done and are introduced subsequently. The chapter is ended with an algorithm and error estimations for the FPM method applied in the lay-down process.

In chapter 5, we give numerical results for the algorithm in the chapter 4 with some testing functions. We compare the results with regular grids. We also calculate the errors without boundaries in order to see where the errors arise. Finally, the thesis is ended with a summary in chapter 6.

---

\(^0\)The logos on the cover page are taken from the web sites of the Erasmus mundus program, Eindhoven university of technology and Kaiserslautern university of technology
Chapter 2

Finite Pointset Method for approximation of derivatives

This chapter introduces the Finite Pointset Method (FPM) for approximation of derivatives of a function. Techniques need to be done in the framework of the FPM such as neighbor searching, point cloud management are also discussed. To begin with, we have an overview on the literature of the FPM and the moving least square method since it is the main ingredient in the FPM.

2.1 Literature Overview

The finite pointset method (FPM) has been introduced by Kuhnert ([6]) and is a meshfree Lagrangian type. In the FPM, a continuum domain (fluid or solid) is represented by a finite number of particles (pointset). Each finite point moves with the fluid velocity and carries all fluid information, like density, velocity, pressure, and temperature. The distribution of the pointset can be arbitrary and changes during simulation. The boundaries are also approximated by a finite number of boundary particles and boundary conditions are prescribed on them. The main feature of the FPM is that the spatial differential operators at each particle position are approximated from its surrounding particles with the help of the (weighted) moving least squares method.

The FPM has advantage over the grid-based methods since it does not require mesh discretization. In addition, compared to other mesh-less methods, e.g. the Smoothed Particle Hydrodynamics method, the FPM is more flexible in handling the boundary particles and conditions. FPM is suitable to handle, for examples, flow problems with complicated and rapidly changing geometry ([8]), free surface flows ([13]) and multi-phase flows ([16]), simulation of the liquid-liquid flow field in an extractor([3]). For industrial applications and a list of literatures of the FPM, readers can see in ([17]).

In the next section, we will shortly introduce the moving least square method.
2.2 The moving least square method

**Definition 1.** Let \( \{x_i\}_{i=1}^N \) be a set of distinct data points in \( \mathbb{R}^d \) (\( d = 1, 2, 3 \)), and let \( \{f(x_i)\}_{i=1}^N \) be the data values at these points. Denote by \( P_m \) the set of all polynomials \( p \) of order \( m \) in \( \mathbb{R}^d \). The moving least square approximation of order \( m \) at a point \( x \in \mathbb{R}^d \) is the value \( p^*(x) \) where \( p^* \in P_m \) is minimizing, among all \( p \in P_m \), the weighted least square error

\[
J = \sum_{i=1}^{N} (p(x_i) - f(x_i))^2 w(||x - x_i||),
\]

where \( w \) is a non-negative weight function and \( || \cdot || \) is the Euclidean distance in \( \mathbb{R}^d \).

Assume \( b(x) = [b_1(x),...,b_k(x)] \) is the polynomial basis vector of \( P_m \). Then \( p(x) \in P_m \) can be expressed as

\[
p(x) = b(x)^T c = b(x) \cdot c
\]

where \( c = [c_1,...,c_k]^T \) is the vector of unknown coefficients.

The minimizer’s coefficients satisfy the condition \( \nabla_c J = 0 \) where \( \nabla_c = [\partial / \partial c_1,...,\partial / \partial c_k] \).

Hence

\[
\nabla_c J = \sum_i w(d_i)2b(x_i)[b(x_i)^T c(x) - f_i] = 0
\]

\[
2 \sum_i [w(d_i)b(x_i)^T c(x) - w(d_i)b(x_i)f_i] = 0
\]

\[
\sum_i w(d_i)b(x_i)^T c(x) = \sum_i w(d_i)b(x_i)f_i
\]

where \( d_i = ||x - x_i|| \). If the matrix \( \sum_i w(d_i)b(x_i)^T \) is nonsingular then

\[
c(x) = \left[ \sum_i w(d_i)b(x_i)^T \right]^{-1} \sum_i w(d_i)b(x_i)f_i.
\]

and

\[
p(x) = b(x)^T \left[ \sum_i w(d_i)b(x_i)^T \right]^{-1} \sum_i w(d_i)b(x_i)f_i.
\]

For more information on the moving least square methods, readers can read ([4]).

2.3 The finite pointset method

Let us recall the Taylor expansion for multivariable function
Theorem 2.3.1. Let $B$ be a ball in $\mathbb{R}^d$ centered at a point $a$, and $f$ be a real-valued function defined on the closure $\overline{B}$ having $n+1$ continuous partial derivatives at every point. Taylor’s theorem asserts that for any $x \in B$

$$f(x) = \sum_{|\alpha|=0}^{n} \frac{\partial^{\alpha} f(a)}{\partial x^{\alpha}} (x-a)^{\alpha} + \sum_{i=1}^{d} o(|x_i - a_i|^n).$$

Note that the above formula uses the multi-indices notation, i.e., for $\alpha = (\alpha_1, ..., \alpha_d), x = (x_1, ..., x_d) \in \mathbb{R}^d$ then $|\alpha| = \sum_{i=1}^{d} \alpha_i, x^\alpha = \prod_{i=1}^{d} x_i^{\alpha_i}, \alpha! = \alpha_1!...\alpha_d!$, etc.

When $n = 2, d = 2$ the Taylor expansion is as follows

$$f(x) = f(a) + \frac{\partial f(a)}{\partial x}(x_1 - a_1) + \frac{\partial f(a)}{\partial y}(x_2 - a_2) + \frac{1}{2} \frac{\partial^2 f(a)}{\partial x^2}(x_1 - a_1)^2$$

$$+ \frac{\partial^2 f(a)}{\partial x \partial y}(x_1 - a_1)(x_2 - a_2) + \frac{1}{2} \frac{\partial^2 f(a)}{\partial y^2}(x_2 - a_2)^2 + o(|x_1 - a_1|^2 + |x_2 - a_2|^2).$$

where

$$\frac{\partial}{\partial x} = \frac{\partial}{\partial x_1}, \quad \frac{\partial}{\partial y} = \frac{\partial}{\partial x_2}.$$

Having the moving least square method and Taylor expansion in hand, we are ready to introduce the Finite Pointset Methods.

Approximation of partial derivative problem: Let $\Omega \subset \mathbb{R}^2$ be a domain, $\{x_i\}_{i=1}^{N}$ be a set of distinct data points in $\Omega$, and $\{f_i = f(x_i)\}_{i=1}^{N}$ be the data values at these points. We want to approximate the first and second order derivatives of $f$ at any data point.

Finite Pointset Method (FPM) will approximate the derivatives of $f$ at a point $x = (x, y) \in X$ in term of its values of its neighboring points using Taylor expansion and moving least square methods. In other words, the FPM is nothing but the moving least square approximation of order 2 to $f$ with respect to the polynomial basis $b(x) = [1, x, x^2, xy, y^2]^T$. The coefficients $c_1, ..., c_k$ in (2.2) are the partial derivatives.

Let $P(x, r) = \{x_i = (x_i, y_i): i = 1, 2, ..., n\}$ be the set of $n$ neighbor points of $x = (x, y)$ in a ball of radius $r$. Consider the Taylor expansion of $f(x_i)$ around $x$:

$$f(x_i) = f(x) + \frac{\partial f(x)}{\partial x}(x_i - x) + \frac{\partial f(x)}{\partial y}(y_i - y) + \frac{1}{2} \frac{\partial^2 f(x)}{\partial x^2}(x_i - x)^2$$

$$+ \frac{\partial^2 f(x)}{\partial x \partial y}(x_i - x)(y_i - y) + \frac{1}{2} \frac{\partial^2 f(x)}{\partial y^2}(y_i - y)^2 + e_i$$  \hspace{1cm} (2.3)
for \(i = 1, \cdots, n\), where \(e_i\) is the error in the Taylor expansion at the point \(x_i\). Denote

\[
a_1 = \frac{\partial f(x)}{\partial x} = f_x, \quad a_2 = \frac{\partial f(x)}{\partial y} = f_y, \quad a_3 = \frac{\partial^2 f(x)}{\partial x^2} = f_{xx},
\]

\[
a_4 = \frac{\partial^2 f(x)}{\partial x \partial y} = f_{xy}, \quad a_5 = \frac{\partial^2 f(x)}{\partial y^2} = f_{yy},
\]

\[
dx_i = x_i - x, \quad dy_i = y_i - y
\]

and

\[
a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_5 \end{pmatrix}, \quad b = \begin{pmatrix} f(x_1) - f(x) \\ f(x_2) - f(x) \\ \vdots \\ f(x_n) - f(x) \end{pmatrix} = \begin{pmatrix} f_1 - f \\ f_2 - f \\ \vdots \\ f_n - f \end{pmatrix}, \quad e = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix}
\]

and

\[
M = \begin{pmatrix} dx_1 & dy_1 & \frac{1}{2}dx_1^2 & dx_1dy_1 & \frac{1}{2}dy_1^2 \\ dx_2 & dy_2 & \frac{1}{2}dx_2^2 & dx_2dy_2 & \frac{1}{2}dy_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ dx_n & dy_n & \frac{1}{2}dx_n^2 & dx_ndy_n & \frac{1}{2}dy_n^2 \end{pmatrix}
\]

Then the system of equations (2.3) can be written as

\[
e = Ma - b
\]

The approximation of the vector \(\mathbf{a}\) is obtained by minimizing the error \(\mathbf{e}\) using moving least square methods. Let \(w: \mathbb{R}^2 \rightarrow \mathbb{R}_+\) be a positive weight function. The unknowns \(\mathbf{a}\) are computed by minimizing a weighted error over the neighboring points. Thus, we have to minimize the following quadratic form

\[
J = \sum_{i=1}^n w_i e_i^2 = (Ma - b)^T W(Ma - b)
\]

with \(w_i\) is the weight assigned to \(x_i\) and \(W = \text{diag}(w_1, \cdots, w_n)\). The minimizer \(\mathbf{a}\) satisfies the condition \(\nabla_a J = 0\), which gives the following system of linear equations for \(\mathbf{a}\)

\[
(M^T WM)\mathbf{a} = M^T Wb.
\]

If the matrix \(M^T WM\) is invertible, then

\[
\mathbf{a} = (M^T WM)^{-1}(M^T Wb).
\]
Let $A_x, A_y, A_{xx}, A_{xy}, A_{yy}$ are the rows of the matrix $A = (M^TWM)^{-1}$, then the unknowns $f_x, f_y, f_{xx}, f_{xy}, f_{yy}$ can be calculated explicitly as follows

\[
\begin{align*}
    f_x &= (b \otimes w) \cdot (MA^T_x) \\
    f_y &= (b \otimes w) \cdot (MA^T_y) \\
    f_{xx} &= (b \otimes w) \cdot (MA^T_{xx}) \\
    f_{xy} &= (b \otimes w) \cdot (MA^T_{xy}) \\
    f_{yy} &= (b \otimes w) \cdot (MA^T_{yy})
\end{align*}
\]

where $b = [f_1 - f, f_2 - f, \cdots, f_n - f]^T$, $w = (w_1, \cdots, w_n)^T$, $b \otimes w$ is the element-by-element product of two vectors $b$ and $w$, i.e. $b \otimes w = [b_i w_i]_{i=1..n}$, and $\cdot$ is inner product in $\mathbb{R}^n$.

### 2.3.1 Neighborhood searching

In the FPM, the spatial differential operators at each particle position are approximated from its surrounding clouds of particles with the help of the weighted moving least squares method. Hence in implementation, for each particle one needs to find its set of neighboring points.

The most common criterion for finding the neighborhoods is circular neighboring, i.e. a point $x' = (x', y')$ is a neighbor of $x = (x, y)$ if

\[d(x', x) = \|x' - x\|_2 = \sqrt{(x - x')^2 + (y - y)^2} \leq r\]

where $r > 0$.

Another criterion often used is the $k-$closest neighborhoods, where $k$ is a natural number.

In some situations, especially when the point clouds are irregular, we need to combine these criteria or to find a new algorithm for neighboring searching. This is the main task of the next chapter when we use the FPM for application in the lay-down process of the non-woven production and will be discussed in details.

### 2.3.2 Point cloud management

The basis of the computations in the FPM is a point cloud, which represents the flow field. The particles move with the fluid (see, e.g. section 2.3.3 in [2] the Lagrangian form of the adiabatic compressible Euler equations). One can not know in advance where the particles will move or how they will distribute. Hence, even if the particles are well distributed initially, the regularity will be destroyed after some times. They will in general cluster in some places and scatter in others. The former can result in an unnecessarily high local resolution and costs computational effort. The latter can spoil the accuracy and causes problems in the search of neighboring points. Therefore, we need to manage the point cloud in each computational step in order to maintain the good quality of the particle
distribution. Point cloud management is the main feature of the FPM. The method does not run stable without it (see [2]).

- **Removing Particles**

Particles which are too clustered are thinned by merging pairs of close by points into a single one. By doing so iteratively, large cluster can be thinned out. One prescribes a minimum distance between two points. If one finds two points closer than this distance, then both are removed and replaced by a new one, i.e. two particles are merged into a new single one. Let the two particles have the positions \( x_k \) and \( x_l \), and let the data \( u_k \) and \( u_l \) be associated with data vectors. The new particle is inserted in the center of mass of the two particles, and the data is linearly interpolated (e.g. see section 2.6 in [2] for solving Burger’s equation)

\[
\begin{align*}
  x &= \frac{x_k + x_l}{2} \\
  u &= \frac{u_k + u_l}{2}.
\end{align*}
\]

- **Adding Particles**

One needs to add new particles in some place where the particles are too scarce. Into each center of a hole, a new particle is inserted. The data at this new particle is interpolated from the neighboring particles by scattered data interpolation. For instance, in one dimension, if two particles \( x_k \) and \( x_l \) are too far away from each other (i.e. the distance between them is bigger than a prescribed maximum distance), a new particle is added with position \( \frac{x_k + x_l}{2} \) and the new function value is \( \frac{u_k + u_l}{2} \).

For more information on the point cloud management, readers can see ([2], [3]).

### 2.3.3 Weight functions

The first step of the MLS method is to assign weights to the points \( x_i \) via a *distance weight function* \( w_i \)

\[
w_i(x) = w(\|x - x_i\|_2).
\]

The distance weight function is chosen smooth and decaying with increasing distance. One distinguishes distance weight functions with respect to two features:

1. **Regular vs singular**
   
   A regular weight function satisfies
   
   \[
   w(0) = \lim_{d \to 0} w(d).
   \]
   
   whereas a singular one has a pole at zero distance, i.e.
   
   \[
   \lim_{d \to 0} w(d) = \infty.
   \]
If the distance weight function is regular, the function $w(\|x\|)$ must be twice differentiable in the origin, hence one needs to require $w'(0) = 0$.

2. **Compactly supported vs global**

A compactly supported distance weight function vanishes for large distances, i.e. there exists a $d_0 > 0$ such that

$$w(d) = 0 \quad \forall d > d_0$$

while a global one does not.

The following weight functions are often used in the implementation of the meshfree methods.

- **Gaussian weight function**

  $$w(x_i, x) = \begin{cases} 
  \exp \left( -\alpha \frac{\|x_i - x\|^2}{r^2} \right), & \text{if } \frac{\|x_i - x\|^2}{r^2} \leq 1 \\
  0 & \text{else} 
  \end{cases} \quad (2.10)$$

  where $\alpha, r > 0$.

- **Inverse weight function**

  $$w(x_i, x) = \|x_i - x\|^{-\alpha}. \quad (2.11)$$

  where $\alpha > 0$.

- **Cubic weight function**

  $$w(q) = \begin{cases} 
  \frac{2}{3} - 4q^2 + q^3 & \text{if } 0 \leq q \leq \frac{1}{2} \\
  \frac{4}{3} - 4q + 4q^2 - \frac{4}{3}q^3 & \text{if } \frac{1}{2} \leq q \leq 1 \\
  0 & \text{else.} 
  \end{cases} \quad (2.12)$$

For a more detail discussion on weigh functions, readers can see section 5.2.1 in [2].

2.4 **General FPM algorithm for approximation of derivatives**

We have the following algorithm to approximate the first and second order derivatives of a function $f$ using the FPM.
Algorithm 1 The general FPM algorithm for approximation of derivatives

**Input:** A finite set of points $X = \{x_i = (x_i, y_i) \in \mathbb{R}^2 : i = 1, \ldots, N\}$ and function values $f_1 = f(x_1), \ldots, f_N = f(x_N)$.

**Output:** The first and second order derivatives of $f$ at each point of $X$.

1: Choose a weight function $w : \mathbb{R}^2 \to \mathbb{R}_+$
2: for each $x \in X$ do
3: Find the neighborhoods $\text{Nbh}_x = \{x_1, \ldots, x_n\}$ of $x$.
4: Point cloud management
5: Calculate the matrix $M$ in (4.1), the weight matrix $W$, the vector $b$, and the matrix $M^T WM$.
6: Calculate the rows $A_x, A_y, A_{xx}, A_{xy}, A_{yy}$ of the matrix $(M^T WM)^{-1}$.
7: Calculate the derivatives from the formula

$$f_x(x) = (b \otimes w) \cdot (MA_x^T)$$
$$f_y(x) = (b \otimes w) \cdot (MA_y^T)$$
$$f_{xx}(x) = (b \otimes w) \cdot (MA_{xx}^T)$$
$$f_{xy}(x) = (b \otimes w) \cdot (MA_{xy}^T)$$
$$f_{yy}(x) = (b \otimes w) \cdot (MA_{yy}^T)$$

8: end for
Chapter 3

Mathematical model for the fibre lay-down process in the non-woven production

This chapter introduces a mathematical model for the lay-down process in the non-woven production and then considers the numerical solutions to the deterministic case of the model.

3.1 Introduction

Nonwoven fabrics are broadly defined as sheet or web structures bonded together by entangling fiber or filaments (and by perforating films) mechanically, thermally or chemically. They are flat, porous sheets that are made directly from separate fibers or from molten plastic or plastic film. Nonwoven fabrics provide specific functions such as absorbency, liquid repellency, resilience, stretch, softness, strength, flame retardancy, washability, cushioning, filtering, bacterial barrier and sterility. Nonwoven fabrics plays an importance role in our life. Many products such as hygiene products, filters (gasoline, oil and air filtration, liquid cartridge and bag filters, etc), geotextlies, etc are made from nonwoven fabrics.
In the melt-spinning process of nonwoven materials, hundreds of individual endless fibers being obtained by the continuous extrusion of a melted polymer are stretched and entangled by highly turbulent air flows to finally form a web on the conveyor belt. The quality of this web and the resulting nonwoven material, in terms of homogeneity and load capacity, depends essentially on the dynamics and the deposition of the fibers.

Mathematical modelling and simulation of the lay-down of flexible fibers onto a conveyor belt is of great importance for quality improvements, design of new machines and optimization of production processes. In [18], the authors presented a simplified stochastic model for the fiber lay-down process. The model is based on a stochastic differential equation taking into account the motion of the fiber under the influence of turbulence.

---

0 The picture Felt and Meltblown are respectively taken from http://en.wikipedia.org and http://www.neumag.oerlikontextile.com
the authors developed a new model in order to get smoother fiber trajectories which is more appropriate to the real physical process. For more information on the mathematical aspect of fibre dynamics, readers can see ([9],[10]).

3.2 Stochastic model for the fibre lay-down process

We will summarize briefly the model in ([18]) because the point cloud that the thesis is dealing with arises from this model.

Consider a single slender elastic inextensible fiber in an isotropic lay-down process, the fiber on the non-moving conveyor belt can be described by a parameterized curve (see picture 3.3):

\[ \xi : \mathbb{R}^+_0 \rightarrow \mathbb{R}^2, \quad t \mapsto \xi(t) = \left( \begin{array}{c} \xi_1(t) \\ \xi_2(t) \end{array} \right). \]

The inextensibility condition means that the length of fibre material laid down on the belt per interval time is constant, and it holds if

\[ \| \partial_t \xi \| = c \quad \forall t \in \mathbb{R} \quad \text{and constant} \quad c \in \mathbb{R}. \]  \hspace{1cm} (3.1)
Without loss of generality we can take $c = 1$, i.e.

$$
\|\partial_t \xi\| = 1.
$$

(3.2)

Indeed, let us calculate the length of the curve $\xi$ in a unit interval of time

$$
l = \int_0^1 \sqrt{\xi'_1(t)^2 + \xi'_2(t)^2} dt = \int_0^1 \|\partial_t \xi\| dt = \int_0^1 dt = 1.
$$

Let $\alpha = \alpha(t)$ denote the normalized tangent on the fiber. Then, the condition (3.2) is satisfied for

$$
d\xi = \left(\frac{d\xi_1}{d\xi_2}\right) dt = \left(\begin{array}{c}
\cos \alpha \\
\sin \alpha
\end{array}\right) dt = \tau(\alpha) dt
$$

where $\tau(\alpha) = (\cos \alpha, \sin \alpha)^T$.

Since a curved fiber tends back to its starting point, the deterministic motion of the spinning process around its starting point is modelled with the assumption that the change of the angle $\alpha$ is proportional to $\xi \cdot \tau^\perp(\alpha) = (-\sin \alpha, \cos \alpha)^T$ and is dependent on the length $\|\xi\|$. The amplitude of this drive is prescribed by a continuously differentiable function $b : \mathbb{R}_0^+ \rightarrow \mathbb{R}$ with $b(r) > 0$ for $r > 0$ and $b'(r) \geq 0$ for $r \geq 0$.

$$
d\alpha = -b(\|\xi\|) \frac{\xi}{\|\xi\|} \cdot \tau^\perp(\alpha) dt.
$$

All other forces on the fibre and their resulting turbulence are modelled as a one-dimensional Wiener process $W_t$ with amplitude $A$, thus

$$
d\alpha = -b(\|\xi\|) \frac{\xi}{\|\xi\|} \cdot \tau^\perp(\alpha) dt + A dW_t.
$$

Hence, the fibre lay-down process is modelled by the following stochastic differential system

$$
\begin{align*}
    d\xi &= \tau(\alpha) dt \\
    d\alpha &= -b(\|\xi\|) \frac{\xi}{\|\xi\|} \cdot \tau^\perp(\alpha) dt + A dW_t.
\end{align*}
$$

(3.3)

Let $r_0$ be the radius such that $b(r_0) = \frac{1}{r_0}$. The system can be non-dimensionalized by setting $\tilde{r} = r_0 b(r_0 \tilde{r})$ and $\tilde{A} = \sqrt{r_0 A}$. For the sake of simplicity, we will omit the tilde.

We introduce a polar coordinates $(\xi, r, \phi) \in \mathbb{R}^+ \times [0, 2\pi] \times [0, 2\pi]$ with

$$
\begin{align*}
    \xi &= (r \cos \phi, r \sin \phi)^T \\
    r &= \|\phi\| \\
    \beta &= \alpha - \phi
\end{align*}
$$
(see the picture 3.3). Then the system (3.3) can be rewritten as

\[
\begin{align*}
    dr &= \cos \beta dt \\
    d\beta &= \left( b(r) - \frac{1}{r} \right) \sin \beta dt + A dW_t \\
    d\phi &= \frac{\sin \beta}{r} dt.
\end{align*}
\]

(3.4)

Due to symmetry, we can restrict to \( \beta \in [0, \pi] \). Since the equation for \( \phi \) is separated from the remaining variables, this transformation reduces the dimension of the problem.

### 3.3 The deterministic ODEs system

In this section, we consider the numerical solution in the phase plane \((r, \beta)\) for the deterministic case (i.e., \( A = 0 \)) of the system (3.4) where \( b \) is taken to be linear since the point cloud we are interested in arises from the deterministic system. For the sake of convenience, we use the \((x, y)\) notation instead of \((r, \beta)\).

\[
\begin{align*}
    dx &= \cos y \\
    dy &= \left( b(x) - \frac{1}{x} \right) \cdot \sin y
\end{align*}
\]

(3.5) (3.6)

where \( b : \mathbb{R} \to \mathbb{R} \) is a fixed function \( b(x) = \gamma_a \cdot x + \gamma_b \) with parameters \( \gamma_a, \gamma_b \). However, w.o.l.g. we can take \( \gamma_a = 1, \gamma_b = 0 \).

Consider the following energy function ([18]):

\[
G(x, y) = \exp \left( - \int_1^x b(x') dx' \right) \cdot x \cdot \sin y.
\]

Then \( G(x, y) \) is constant along the curve defined by the ODEs system above since:

\[
\begin{align*}
    \frac{dG}{dt} &= \frac{\partial G}{\partial x} \cdot \frac{dx}{dt} + \frac{\partial G}{\partial y} \cdot \frac{dy}{dt} \\
    &= \exp\left[ -\left( x^2 - 1 \right)/2 \right] \cdot (1 - x^2) \cdot \sin y \cdot \frac{dx}{dt} + \exp\left[ -\left( x^2 - 1 \right)/2 \right] \cdot x \cdot \cos y \cdot \frac{dy}{dt} \\
    &= 0
\end{align*}
\]

The initial condition \((x_0, y_0)\) has to be chosen in a way, such that:

\[
G(x_0, y_0) = \exp \left( - \int_1^{x_0} b(x') dx' \right) \cdot x_0 \cdot \sin y_0 = G_0,
\]

with \( G_0 \in [0, 1] \) is a chosen initial energy.

We have

\[
\int_1^{x_0} b(x') dx' = \left[ \frac{\gamma_a}{2} x^2 + \gamma_b x \right] \bigg|_1^{x_0} = (x_0 - 1) \left[ \frac{\gamma_a}{2} (x_0 + 1) + \gamma_b \right].
\]
So

\[ G_0 = \exp \left[ -(x_0 - 1)\left(\frac{\gamma_a}{2}(x_0 + 1) + \gamma_b\right) \right] \cdot x_0 \cdot \sin y_0. \]

When \( \gamma_a = 1, \gamma_b = 0 \), we have:

\[ \exp \left( -\frac{x_0^2 - 1}{2} \right) \cdot x_0 \cdot \sin y_0 = G_0. \] (3.7)

If \( G_0 = 1 \) then \( x_0 = 1, y_0 = \frac{\pi}{2} \), the above ODEs have unique solution \( x = 1, y = \frac{\pi}{2} \).

If \( G_0 = 0 \) then \( y_0 = 0 \), hence \( y = y_0 = 0, x = x + x_0 \).

If \( G_0 \neq 0 \) we choose \( x_0 = G_0 \), then

\[ \sin y_0 = \exp \left( \frac{x_0^2 - 1}{2} \right). \]

The function \( x_0 \mapsto \exp \left( \frac{x_0^2 - 1}{2} \right) \) is monotone increasing over \([0, 1] \) and takes value in \([0, 1]\), so we have \( y_0 = \text{argsin} \exp \left( \frac{x_0^2 - 1}{2} \right) \).

![Figure 3.4: The exponential function](image)
Hence we get the following initial value ODEs system

\[
\begin{align*}
\frac{dx}{dt} &= \cos y \\
\frac{dy}{dt} &= \left(x - \frac{1}{x}\right) \cdot \sin y \\
x_0 &= G_0, \quad y_0 = \arcsin \exp \left(\frac{x_0^2 - 1}{2}\right)
\end{align*}
\]

where \(G_0 \in [0, 1]\) is given. This system of ODEs can be solved by ODEs solver ode45 in MATLAB.

**Remark 1.** From the ODEs (3.8) we have

\[
\frac{\cos y}{\sin y} \cdot dy = \left(x - \frac{1}{x}\right) \cdot dx
\]

\[\Rightarrow \ln |\sin y| = \frac{x^2}{2} - \ln x + C\]

\[\Rightarrow |\sin y| = \frac{C}{x} \exp \left(\frac{x^2}{2}\right)\]

\[\Rightarrow \sin y = \frac{C}{x} \exp \left(\frac{x^2}{2}\right) \quad \text{since} \quad y \in [0, \pi].\]

From the initial condition (3.7), we get \(C = \frac{G_0}{\sqrt{e}}\). Hence, we get the implicit solution for the ODEs as follows

\[y = \arcsin \left(\frac{G_0}{x} \exp \left(\frac{x^2 - 1}{2}\right)\right)\]

or

\[y = \pi - \arcsin \left(\frac{G_0}{x} \exp \left(\frac{x^2 - 1}{2}\right)\right)\]
Chapter 4

Application of the FPM in the fibre lay-down process

This chapter introduces the FPM to approximate derivatives of a function with the point cloud and data values arise from the model in the previous chapter. Generating the point cloud, selecting and filtering the neighbors as well as the algorithm of the FPM in the fibre lay-down process are introduced subsequently. The last section of this chapter is for error estimations.

4.1 Generating the point cloud

The point cloud are the numerical solutions of the ODEs (3.8) in the phase plane \((x, y)\) with a given sequence of energies \(\{G_0^1, \ldots G_0^n\}\) of \(G_0 \in [0, 1]\).

**Algorithm 2** point cloud generating

**Input:** \(\{G_0^1, \ldots G_0^n\} \subseteq [0, 1]\)

**Output:** the point cloud \(X\)

1: for i=1...n do
2:  Solving the ODEs (3.8) in the phase plane \((x, y)\) with energy \(G_0^i\):
3:  \(X_i = \{(x, y)| (x, y) \text{ are solution of (3.8)}\}\)
4: end for
5: \(X = \bigcup_{i=1}^n X_i\)

**Example 1.** The following pictures illustrate the point cloud with \(G_0 \in \text{linespace}(0.05, 0.95, 40)\) (the left one) and \(G_0 \in \text{linespace}(0.05, 0.95, 100)\) (the right one) where \(\text{linespace}(a, b, n)\) is to create \(n\) linearly vector in the interval \([a, b]\).
4.2 Neighborhood Selecting

Searching neighbors of particles is the most important and time consuming part of the finite pointset method ([3]). After the initialization, particles are numbered from 1 to $N$ and for each index $i$ one can directly access to the position $x_i$. In this thesis, the particles are fixed and hence we can not apply the techniques of removing or adding new points mentioned in the section 2.3.2 as in the general FPM applications. In other words, we have less control on the points. The attribute of the problem we are considering is that the better we solve the ODEs, the closer point cloud we will get. The more initial energies we compute, the closer the trajectories we will get in certain areas. The points in these areas are too close together and might cause problems in the numerical computation due to the rounding error and machine precision. Hence for each particle, it is not obvious to find out which particles among the others should be taken as its neighbors. The main task of this section is to find out criteria to select neighborhoods for each particle.

Given a point cloud $\mathbb{X} = \{x_1, \ldots, x_N\}$ and $x \in \mathbb{X}$. We need to find neighborhoods
\( \text{Nbh}_x = \{x_1, ..., x_n\} \) of \( x \) with \( n \leq N \)

Since the FPM uses the Taylor expansion, each \( x_i \) should be not too far from \( x \). As mentioned above, it also should not too close to \( x \). Moreover, we want that the matrix

\[
M = \begin{pmatrix}
dx_1 & dy_1 & \frac{1}{2}dx_1^2 & dx_1dy_1 & \frac{1}{2}dy_1^2 \\
dx_2 & dy_2 & \frac{1}{2}dx_2^2 & dx_2dy_2 & \frac{1}{2}dy_2^2 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
dx_n & dy_n & \frac{1}{2}dx_n^2 & dx_ndy_n & \frac{1}{2}dy_n^2 
\end{pmatrix}
\]

and \( A = M^T W M \) are "as good as possible" e.g. not (numerically) close to singular. Therefore we have the following criteria to select the neighborhood \( \text{Nbh}_x \) of \( x \):

- \( x_i \) is not too far from \( x \), \( \forall i = 1...n \).
- \( x_i \) is not too close to \( x \), \( \forall i = 1...n \).
- \( \text{Nbh}_x \) are distributed around \( x \).
- There are enough neighborhoods: \( |\text{Nbh}_x| \geq 5 \).

In the sequel, we are going to introduce algorithms to select \( \text{Nbh}_x \). First we will consider each algorithm separately and then combine them together.

1. **Circular neighborhoods**

A point \( x' \) is a neighbor of \( x \) if

\[
d(x', x) = \|x' - x\| = \sqrt{(x - x')^2 + (y - y')^2} \leq r
\]

where \( r > 0 \).

**Algorithm 3** Circular neighborhood

**Input:** A list of points \( X \), a point \( x \in X \), and a radius \( r > 0 \)

**Output:** The list \( \text{Nbh}_x \) of circular neighborhoods of \( x \)

1: \( \text{Nbh}_x = \emptyset \)
2: for \( x' \in X \) do
3: if \( \|x - x'\| \leq r \) then
4: \( \text{Nbh}_x = \text{Nbh}_x \cup \{x'\} \)
5: end if
6: end for

The radius \( r \) may be prescribed and fixed for all points or depends on each point. However it is not obvious what \( r \) should be since the distribution of particles is
irregular. They may cluster in some areas but scatter in others. If $r$ is small, it can happen that there is not enough neighborhoods for some points. In contrast, if $r$ is big, we can find many neighborhoods for each point.

**Example 2.** In this example, number of energies (curves): $n = 40, G = \text{linspace}(0.05, 0.95, n)$, $r = 0.1$ where linspace is to create linearly vector in MATLAB. The index of the point is 125 with coordinate $\mathbf{x} = (2.8911, 0.7547)$ (the circle one in the figure below). $\mathbf{x}$ has only 3 circular neighborhoods (the star ones).

![Figure 4.2: A point has not enough neighborhoods.](image)

**Example 3.** In this example, $n = 80, G = \text{linspace}(0.05, 0.95, n), r = 0.1$. The index of the point is 3225 with coordinate $\mathbf{x} = (0.3119, 0.3661)$ (the circle one in the figure below). $\mathbf{x}$ has 185 circular neighborhoods (the star ones).
2. **k-closest neighborhoods:**

As illustrated in two examples above, we have no control on the number of neighborhoods of a particle when using the circular criterion. It can be not enough or too many. The former does not satisfy the condition of the FPM, while the latter costs time. In order to balance these two situations, we can select the neighborhoods of a particle $x$ as its *k-closest points to* $x$, where $k \geq 5$ is a natural number. Given a point $x$ and a natural number $k$, to find the $k$-closest points to $x$, we compute the distances from all the other points to $x$, sort them in ascending order and then select the first $k$ points. We will use a sorting algorithm available in MATLAB.

Figure 4.3: A point has too many neighborhoods
Algorithm 4 \( k \)-closest neighborhoods

**Input:** A list of points \( \mathbb{X} \), a point \( x \in \mathbb{X} \), and a natural number \( k > 0 \)

**Output:** The list \( \text{Nbh}_x \) of neighborhoods of \( x \)

1. Calculate the distances from \( x \) to all others points
2. \( d = \{ \| x' - x \|, x' \in \mathbb{X} \} \)
3. Sort \( d \) in ascending order
4. \( \text{Nbh}_x \) are the first \( k \) points from the sorted list.

Using this algorithm, we know exactly how many neighbors a point has. However there are a lot of points whose closest neighborhoods lie (nearly) in a line or a circle. The matrices \( M \) and \( M^T M \) in these situations are close to singular.

**Example 4.** In this example, number of energies (curves): \( n = 40 \), \( G = \text{linspace}(0.05,0.95,n) \), \( k = 5 \) where \( \text{linspace} \) is to create linearly vector in MATLAB. The index of the point is 125 with coordinate \( x = (2.8911,0.7547) \) (the circle one in the figure below). 5 closest neighborhoods of \( x \) are in one line. The matrices \( M \), \( M^T M \) and their condition numbers are

\[
M = \begin{pmatrix}
-0.018967280495252 & -0.043273786050618 & 0.00017987864693 & 0.000820786038114 & 0.000936310279577 \\
0.018683787956839 & 0.047052169629502 & 0.00017541966208 & 0.000879112760267 & 0.001106953333422 \\
-0.038658518898584 & -0.084136349788558 & 0.000476384525561 & 0.003250679548961 & 0.003539462677871 \\
0.036455143559923 & 0.096751795466288 & 0.000664887459987 & 0.003527100593404 & 0.004680454962975 \\
-0.058904190556268 & -0.122658887930014 & 0.001736619396474 & 0.007228802250156 & 0.007522601394114
\end{pmatrix}
\]

\[
M^T M = \begin{pmatrix}
0.007003786997959 & 0.015706481190901 & -0.000107109362522 & -0.000422178345578 & -0.000406529099069 \\
0.015796481190901 & 0.035571565295919 & -0.000211089172789 & -0.000813078198137 & -0.000756102104704 \\
-0.000107109362522 & -0.000211089172789 & 0.0000040772723531 & 0.000016724672881 & 0.000019177366766 \\
-0.000422178345578 & -0.000107109362522 & 0.000016724672881 & 0.000076790467064 & 0.000084135139509 \\
-0.000406529099069 & -0.000516481190901 & 0.000016724672881 & 0.000076790467064 & 0.000093126009065
\end{pmatrix}
\]

\[\text{cond}(M) = 4.2181e+006, \quad \text{cond}(M^T M) = 1.7781e+013\]
3. **Combination**: As illustrated in the examples (2,3,4) circular or $k$-closest neighboring criteria has both advantage and disadvantage. To make use of their good properties, we will combine them together. Consider a radius $r > 0$, a natural number $k$, for each particle $\mathbf{x}$, we divide the domain around it into 6 equal parts. Let $n_i$, $(i = 1, \cdots, 6)$, be the number of particles in $i$-th part having distance to $\mathbf{x}$ smaller than $r$. Then in part $i$, we select $\min\{n_i, k\}$ closest points to $\mathbf{x}$.

**Algorithm 5** Combination algorithm

**Input:** A list of points $\mathbb{X}$, a point $\mathbf{x} \in \mathbb{X}$, a radius $r$, and a natural number $k$

**Output:** The list $\text{Nbh}_x$ of neighborhoods of $\mathbf{x}$

1: Divide the domain around $\mathbf{x}$ into 6 equal parts
2: In each part, create a list of points having distance to $\mathbf{x}$ smaller than $r$
3: In each list from previous step, select $\min\{n_i, k\}$ closest points to $\mathbf{x}$
4: $\text{Nbh}_x$ is the union of those points in the previous step.

Figure 4.4: neighborhoods are (nearly) in one line
This algorithm makes sure that the neighborhoods are not too far and distribute around \(x\). However, we need to define a local coordinate at each point.

### 4.3 Filtering the points

Consider the matrix

\[
M = \begin{pmatrix}
  dx_1 & dy_1 & \frac{1}{2}dx_1^2 & dx_1dy_1 & \frac{1}{2}dy_1^2 \\
  dx_2 & dy_2 & \frac{1}{2}dx_2^2 & dx_2dy_2 & \frac{1}{2}dy_2^2 \\
  \vdots & \vdots & \vdots & \vdots & \vdots \\
  dx_n & dy_n & \frac{1}{2}dx_n^2 & dx_ndy_n & \frac{1}{2}dy_n^2
\end{pmatrix}
\]

In some areas, the particles are too close together. Extremely small numbers can cause computational error due to machine precision. In addition, if two neighbors are too close together, then the corresponding rows in the matrix \(M\) are almost the same. Hence, it is close to singular (does not have full rank). To avoid these situations, we need to filter the neighbors. To do that we prescribe a positive distance \(\varepsilon\). For each particle \(x\), we create its list of potential neighbors. Then if we detect two particles whose distance closer than \(\varepsilon\) then we select only one to be neighbor of \(x\).

**Algorithm 6** Filtering algorithm

**Input:** A list of points \(\mathbb{X}\), a point \(x \in \mathbb{X}\), a positive number \(\varepsilon\)

**Output:** The list \(\text{Nbh}_x\) such that the distance from any point to \(x\) or between any two points is bigger than \(\varepsilon\)

1. \(\text{Nbh}_x = \emptyset\)
2. for \(x' \in \mathbb{X}\) do
3. \(\text{if } \|x' - x\| > \varepsilon \text{ and } \min\{\|x' - y\|, y \in \text{Nbh}_x\} > \varepsilon \text{ then}\)
4. \(\text{Nbh}_x = \text{Nbh}_x \cup \{x'\}\)
5. \(\text{end if}\)
6. \(\text{end for}\)

**Example 5.** Data is as in example 3 but we use the combination and filtering algorithm with \(k = 3\). The point now has only 12 neighborhoods distributed around it.
4.4 Choosing weight functions

In order to restrict the number of points we use the Gaussian weight function introduced in 2.3.3 since it has a small compact support.

\[ w(x, x') = \begin{cases} 
\exp \left( -\alpha \frac{||x-x'||^2}{r^2} \right), & \text{if } \frac{||x-x'||^2}{r^2} \leq 1 \\
0, & \text{else.}
\end{cases} \quad (4.1) \]

where \(\alpha, r > 0\).

The parameter \(\alpha\) is often chosen in \([2, 6]\) (see \([3],[15]\)). In this thesis, we use \(\alpha = 5\).

4.5 Algorithm

We are ready to introduce the following FPM algorithm for approximating the first and second order derivatives with the point cloud we are considering.
Algorithm 7 the FPM for approximation of derivatives for the lay-down process application

Input: A finite set of points \( X = \{ x_1, \ldots, x_N \} \) and values of a function \( f_i = f(x_i) \) at these points.

Output: The first and second order derivatives of \( f \) at each point of \( X \).

1: for \( x \in X \) do
2:   • Find the neighborhoods \( \text{Nbh}_x \) of \( x \).
3:      Divide the domain around \( x \) into 6 equal parts \( S_j \) \( (j = 1, \ldots, 6) \)
4:      for \( j = 1 \ldots 6 \) do
5:         \( S_j = \emptyset \)
6:         \( n_j = 0 \)
7:         for \( x' \in X \) do
8:            if \( \varphi([x-x', y-y']^T, [1 0]^T) \in \left[ \frac{2\pi}{3}, \frac{(j+1)\pi}{3} \right] \) then
9:               \( S_j = S_j \cup \{ x' \} \)
10:              \( n_j = n_j + 1 \)
11:         end if
12:      end for
13: end for
14: In each part, select \( s_j = \min\{ k, n_j \} \) points closest to \( x \).
15: for \( j = 1 \ldots 6 \) do
16:   \( s_j = \min\{ k, n_j \} \)
17:   \( \text{tNbh} = \emptyset \). Note: \( \text{tNbh} \) is the temporary neighborhoods of \( x \), which need to be filtered later.
18:      if \( n_j > 0 \) then
19:         \( \text{vd} = \{ \| q - p \|, q \in S_j \} \)
20:         \( [\text{svd}, \text{idx}] = \text{sort}(\text{vd}). \)
21:            Note: \( \text{sort(vd)} \) is a sorting algorithm in MATLAB, which sort the vector \( \text{vd} \) in ascending order. The sorted vector is \( \text{svd} \) and \( \text{idx} \) is the corresponding indices in the original vector.
22:         \( \text{tNbh} = \text{tNbh} \cup \{ y \in S_j \mid \text{the index of } y \text{ in } S_j \in \text{idx}(1:s_j) \} \).
23:      end if
24: end for
25: • Filtering
26: \( \text{Nbh} = \emptyset \)
27: for \( y \in \text{tNbh} \) do
28:    if \( \min\{ \| y - y' \|, y' \in \text{Nbh} \} \geq \varepsilon \) then
29:       \( \text{Nbh} = \text{Nbh} \cup \{ y \} \). Note: Filtering
30:    end if
31: end for
32: • Calculate the matrix \( M, A = M^TWM \) and the rows \( A_x, A_y, A_{xx}, A_{xy}, A_{yy} \) of \( A^{-1} \)
33: \( f_x(x) = (b \otimes w) \cdot (MA_x^T) \)
34: \( f_y(x) = (b \otimes w) \cdot (MA_y^T) \)
35: \( f_{xx}(x) = (b \otimes w) \cdot (MA_{xx}^T) \)
36: end for
4.6 Error estimations

Let $f$ be a given function and $u_x, u_y, u_{xx}, u_{xy}, u_{yy}$ be the approximations of derivatives of $f$ calculated from the algorithm above. We will numerically compute the $L^2$--norm of the errors:

\[
\|e_x\|_2 = \sqrt{\iint_{\Omega} |u_x - f_x|^2 \, dx \, dy} \\
\|e_y\|_2 = \sqrt{\iint_{\Omega} |u_y - f_y|^2 \, dx \, dy} \\
\|e_{xx}\|_2 = \sqrt{\iint_{\Omega} |u_{xx} - f_{xx}|^2 \, dx \, dy} \\
\|e_{xy}\|_2 = \sqrt{\iint_{\Omega} |u_{xy} - f_{xy}|^2 \, dx \, dy} \\
\|e_{yy}\|_2 = \sqrt{\iint_{\Omega} |u_{yy} - f_{yy}|^2 \, dx \, dy}
\]

where $\Omega \subseteq \mathbb{R}^2$ is the underlying domain.

Let $X = \{x_1 = (x_1, y_1), \ldots, x_N = (x_N, y_N)\}$ be a set of points and $g(x, y)$ be a function in $\Omega$. We will numerically compute the integral

\[
I = \iint_{\Omega} g(x, y) \, dx \, dy
\]

from the values of $f$ at the points $p_i, i = 1, \ldots, N$. Let $|\Omega|$ be the area of $\Omega$. We have

\[
I = \iint_{\Omega} g(x, y) \, dx \, dy = |\Omega| \iint_{\Omega} g(x, y) \frac{1}{|\Omega|} \, dx \, dy = |\Omega| E(g(U))
\]

where $U$ is an uniform random variable in $\Omega$. Assume $U_1, \ldots, U_n$ are $n$ samples of $U$. By the law of large number $\frac{g(U_1)+\ldots+g(U_n)}{n} \to E(g(U))$ as $n \to \infty$.

Hence, similarly to the Monte Carlo methods ([1]), we will approximate $I$ by

\[
I \approx \frac{|\Omega|}{N} \sum_{i=1}^{N} g(x_i, y_i).
\]

To calculate $|\Omega|$, we apply again the above method for 1D. Assume that $\Omega$ is formed by the graph of two functions $h_1(x)$ and $h_2(x)$ with $0 \leq h_2(x) \leq h_1(x)$ and $x$ from $a$ to $b$. Let
Let \( x_1, \ldots, x_n \) be \( n \) points in \([a, b]\) then
\[
|\Omega| = \int_a^b (h_1(x) - h_2(x)) \, dx \approx \frac{b-a}{n} \sum_{i=1}^n (h_1(x_i) - h_2(x_i)).
\]

The **Koksma Hlawka Inequality** theorem below gives an estimation of error for the numerical integration. Before introducing it, we need some definitions.

**Definition 2.** If \( f \) is sufficiently differentiable then the variation of \( f \) on \([0,1]^d\) in the sense of Hardy and Krause is
\[
V(f) = \sum_{k=1}^{s} \sum_{1 \leq i_1 \leq \cdots \leq i_k \leq s} V^{(k)}(f; i_1, \ldots, i_k),
\]
where
\[
V^{(k)}(f; i_1, \ldots, i_k) = \int_0^1 \cdots \int_0^1 \left| \frac{\partial^k f}{\partial x_{i_1} \cdots \partial x_{i_k}} \right| \, dx_{i_1} \cdots dx_{i_k}.
\]

**Definition 3.** Suppose we construct
\[
\hat{F}_N(x) = \frac{1}{N} \sum_{n=1}^N I(x_n \leq x),
\]
the empirical cumulative distribution function of the points \( x_1, \ldots, x_N \), and compare it with
\[
F(x) = F(x_1, \ldots, x_d) = \min\{1, x_1 \cdots x_d\} \quad \text{if all } x_i \geq 0
\]
the theoretical uniform distribution on \([0,1]^d\). The star discrepancy \( D^*_n(x_1, \ldots, x_n) \) of the sequence \( x_1, \ldots, x_n \) is
\[
D^*_n(x_1, \ldots, x_n) = \sup_x |\hat{F}_N(x) - F(x)| = \sup_B \left\| \frac{\# \text{ of points in } B}{N} - \lambda(B) \right\|
\]
where the supremum is taken over all rectangles \( B \) of the form \([0, x_1] \times \cdots \times [0, x_d]\) and where \( \lambda(B) \) is the Lebesgue measure of \( B \) in \( \mathbb{R}^d \).

**Theorem 4.6.1. (Koksma-Hlawka Inequality)** (see [1])

Let \( f \) have bounded variation \( V(f) \) on \([0,1]^d\) in the sense of Hardy and Krause. Then for any \( x_1, \ldots, x_n \) in \([0,1]^d\)
\[
\frac{1}{n} \sum_{i=1}^n f(x_i) - \int_{[0,1]^d} f(x) \, dx \leq V(f)D^*_n(x_1, \ldots, x_n)
\]

where \( D^*_n(x_1, \ldots, x_n) \) is the star discrepancy of the sequence \( x_1, \ldots, x_n \). Moreover, the Koksma-Hlawka inequality is sharp in the following sense: For any point set \( x_1, \ldots, x_n \) in \([0,1]^d\) and any \( \varepsilon > 0 \), there is a function \( f \) with bounded variation and \( V(f) = 1 \) such that
\[
\frac{1}{n} \sum_{i=1}^n f(x_i) - \int_{[0,1]^d} f(x) \, dx > D^*_n(x_1, \ldots, x_n) - \varepsilon
\]
The Koksma-Hlawka inequality can be generalized for integrals over arbitrary regions (see Niederreiter [11]). Hence, according to the Koksma-Hlawka Inequality, the quality of a numerical integration rule depends on the star discrepancy $D^*_n(x_1, \ldots, x_n)$ of $x_1, \ldots, x_n$. However, it is computationally hard to find the exact value of the discrepancy of large point sets and it is out of the scope of this thesis.

In our problem, we generate the point cloud as introduced in section 4.1. We fix the innermost curve (corresponding to the smallest energy) and the outermost one (corresponding to the highest energy). Then, the particles are fulfilled inside.

**Example 6.** In this example, $\Omega$ is formed by the outermost and innermost curves with $G_0 = 0.05$ and $G_0 = 0.95$ respectively.

The domain is symmetric via the line $y = \frac{\pi}{2}$, so we can calculate its area by calculating the area of the upper part, which is formed by the graphs of two functions (Remark1), then multiply by 2.

Let $\Gamma_1$ and $\Gamma_2$ be the outermost and innermost curves. Assume that their $x$-coordinates

![Figure 4.6: the domain formed by the inner and outer curves](image-url)
respectively are in \([x_{1\text{min}}, x_{1\text{max}}]\) and \([x_{2\text{min}}, x_{2\text{max}}]\). Define \(A = \{x = (x, y) \in \Gamma_1 : y \geq \frac{\pi}{2}\}\) and \(B = \{x = (x, y) \in \Gamma_2 : y \geq \frac{\pi}{2}\}\), then

\[
|\Omega| \approx 2 \left[ \frac{x_{1\text{max}} - x_{1\text{min}}}{|A|} \sum_{(x,y) \in A} (y - \frac{\pi}{2}) - \frac{x_{2\text{max}} - x_{2\text{min}}}{|B|} \sum_{(x,y) \in B} (y - \frac{\pi}{2}) \right].
\]

Having an approximation for \(|\Omega|\), we can approximate the \(L^2\)–errors as following:

\[
\|e_x\|_2 \approx \sqrt{\frac{|\Omega|}{N} \sum_{i=1}^{N} |u_{x_i} - f_{x_i}|^2}
\]

\[
\|e_y\|_2 \approx \sqrt{\frac{|\Omega|}{N} \sum_{i=1}^{N} |u_{y_i} - f_{y_i}|^2}
\]

\[
\|e_{xx}\|_2 \approx \sqrt{\frac{|\Omega|}{N} \sum_{i=1}^{N} |u_{xx_i} - f_{xx_i}|^2}
\]

\[
\|e_{xy}\|_2 \approx \sqrt{\frac{|\Omega|}{N} \sum_{i=1}^{N} |u_{xy_i} - f_{xy_i}|^2}
\]

\[
\|e_{yy}\|_2 \approx \sqrt{\frac{|\Omega|}{N} \sum_{i=1}^{N} |u_{yy_i} - f_{yy_i}|^2}
\]
Chapter 5

Numerical Implementation and Examples

This chapter gives numerical results for some testing functions. In the first section, we compare the results between regular and irregular grids. In the latter case, we also plot the errors when we do not take some outer curves into account. A summary of the numerical results will be given in the last section.

5.1 Testing Examples

The algorithm presented in the previous chapter can be applied to arbitrary point clouds. In this section, regular grid is the usual mesh of $[0,1]^2$ with spacing $\delta_x$ and irregular grid is the point cloud generated as in the previous chapter.
In the sequel, we will test the algorithm with some given functions. In each example, we do for both regular and irregular grids. For the latter case, we also plot the errors when we do not take some outer curves into account since we want to know where the errors arise.

Parameter explanation:

\( \delta_x \) = mesh spacing in the regular grids,
\( N \) = number of particles,
\( n \) = number of curves (i.e. number of energies used) in the irregular grid,
\( \varepsilon \) = filtering parameter (i.e. distance between any two points in a neighborhood set is bigger than \( \varepsilon \)),
\( k \) = number of closest point used in \( k \)-closest neighboring criterion,
\( r \) = radius used in circular neighboring criterion,
\( \gamma \) = parameter in the Gaussian weight functions.

In the code, we use \( G_0 \in \text{linspace}(0.05, 1, n) \), \( \varepsilon = 0.0001 \), \( \gamma = 5 \) and \( r = 5\delta_x \) (for regular grids). Other parameters are shown in the tables in each example.

**Example 7.** \( f(x, y) = x^2y \). The exact derivatives are \( f_x = 2xy, f_y = x^2, f_{xx} = 2y, f_{xy} = 2x, f_{yy} = 0 \)
Figure 5.2: Testing function: $f(x, y) = x^2 y$

- Regular grid

Table 5.1: $L^2$-norm of errors for regular grid. Testing function: $f(x, y) = x^2 y$

<table>
<thead>
<tr>
<th>$\delta_x$</th>
<th>$N$</th>
<th>$|e_x|_2$</th>
<th>$|e_y|_2$</th>
<th>$|e_{x,x}|_2$</th>
<th>$|e_{x,y}|_2$</th>
<th>$|e_{y,y}|_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>9</td>
<td>0.364347488937676</td>
<td>0.271886209429157</td>
<td>0.948580683147287</td>
<td>0.752447870848261</td>
<td>1.085965148874668</td>
</tr>
<tr>
<td>0.25</td>
<td>25</td>
<td>0.043860389308567</td>
<td>0.032569417039691</td>
<td>0.257113917694228</td>
<td>0.327811933108381</td>
<td>0.073186159663691</td>
</tr>
<tr>
<td>0.2</td>
<td>36</td>
<td>0.029446808186493</td>
<td>0.029454344369000</td>
<td>0.193772967718159</td>
<td>0.270663937010148</td>
<td>0.053954601250518</td>
</tr>
<tr>
<td>0.125</td>
<td>81</td>
<td>0.01117553206383</td>
<td>0.021284777296413</td>
<td>0.112342776899532</td>
<td>0.169758229437030</td>
<td>0.056094071911812</td>
</tr>
<tr>
<td>0.1</td>
<td>121</td>
<td>0.00649389403111</td>
<td>0.016205079008745</td>
<td>0.089274876638655</td>
<td>0.129149493258857</td>
<td>0.03586201839180</td>
</tr>
<tr>
<td>0.05</td>
<td>441</td>
<td>0.001338280548014</td>
<td>0.004784327565051</td>
<td>0.031574203252700</td>
<td>0.057793261682472</td>
<td>0.026642907462905</td>
</tr>
<tr>
<td>0.04</td>
<td>676</td>
<td>0.000780803707548</td>
<td>0.00392538193952</td>
<td>0.02302630442863</td>
<td>0.043700932985916</td>
<td>0.020399286626996</td>
</tr>
<tr>
<td>0.025</td>
<td>1681</td>
<td>0.000246573485077</td>
<td>0.00120436088808</td>
<td>0.01836497615831</td>
<td>0.02628532200719</td>
<td>0.009542018236713</td>
</tr>
<tr>
<td>0.02</td>
<td>2601</td>
<td>0.00014708894113</td>
<td>0.00078244843590</td>
<td>0.008763798227706</td>
<td>0.019990828680436</td>
<td>0.00801697463764</td>
</tr>
<tr>
<td>0.0125</td>
<td>6561</td>
<td>0.000044741182889</td>
<td>0.00039316391672</td>
<td>0.004342929735140</td>
<td>0.012130742661835</td>
<td>0.00365189975072</td>
</tr>
<tr>
<td>0.01</td>
<td>10201</td>
<td>0.000026896061183</td>
<td>0.000192818284368</td>
<td>0.003480757496468</td>
<td>0.009357494818066</td>
<td>0.002829075305256</td>
</tr>
<tr>
<td>0.008</td>
<td>15876</td>
<td>0.000016325645826</td>
<td>0.000123788828705</td>
<td>0.002659776917561</td>
<td>0.007392518806456</td>
<td>0.00197375721534</td>
</tr>
<tr>
<td>0.00625</td>
<td>25921</td>
<td>0.000008309820004</td>
<td>0.000075853361019</td>
<td>0.001757193882857</td>
<td>0.005748635151850</td>
<td>0.001467538069638</td>
</tr>
<tr>
<td>0.005</td>
<td>40401</td>
<td>0.0000051901629038</td>
<td>0.000048184095004</td>
<td>0.001547437415912</td>
<td>0.00492760337862</td>
<td>0.000939052664736</td>
</tr>
</tbody>
</table>
Figure 5.3: \(L^2\)-norm of errors for regular grids. Testing function: \(f(x, y) = x^2y\)

- Irregular grid

Table 5.2: \(L^2\)-norm of errors for irregular grid. Testing function: \(f(x, y) = x^2y\)
Figure 5.4: $L^2$-norm of errors for irregular grids. Testing function: $f(x, y) = x^2y$
Figure 5.5: errors with boundary (green), without 5 outmost curves (red) and without 10 outmost curves (black). Testing function: $f(x, y) = x^2 y$

**Example 8.** $f(x, y) = \sin(x) \cos(y)$. The exact derivatives are: $f_x = \cos(x) \cos(y)$, $f_y = -\sin(x) \sin(y)$, $f_{xx} = -\sin(x) \cos(y)$, $f_{xy} = -\cos(x) \sin(y)$, $f_{yy} = -\sin(x) \cos(y)$. 
Figure 5.6: Testing function: \( f(x, y) = \sin(x) \cos(y) \)

- Regular grid

Table 5.3: \( L^2 \)-norm of errors for regular grid. Testing function: \( f(x, y) = \sin(x) \cos(y) \)

<table>
<thead>
<tr>
<th>( \delta_x )</th>
<th>( N )</th>
<th>( |e_x|_2 )</th>
<th>( |e_y|_2 )</th>
<th>( |e_{xx}|_2 )</th>
<th>( |e_{yy}|_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>9</td>
<td>0.120615589844898</td>
<td>0.057736216529774</td>
<td>0.382956361210522</td>
<td>0.288376078293107</td>
</tr>
<tr>
<td>0.25</td>
<td>25</td>
<td>0.0252010849566455</td>
<td>0.014350068042187</td>
<td>0.152894549139699</td>
<td>0.125158700502675</td>
</tr>
<tr>
<td>0.2</td>
<td>36</td>
<td>0.021724489573717</td>
<td>0.011488314426837</td>
<td>0.114103638484708</td>
<td>0.099329070460744</td>
</tr>
<tr>
<td>0.125</td>
<td>81</td>
<td>0.012978555492100</td>
<td>0.006082376934443</td>
<td>0.062202742014695</td>
<td>0.055808971047159</td>
</tr>
<tr>
<td>0.1</td>
<td>121</td>
<td>0.009668667323457</td>
<td>0.004450829411610</td>
<td>0.044882899411610</td>
<td>0.040553824710355</td>
</tr>
<tr>
<td>0.05</td>
<td>441</td>
<td>0.002940027061993</td>
<td>0.001243114245310</td>
<td>0.018061250571999</td>
<td>0.0148881795170</td>
</tr>
<tr>
<td>0.04</td>
<td>676</td>
<td>0.001939706130025</td>
<td>0.000482497452681</td>
<td>0.014675671739866</td>
<td>0.010866945436581</td>
</tr>
<tr>
<td>0.025</td>
<td>1681</td>
<td>0.000792293142615</td>
<td>0.000311309898603</td>
<td>0.007502870989914</td>
<td>0.005619333392112</td>
</tr>
<tr>
<td>0.02</td>
<td>2601</td>
<td>0.000512921251930</td>
<td>0.000208300262461</td>
<td>0.00610305168029</td>
<td>0.004164894595025</td>
</tr>
<tr>
<td>0.0125</td>
<td>6561</td>
<td>0.000204389930106</td>
<td>0.000076781053873</td>
<td>0.00326979956307</td>
<td>0.002228273448109</td>
</tr>
<tr>
<td>0.01</td>
<td>10201</td>
<td>0.000131741063890</td>
<td>0.000049828763750</td>
<td>0.002690463798413</td>
<td>0.001660201437108</td>
</tr>
<tr>
<td>0.008</td>
<td>15876</td>
<td>0.000084592688836</td>
<td>0.000031406393449</td>
<td>0.00203128407119</td>
<td>0.001268455326522</td>
</tr>
<tr>
<td>0.00625</td>
<td>25921</td>
<td>0.000051840944886</td>
<td>0.000019176170411</td>
<td>0.001482086609061</td>
<td>0.000948751743686</td>
</tr>
<tr>
<td>0.005</td>
<td>40401</td>
<td>0.000033636871941</td>
<td>0.000011949793464</td>
<td>0.001221125088138</td>
<td>0.000697031705529</td>
</tr>
</tbody>
</table>
Figure 5.7: $L^2$-norm of errors for regular grid. Testing function: $f(x, y) = \sin(x) \cos(y)$

- Irregular grid

Table 5.4: $L^2$-norm errors for irregular grids. Testing function: $f(x, y) = \sin(x) \cos(y)$
Figure 5.8: $L^2$-norm of errors for irregular grid. Testing function: $f(x, y) = \sin(x) \cos(y)$
Figure 5.9: errors with boundary (green), without 5 outmost curves (red) and without 10 outmost curves (black)

Example 9. \( f(x, y) = \exp \left( -\frac{(x-1)^2 + (y-\pi/2)^2}{4} \right) \).
Figure 5.10: \( f(x, y) = \exp \left( -\frac{(x-1)^2 + (y-\pi/2)^2}{4} \right) \)

- Regular grid

Table 5.5: \( L^2 \)-norm of errors for regular grid. Testing function: \( f(x, y) = \exp \left( -\frac{(x-1)^2 + (y-\pi/2)^2}{4} \right) \).

<table>
<thead>
<tr>
<th>( n, k, r )</th>
<th>( N )</th>
<th>( |e_x|_2 )</th>
<th>( |e_y|_2 )</th>
<th>( |e_{xx}|_2 )</th>
<th>( |e_{xy}|_2 )</th>
<th>( |e_{yy}|_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>9</td>
<td>0.031157427311096</td>
<td>0.042044677130126</td>
<td>0.120590855664554</td>
<td>0.066493409518022</td>
<td>0.207667504307205</td>
</tr>
<tr>
<td>0.25</td>
<td>25</td>
<td>0.008034428569278</td>
<td>0.011466386354540</td>
<td>0.056663412542699</td>
<td>0.028181954702427</td>
<td>0.088659605407397</td>
</tr>
<tr>
<td>0.2</td>
<td>36</td>
<td>0.006242880726889</td>
<td>0.009204773590726</td>
<td>0.045839332142293</td>
<td>0.024039172832730</td>
<td>0.067458094467925</td>
</tr>
<tr>
<td>0.125</td>
<td>81</td>
<td>0.00302257220907</td>
<td>0.005222654095194</td>
<td>0.023991214789428</td>
<td>0.014386642125142</td>
<td>0.032518031900628</td>
</tr>
<tr>
<td>0.1</td>
<td>121</td>
<td>0.002078368074359</td>
<td>0.003767926103131</td>
<td>0.016733720572893</td>
<td>0.011572009262132</td>
<td>0.021382670237427</td>
</tr>
<tr>
<td>0.05</td>
<td>441</td>
<td>0.000571529247773</td>
<td>0.001120867490785</td>
<td>0.004846113255806</td>
<td>0.006769794701003</td>
<td></td>
</tr>
<tr>
<td>0.04</td>
<td>676</td>
<td>0.000373157668002</td>
<td>0.000738328019191</td>
<td>0.003674506435200</td>
<td>0.0046509036602</td>
<td></td>
</tr>
<tr>
<td>0.025</td>
<td>1681</td>
<td>0.000146096056530</td>
<td>0.000296972025145</td>
<td>0.002135205525530</td>
<td>0.002239883824207</td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td>2601</td>
<td>0.000094631849768</td>
<td>0.000197731492589</td>
<td>0.001080008207892</td>
<td>0.001517968227610</td>
<td></td>
</tr>
<tr>
<td>0.0125</td>
<td>6561</td>
<td>0.000036867535005</td>
<td>0.000076211648295</td>
<td>0.001040476990903</td>
<td>0.000812673505110</td>
<td></td>
</tr>
<tr>
<td>0.01</td>
<td>10201</td>
<td>0.000023830982111</td>
<td>0.000049113326488</td>
<td>0.000852605580233</td>
<td>0.000791939655154</td>
<td></td>
</tr>
<tr>
<td>0.008</td>
<td>15876</td>
<td>0.000015290531072</td>
<td>0.000031490583328</td>
<td>0.000659592415477</td>
<td>0.000621500315314</td>
<td></td>
</tr>
<tr>
<td>0.00625</td>
<td>25921</td>
<td>0.0000092538282218</td>
<td>0.000019285646333</td>
<td>0.000491852898955</td>
<td>0.000496480551000</td>
<td></td>
</tr>
<tr>
<td>0.005</td>
<td>40401</td>
<td>0.000006000237003</td>
<td>0.000012553601606</td>
<td>0.000393954044802</td>
<td>0.000376715722775</td>
<td></td>
</tr>
</tbody>
</table>

48
Figure 5.11: \( L^2 \)-norm of errors for regular grid. Testing function: 
\[
 f(x, y) = \exp\left(-\frac{(x-1)^2+(y-\pi/2)^2}{4}\right)
\]

- Irregular grid

Table 5.6: \( L^2 \)-norm of errors for irregular grid. Testing function: 
\[
 f(x, y) = \exp\left(-\frac{(x-1)^2+(y-\pi/2)^2}{4}\right)
\]

<table>
<thead>
<tr>
<th>( n, k, r )</th>
<th>( N )</th>
<th>( |e_x|_2 )</th>
<th>( |e_y|_2 )</th>
<th>( |e_{xx}|_2 )</th>
<th>( |e_{xy}|_2 )</th>
<th>( |e_{yy}|_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>50, 20, 0.2</td>
<td>9586</td>
<td>0.004130712867827</td>
<td>0.002715185250205</td>
<td>0.055317754369959</td>
<td>0.013122947362960</td>
<td>0.013254180206437</td>
</tr>
<tr>
<td>60, 20, 0.2</td>
<td>11524</td>
<td>0.002135832426183</td>
<td>0.002258015051629</td>
<td>0.008812167558664</td>
<td>0.003194226121106</td>
<td>0.009998191015709</td>
</tr>
<tr>
<td>70, 20, 0.2</td>
<td>13446</td>
<td>0.001926262682253</td>
<td>0.002010798313030</td>
<td>0.007904008634448</td>
<td>0.003023998642574</td>
<td>0.009361721829307</td>
</tr>
<tr>
<td>80, 20, 0.2</td>
<td>15376</td>
<td>0.001762490449948</td>
<td>0.001803219237434</td>
<td>0.007213140557686</td>
<td>0.002939442914175</td>
<td>0.008701580801215</td>
</tr>
<tr>
<td>90, 20, 0.2</td>
<td>17306</td>
<td>0.001631038414827</td>
<td>0.001632020076704</td>
<td>0.006678039280818</td>
<td>0.002762188332649</td>
<td>0.008206650558170</td>
</tr>
<tr>
<td>100, 20, 0.2</td>
<td>19228</td>
<td>0.001517574390022</td>
<td>0.001491649377844</td>
<td>0.006395294022045</td>
<td>0.002711883214904</td>
<td>0.007696758368762</td>
</tr>
<tr>
<td>120, 20, 0.15</td>
<td>23080</td>
<td>0.00115605624206</td>
<td>0.001171678577591</td>
<td>0.005113663777403</td>
<td>0.002150189602122</td>
<td>0.006120399204716</td>
</tr>
<tr>
<td>150, 20, 0.12</td>
<td>28850</td>
<td>0.00082254386886</td>
<td>0.000879880052483</td>
<td>0.004271666146067</td>
<td>0.001719147029568</td>
<td>0.004743781833951</td>
</tr>
<tr>
<td>180, 20, 0.1</td>
<td>34628</td>
<td>0.000632126947630</td>
<td>0.000686365471944</td>
<td>0.003737107824970</td>
<td>0.001628377643068</td>
<td>0.003833026778016</td>
</tr>
<tr>
<td>200, 10, 0.09</td>
<td>38488</td>
<td>0.000537024997616</td>
<td>0.000588484417971</td>
<td>0.003493593304511</td>
<td>0.001393762519661</td>
<td>0.003490504112974</td>
</tr>
</tbody>
</table>
Figure 5.12: $L^2$-norm of errors for irregular grid. Testing function: $f(x, y) = \exp\left(-\frac{(x-1)^2 + (y-\pi/2)^2}{4}\right)$
Figure 5.13: errors with boundary (green), without 5 outmost curves (red) and without 10 outmost curves (black). Testing function: $f(x, y) = \exp \left( \frac{-(x-1)^2 + (y-\pi/2)^2}{4} \right)$

Example 10. $f(x, y) = e^{x+y}$. The derivatives are $f_x = e^{x+y}$, $f_y = e^{x+y}$, $f_{xx} = e^{x+y}$, $f_{xy} = e^{x+y}$, $f_{yy} = e^{x+y}$
Figure 5.14: $\exp(x + y)$

- Regular grid

Table 5.7: $L^2$-norm of errors for regular grids. Testing function: $f(x, y) = e^{x+y}$

<table>
<thead>
<tr>
<th>$\delta_x$</th>
<th>$N$</th>
<th>$|e_x|_2$</th>
<th>$|e_y|_2$</th>
<th>$|e_{xx}|_2$</th>
<th>$|e_{xy}|_2$</th>
<th>$|e_{yy}|_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>9</td>
<td>0.88823504860421</td>
<td>0.66913671979692</td>
<td>1.946462170364392</td>
<td>2.350097773432172</td>
<td>2.888983544522133</td>
</tr>
<tr>
<td>0.25</td>
<td>25</td>
<td>0.140836497367585</td>
<td>0.136368324777584</td>
<td>0.858268908167091</td>
<td>0.868431779409235</td>
<td>0.884039882247917</td>
</tr>
<tr>
<td>0.2</td>
<td>36</td>
<td>0.1110268644318</td>
<td>0.10483866046477</td>
<td>0.68450154884765</td>
<td>0.723660849827877</td>
<td>0.68945150816126</td>
</tr>
<tr>
<td>0.125</td>
<td>81</td>
<td>0.05997349711407</td>
<td>0.05993886444775</td>
<td>0.36286457717459</td>
<td>0.413670013502974</td>
<td>0.388387600853351</td>
</tr>
<tr>
<td>0.1</td>
<td>121</td>
<td>0.043168423206122</td>
<td>0.04345329912245</td>
<td>0.26657165650388</td>
<td>0.316177395164513</td>
<td>0.24489387856028</td>
</tr>
<tr>
<td>0.05</td>
<td>441</td>
<td>0.012765168265901</td>
<td>0.01298158546118</td>
<td>0.09128085281688</td>
<td>0.123809120106266</td>
<td>0.096084056715250</td>
</tr>
<tr>
<td>0.04</td>
<td>676</td>
<td>0.008455634576499</td>
<td>0.00881135915203</td>
<td>0.079352339768262</td>
<td>0.092632537581324</td>
<td>0.071342963898786</td>
</tr>
<tr>
<td>0.025</td>
<td>1681</td>
<td>0.00343086290575</td>
<td>0.003397716337803</td>
<td>0.034661823010541</td>
<td>0.050654896434902</td>
<td>0.038637578594788</td>
</tr>
<tr>
<td>0.02</td>
<td>2601</td>
<td>0.00229508498376</td>
<td>0.0023282179740</td>
<td>0.0205615441647</td>
<td>0.02588221540569</td>
<td>0.02660507082366</td>
</tr>
<tr>
<td>0.0125</td>
<td>6561</td>
<td>0.00088675859818</td>
<td>0.00088656458631</td>
<td>0.01331295567948</td>
<td>0.022086452870450</td>
<td>0.015841838751833</td>
</tr>
<tr>
<td>0.01</td>
<td>10201</td>
<td>0.000573403613122</td>
<td>0.000579196044007</td>
<td>0.009586161020297</td>
<td>0.017200321026664</td>
<td>0.01171462908191</td>
</tr>
<tr>
<td>0.008</td>
<td>15876</td>
<td>0.000368396217109</td>
<td>0.00037242578855</td>
<td>0.007027651699667</td>
<td>0.013446080055654</td>
<td>0.009257608601065</td>
</tr>
<tr>
<td>0.00625</td>
<td>25921</td>
<td>0.00022649630257</td>
<td>0.000225873156451</td>
<td>0.005817554288823</td>
<td>0.010132770526332</td>
<td>0.00669127415158</td>
</tr>
<tr>
<td>0.005</td>
<td>40401</td>
<td>0.000144887948103</td>
<td>0.0001468427464947</td>
<td>0.003723819036054</td>
<td>0.008128938293999</td>
<td>0.005696043097810</td>
</tr>
</tbody>
</table>
Figure 5.15: $L^2$-norm of errors for regular grids. Testing function: $f(x, y) = e^{x+y}$.

- Irregular grid

Table 5.8: $L^2$-norm of errors for irregular grid. Testing function: $f(x, y) = e^{x+y}$.

<table>
<thead>
<tr>
<th>$n$, $k$, $r$</th>
<th>$N$</th>
<th>$|e_x|_2$</th>
<th>$|e_y|_2$</th>
<th>$|e_{xx}|_2$</th>
<th>$|e_{xy}|_2$</th>
<th>$|e_{yy}|_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>50,20,0.2</td>
<td>9586</td>
<td>0.556490774979533</td>
<td>25.812004125313118</td>
<td>5.900616474206704</td>
<td>4.0089768854174</td>
<td></td>
</tr>
<tr>
<td>60,20,0.2</td>
<td>11524</td>
<td>0.462170367448450</td>
<td>2.964090127190516</td>
<td>3.208409101665976</td>
<td>3.064398968707168</td>
<td></td>
</tr>
<tr>
<td>70,20,0.2</td>
<td>13446</td>
<td>0.4413722594958</td>
<td>2.690810576803576</td>
<td>3.06063329723576</td>
<td>2.87383290112720</td>
<td></td>
</tr>
<tr>
<td>80,20,0.2</td>
<td>15376</td>
<td>0.429198470914126</td>
<td>2.626347980352680</td>
<td>3.007687470672191</td>
<td>2.79239995366796</td>
<td></td>
</tr>
<tr>
<td>90,20,0.2</td>
<td>17306</td>
<td>0.414738776583122</td>
<td>2.54708053967300</td>
<td>2.93532102144987</td>
<td>2.72264621452070</td>
<td></td>
</tr>
<tr>
<td>100,20,0.2</td>
<td>19228</td>
<td>0.40068618879172</td>
<td>2.50764965391389</td>
<td>2.865879662203708</td>
<td>2.65289705288352</td>
<td></td>
</tr>
<tr>
<td>120,20,0.15</td>
<td>23080</td>
<td>0.253941079005656</td>
<td>1.6919217367494</td>
<td>1.93272604099283</td>
<td>1.7758257097725</td>
<td></td>
</tr>
<tr>
<td>150,20,0.12</td>
<td>28850</td>
<td>0.17236836082087</td>
<td>1.227393646599241</td>
<td>1.412374101966284</td>
<td>1.30436157892190</td>
<td></td>
</tr>
<tr>
<td>180,20,0.11</td>
<td>34628</td>
<td>0.145072109999077</td>
<td>1.07104943133347</td>
<td>1.229811443844006</td>
<td>1.11584632946628</td>
<td></td>
</tr>
<tr>
<td>200,20,0.09</td>
<td>38488</td>
<td>0.10171805322406</td>
<td>0.814574521265115</td>
<td>0.95341572425811</td>
<td>0.86471444693164</td>
<td></td>
</tr>
<tr>
<td>210,20,0.09</td>
<td>40438</td>
<td>0.09449690083221</td>
<td>0.78742714390467</td>
<td>0.890153890295157</td>
<td>0.80052396878792</td>
<td></td>
</tr>
</tbody>
</table>
Figure 5.16: $L^2$-norm of errors for irregular grids. Testing function: $f(x, y) = e^x + y$. 
5.2 Conclusions of the numerical results

We have following observations from the examples above:

- The more points we use, the smaller errors we get. Hence, the method is convergent.

- Comparison between tables 5.1 vs 5.2, 5.3 vs 5.4, 5.5 vs 5.6, and 5.7 vs 5.8, we see that with (almost) the same number of points, errors in regular grids are much smaller than in irregular grids, i.e. the convergence is faster in regular grids than in irregular grids.

- From the figures 5.3 – 5.4, 5.7–5.8, 5.11 – 5.12, and 5.15–5.16, we notice that errors of the first order are always smaller than those of the second order. In regular grids, the method converges linearly (order 1) for the first order derivatives, and converges with order 1/2 for the second order derivatives. However, it is not clear to see the order of convergence in irregular grids.
• From the figures 5.5, 5.9, 5.13, and 5.17, we observe that errors without outer boundary curves are always smaller than those of with boundaries. In the first order derivatives, the difference is not much. However, it is significant in the second order. Errors without 5 or without 10 outer boundary curves are almost the same but are much smaller than those of with boundaries. That means, for second order derivatives errors occur in 5 outer boundary curves. This is because the points are scatter in the outer areas, and we can not find neighborhoods in some subregions for those points in the outmost curve.

• Errors depend on functions. They are smaller for quadratic functions and those whose third order derivatives are small. It is reasonable since the FPM use the Taylor expansion.
Chapter 6

Summary

This thesis has applied the Finite Pointset Method (FPM) to approximate derivatives of a function. The point cloud of interest comes from a mathematical modelling of the fibre lay-down process in the non-woven production. The points distribute irregularly. They are cluster in some areas but scatter in the others. Since they are fixed, we could not apply techniques in the general FPM such as adding or removing points. In stead, for each point we had to select its list of neighborhoods. We have given some criterion. Firstly, the neighborhoods are not too far from the considering point and are not too close together. Secondly, they should distribute around it. To do that, we have divided the domain around it into 6 equal subregions. In each part, we selected some closest points which are neither too far nor too close to the point. Then we filtered closely points. This has been done in the algorithm in chapter 4. In chapter 5, we have tested the algorithm with some given functions. We have also calculated the errors without some outer boundary curves. The numerical results have shown that the method is convergent and is faster in regular grids than in irregular grids. It was good for the first order but was not good for the second order in some cases. However, we have observed that errors without 5 outer curves are much smaller than those of with boundaries. This have shown that errors occur mostly due to some outer boundary curves. This is because the points are scatter in outer areas and we could not find neighborhoods in some subregions for the points in the outmost curve. We hope that if we can produce the point cloud better then we can apply the method using real data values in the simulation of the fibre lay-down process.
Bibliography


[17] the NOGRID’s website: http://www.nogrid.com/