Master of Science Thesis

Tracking Free Surface Flows Using the Least Squares Spectral Element Method

Martijn Lantsheer

March 3, 2010

Keywords: Discretisation, Least Squares, Spectral Elements, Poisson, Crank-Nicolson

TU Delft

Tracking Free Surface Flows Using the Least Squares Spectral Element Method

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering at Delft University of Technology

Martijn Lantsheer

March 3, 2010

Faculty of Aerospace Engineering · Delft University of Technology

DELFT UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF AERODYNAMICS

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance the thesis entitled **"Tracking Free Surface Flows Using the Least Squares Spectral Element Method"** by **Martijn Lantsheer** in fulfillment of the requirements for the degree of **Master of Science**.

Dated: March 3, 2010

Board of Examiners:

Prof Dr Ir Drs H. Bijl

Dr Ir M.I. Gerritsma (supervisor)

Dr Ir H.J. de Koning Gans

MSc A. Palha da Silva Clerigo

For some laymen computational fluid dynamics may seem like philosophy,
experimentalists can argue for days about the feasibility of the flow computed,
on grounds that 'PIV is the solution'. Right...

Preface

At this section of the report it is custom to look back with (hopefully) fulfillment over a period of hard work. Of a time of fun, playing with the own written programs but also fun with other Master students who where willing to take up many challenges at the table tennis table with me to blow out frustration when stuck at some problem. Well appreciated was a much closer relation to teachers and PhD students at the department, feeling more than just being an aerospace student, rather it felt as being integrated like an employee (excepts the negative salary being study fees).

And what would be a Preface without special thanks to our supervisor Marc Gerritsma. I then think especially of his interest in his students, by coming down to the basement now and then to check up on the progress and/or problems his students have encountered. Sometimes I remember what Marc wrote at the beginning of his PhD paper that he was saved of becoming a taxi driver by the support of his family and friends. Maybe he just wanted to make sure that I would be saved of that same threat during any time of my graduation. Thinking of this gave me a smile and new motivation to go on. In other words support was always there when needed and with that I also mean being able to drop by his office any time.

Further thanks to all others including family, readers and members of the board for all they have contributed. Of course also special thanks to my fiancée for having patients with me when I had to tell her time after time that my graduation suffered some delay.

Also Alex should not be forgotten, because he was the second pilar of making this graduation possible, we shared a good 6-7 months of efforts on this project and the result was fast progress with a lot of fun.

Summary

The reader is introduced into the basic characteristics of using spectral elements, which implies using orthogonal Lagrange polynomials as basis function and the minimization using least squares. Combined, this will result into the least square spectral element method (LSQSEM). The real strength of the spectral elements is the accuracy of integration by increasing the polynomial order (p-refinement) of the basis function. This will result in a faster convergence of the integration error compared to increasing the amount of elements (h-refinement). With confidence it can therefore be applied on tracking free surface flows using a combination of the LSQSEM spatial and Crank-Nicolson in the time integration.

With the least square spectral element method some mathematical problems can be discretised. This can be for example the setting up of a linear operator that can be applied to any kind of function. The construction of a linear operator is a good tool to use if one desires to solve flow problems like Navier-Stokes, Euler or Potential flows.

The next step is to apply the LSQSEM to a given flow equation: the Poisson equation. To solve the Poisson equation using the LSQSEM one needs to discretise a coupled system defined by the Laplacian of a velocity potential. The Poisson equation is thoroughly tested on different kinds of known functions for both p- and h-refinement and it showed once more that p-refinement results in a faster convergence rate compared to h-refinement.
To be able to apply the LSQSEM to free surface tracking, one needs to explain what free surfaces are and how a free surface can be tracked by means of a tracer function. The tracer function is moved by inserting the velocity field that will determine the movent of the flow. The time dependence that is introduced by tracking the moving surface is discretised using a second order accurate Crank-Nicolson scheme. This is a one step scheme that only requires the storage of the previous time step and is thus efficient in terms of CPU-time. Combining therefore time (Crank-Nicolson) and spatial (LSQSEM) discretisation will yield the solution to the tracer function.
Finally the tracer function and the Poisson equation are combined to solve a flow problem. All the aspects concerning the combination of the two are treated, including the setting up the boundary conditions and how an iterative process can be initiated. As an example, a standing wave is initially exited and then released so that the only driver working on the system is gravity. One side of the free surface was fluid, the other is set as a vacuum. The righthand side of the Poisson equation is taken to be zero, so the flow is considered inviscid, meaning no wall friction applies.

The results are remarkable. On a one elemental basis with polynomial orders up to $N = 5$ it is possible to accurately keep a free surface flow going for quite some time and at the same time being able to track the free surface contour. Low orders like $N = 3$ have a more linear characteristic and increasing the order enables the polynomial basis function to represent the solution in a more accurate way. However to be able to go to even higher orders, using more elements is inevitable. An alternative could be the use of deformed elements that conform to the free surface.

List of Figures

Nomenclature

Latin Letters

$\mathbf{A}$	Stiffnes Matrix	
$\mathbf{f}$	Right-hand forcing function vector	
BC	Boundary condition	
c	Tracer function	
CFD	Computational fluid dynamics	
DOF	Degrees of freedom	
$\mathbf{F}$	Funtion of:	
FEM	Finite element method	
g	Gravity	m/s^2
GLL	Gauss-Lobatto-Legendre	
$\mathbf{K}$	Stiffnes Matrix	
LSQSEM	Least Squares spectral element method	
m	number of elements	
N	Order of convergence	
n	Polynomial order	
SEM	Spectral element method	
$\mathbf{u}$	Solution vector	
u	Velocity in x-direction	m/s
v	Velocity in y-direction	m/s
w	Integration weights	

Greek Letters

ϵ	Integration error
η	y-coordinate on standard element
$\mathcal{L}$	Linear operator
Ω	Computational domain
ϕ	Basis function
φ	Velocity potential
ξ	x-coordinate on standard element
ζ	z-coordinate on standard element

Subscripts

i, j, k, l	Indexing x- and y-direction
p, q	GLL integration weights in x- and y-direction

Contents

Chapter 1

Introduction

The LSQSEM is around for quite some time now, but nevertheless the interest in it seems little in the engineering world. Quite strange considering the fact that it has been shown multiple times that it quite a powerful method in gathering accurate results for flow solving. The finite Element method (FEM) instead is still much preferred. Using spectral element method (SEM) is in fact a higher order elaboration of the FEM. Should it therefore not be more efficient to use SEM [23]?

Considering its low popularity it is once more a challenge to show that the LSQSEM is indeed a powerful method when it comes to solving flow problems. Therefore a lot of time was spend on thoroughly testing the capabilities of the LSQSEM on different functions and differential equations. This however is not enough, because a convergence study does not show how a solution scheme behaves on a flow problem. It merely confirms the proper implementation and highest accuracy possible. Therefore this report will focus on a flow problem, using the LSQSEM, and apply it to free surface flows in a similar manner as Sherwin [28] describes in his paper.

First, a short review of the theory considered important for the SEM using Least Square can be found in Chapter 2. If in depth background reading on the LSQSEM is needed, this can be found in the Appendices B-C. Chapter 3 presents a two-dimensional sample problem where a general linear operator is discretised using the LSQSEM, because this is the basis for more complex flow problems. Next, Chapter 4, a model problem is introduced: the Poisson equation. An explanation is given on how to discretise the coupled Laplacian system and how well it performs by doing a convergence study (h- and p-refinement) in Chapter 5. Chapter 6 goes into detail on the techniques to solve free surface flows by means of a tracer function. Chapter 6 is showing how the tracer function can be discretised in space and time. An application to tracking free surface flows and combining it with the Poisson equation using the LSQSEM is presented in Chapter, 8. The results of the application are explained in Chapter 9, after which conclusions are drawn, Chapter 10.

Chapter 2

Least Squares Spectral Elements

The spectral element method is an elaborated version of the Finite Element Method (FEM). FEM relies on non-orthogonal polynomials whereas in spectral elements orthogonal polynomials are used. The word 'spectral' originates from Fourier series. The spectrum defines the frequency distribution. The convergence can be described as exponential and thus has similar characteristics as using high order polynomials as basis functions.

2.1 Spectral Element Method

A spectral discretisation relies on the expansion of an arbitrary function, say $u(x)$ in the following way, where $\phi_n(x)$ are the so called basis or trial functions [17, 24]:

$$u(x) = \sum_{n=0}^{\infty} \hat{u}_n \phi_n(x) = \hat{u}_0\phi_0(x) + \cdots + \hat{u}_\infty\phi_\infty(x) \tag{2.1}$$

One may also write this infinite sum as:

$$u_{exact}(x) = \sum_{n=0}^{N} \hat{u}_n \phi_n(x) + \sum_{n=N+1}^{\infty} \hat{u}_n \phi_n(x). \tag{2.2}$$

The exact function is equal to the sum of the two sums - a finite part and an infinite part. An approximate representation of the exact function is obtained by truncating the sum after the N^{th} polynomial degree. The larger N is, the better the approximation u_{appr} becomes.

$$u_{appr}(x) = \sum_{n=0}^{N} \hat{u}_n \phi_n(x). \tag{2.3}$$

The important property of $\phi_n(x)$ is that it is orthogonal with any other basis function up to the N^{th} term. While it is possible to design orthogonal basis functions, it would be very time consuming. One might better use some common basis function expansions that were already derived in the past. The Jacobi polynomials and the derived Chebychev, Legendre and Lagrange polynomials are presented in Appendix A.2. A detailed discussion of the SEM is given in Appendix A. A one-dimensional full step implementation is presented in Appendix B.

2.2 Least Squares Method

The least squares method for finite elements can be used as a means to minimize the residual of the function:

$$\mathbf{Au - f}$$

Where $\mathbf{A}$ is the differential operator, $\mathbf{u}$ is the solution vector, and $\mathbf{f}$ the right hand side forcing function. This can be rewritten as, [16]:

$$I(\mathbf{u}) = \int_{\Omega} (\mathbf{A}\mathbf{u} - \mathbf{f})^2 d\Omega \tag{2.4}$$

Applying variational analysis on the minimization problem will return the following:

$$\int_{\Omega} (\mathbf{A}\phi)^T (\mathbf{A}\mathbf{u} - \mathbf{f}) d\Omega = 0 \qquad \forall \quad \phi \tag{2.5}$$

which is the basis of the least squares method (LSM).

Combining least squares with the SEM will result in the Least Squares Spectral Element Method (LSQSEM). LSQSEM is an efficient method as it inherits the locality of the FEM and is symmetric positive for high order problems [7, 4, 16]. This means that the method keeps its geometric flexibility and combines it with the high order accuracy of the Spectral method. Furthermore there is no LBB-condition and is therefore easy to implement in all kinds of different types of equations without the loss of accuracy, showing a robustness on all kinds of problems without any fluctuations or 'wiggle' like convergence behavior for smooth functions as compared to Galerkin. For non-smooth problems wiggles will show in the convergence.

2.3 Computational domain - Standard element

For the remainder of this thesis report a lot of attention will be payed to the computational domain $\Omega = [-1, 1]$, the so called standard element, as on this interval the basis functions will be orthogonal. Nevertheless the basis functions can be made orthogonal on all intervals, but doing this for just one interval allows mapping on to all intervals. For distorted domains a transformation has to be defined to make the domain fit the standard element again. Normally for the x-direction the letter ξ is used. For y-direction η will be used and ζ accordingly for z-direction where:

$$\begin{aligned} x &\rightarrow \xi: \quad -1 \leq \xi \leq 1 \\ y &\rightarrow \eta: \quad -1 \leq \eta \leq 1 \\ z &\rightarrow \zeta: \quad -1 \leq \zeta \leq 1 \end{aligned}$$

All computations of this thesis, both 1D and 2D, are performed on a standard element for simplicity.

Chapter 3

Linear Operator

For the Poisson equation, Heat equation, Navier Stokes equations and other flow problems one needs to discretise (partial) differential equations or systems of partial differential equations. These will consist of first, second or higher order differentials. When applying LS to the spectral elements a linear system is required. Appendix B gives a good illustration how the LSQSEM can be applied on a first order derivative in one dimension. This chapter will expand this to make it possible to write a linear operator in two dimensions in the form of:

$$\mathcal{L} = a\frac{\partial}{\partial x} + b\frac{\partial}{\partial y} + c \tag{3.1}$$

In (3.1) a, b and c are constants so that by choosing them appropriately every possible linear operator desired can be obtained. Higher order differential equations can be solved by LSQSEM. The equations first have to be rewritten as a system of first order differential equations. Chapter 4 shows how the linear operator presented in the next section can be applied to these systems of equations.

3.1 Linear Operator application

In this section a short summary is given on the two-dimensional implementation of a linear operator in abstract form (see B). Let us therefore define a function $u(x, y)$ and apply the linear operator discussed above (3.1). Then the system to be solved becomes:

$$\mathcal{L}u(x, y) = f(x, y) \tag{3.2}$$

The two functions $u(x, y)$ and $f(x, y)$ are expanded using orthogonal basis functions (Chapter 2 and Appendix A.2):

$$u(x, y) = \sum_{i=0}^{N}\sum_{j=0}^{N} \hat{u}_{ij}\phi_i(x)\phi_j(y). \tag{3.3}$$

and,

$$f(x, y) = \sum_{i=0}^{N}\sum_{j=0}^{N} \hat{f}_{ij}\phi_i(x)\phi_j(y). \tag{3.4}$$

Applying the operator $\mathcal{L}$ yields either

$$\mathcal{L}u(x, y) = \sum_{i=0}^{N}\sum_{j=0}^{N} \mathcal{L}\hat{u}_{ij}\phi_i(x)\phi_j(y) = \hat{f}_{ij}\phi_i(x)\phi_j(y) \tag{3.5}$$

or in a more detailed form,

$$\mathcal{L}u(x, y) = \sum_{i=0}^{N}\sum_{j=0}^{N} \hat{u}_{ij}\left[a\frac{\partial\phi_i(x)}{\partial x}\phi_j(y) + b\phi_i(x)\frac{\partial\phi_j(y)}{\partial y} + c\phi_i(x)\phi_j(y)\right] = \hat{f}_{ij}\phi_i(x)\phi_j(y). \tag{3.6}$$

This has to be multiplied with the LS operator, see Appendix B. The system to be solved reads

$$\int_{\Omega} (\mathcal{L}v)^T (\mathcal{L}u - f) d\Omega = 0 \qquad \forall \quad \phi \tag{3.7}$$

which is essentially what was given by 2.5. The LO hence can be presented by a matrix A. For the integration the so-called Gauss-Lobatto quadrature is used (see Appendix A.5.1). The LO used here is actually the same LO as was applied before. However, for the discretizations a different indexing is used.

$$\left[a \frac{\partial \phi_k(x)}{\partial x} \phi_l(y) + b \phi_k \frac{\partial \phi_l(y)}{\partial y} + c \phi_k(x) \phi_l(y) \right] \tag{3.8}$$

Inserting the least sqares linear operator (3.8) into (3.6) returns:

$$\sum_{i=0}^{N} \sum_{j=0}^{N} \left[a \frac{\partial \phi_k(x)}{\partial x} \phi_l(y) + b \phi_k(x) \frac{\partial \phi_l(y)}{\partial y} + c \phi_k(x) \phi_l(y) \right]$$

$$\left(\hat{u}_{ij} \left[a \frac{\partial \phi_i(x)}{\partial x} \phi_j(y) + b \phi_i(x) \frac{\partial \phi_j(y)}{\partial y} + c \phi_i(x) \phi_j(y) \right] - \hat{f}_{ij}(x_p, y_q) \phi_i(x) \phi_j(y) \right) = 0 \forall \quad k, l. \tag{3.9}$$

Finally applying the Gauss-Lobatto quadrature results in,

$$\sum_{p=0}^{N} \sum_{q=0}^{N} \sum_{i=0}^{N} \sum_{j=0}^{N} \left[a \frac{\partial \phi_i(x)}{\partial x} \phi_l(y_q) + b \phi_k(x) \frac{\partial \phi_l(y)}{\partial y} + c \phi_k(x_p) \phi_l(y) \right]$$

$$\left(\hat{u}_{ij} \left[a \frac{\partial \phi_i(x)}{\partial x} \phi_j(y_q) + b \phi_i(x) \frac{\partial \phi_j(y)}{\partial y} + c \phi_i(x_p) \phi_j(y_q) \right] - \hat{f}_{ij}(x_p, y_q) \phi_i(x) \phi_j(y) \right) w_p w_q = 0 \tag{3.10}$$

The points x_p and x_q are the GLL-zeros in x- and y-direction, respectively. Therefore the next step is to construct the stiffness matrix K and fill it in essence with the terms above like:

$$K = \begin{array}{c} \\ \downarrow \bar{l} \end{array} \begin{array}{c} \begin{array}{cc} \rightarrow \bar{k} & \end{array} \\ \begin{bmatrix} (0,0) & \cdots \\ \vdots & \ddots \\ & & (\bar{l}, \bar{k}) \end{bmatrix} \end{array} \tag{3.11}$$

With:

$$\bar{k} = i + (j)(N+1)$$
$$\bar{l} = k + (l)(N+1)$$

as described in Appendix A.2.4. The different terms belonging to one specific point $K(\bar{l}, \bar{k})$ can be added together to form one matrix system. Therefore to solve the linear system

$$\mathbf{K} \hat{u} = \mathbf{M} \hat{f}, \tag{3.12}$$

the last step left is incorporating the boundary condition.

3.2 Sorting for incorporating Boundary condition

For every single derivative in one direction one needs one boundary condition [10]. In the case of the linear operator $\mathcal{L}$ applied to u one would need two boundary conditions to obtain a result. As one would like to solve a first order system still, it is necessary to sort and renumber (in other words sort the stiffness matrix $\mathbf{K}$ so that the non boundary terms are mentioned first in the vector $\hat{u}_b$ instead of a matrix, where (idem for $\hat{f}_b$):

$$\begin{pmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \vdots \\ \hat{u}_{(N+1)^2} \end{pmatrix} = \begin{pmatrix} \hat{u}_{(1,1)} \\ \hat{u}_{(2,1)} \\ \vdots \\ \hat{u}_{(N+1,N+1)} \end{pmatrix}. \tag{3.13}$$

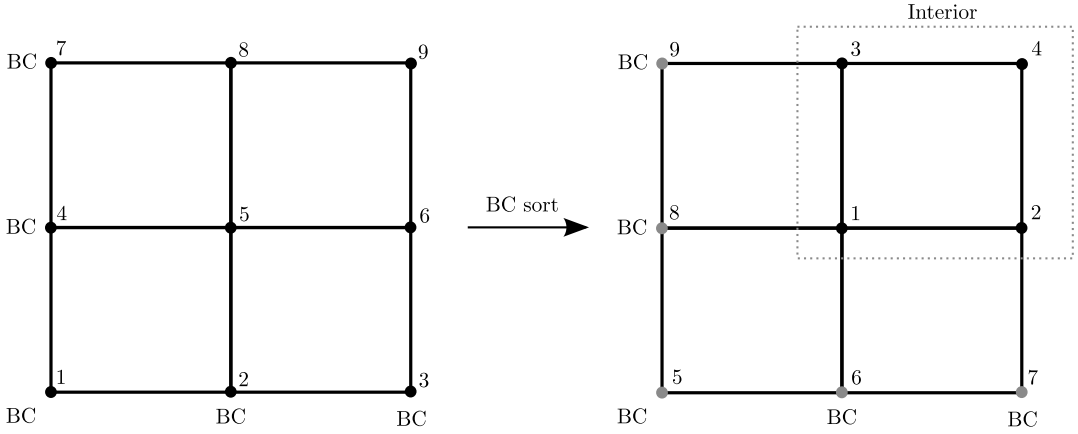


Figure 3.1: Sorting of node numbering from standard to BC numbering ($N = 2$)

This new numbering of $\hat{u}$ and $\hat{f}$, has to be sorted so that the interior nodes are treated first. Therefore figure 3.1 shows how this transform has to be made for the simple case of $N = 2$. This is a similar sorting procedure as depicted in Appendix A.6, except that even more care has to be taken with bookkeeping to keep track of the nodes in the sorted system. The boundary conditions will be used in the system the following way,

$$\left[\begin{array}{c|c} \mathbf{K}_{Int} & \mathbf{K}_{Int-BC} \\ \hline \mathbf{K}_{BC-Int} & \mathbf{K}_{BC} \end{array}\right] \left(\begin{array}{c} \hat{u}_{Int} \\ \hline \hat{u}_{BC} \end{array}\right) = \left(\begin{array}{c} \hat{f}_{Int} \\ \hline \hat{f}_{BC} \end{array}\right), \tag{3.14}$$

by making the distinction between interior (Int) and boundary (BC) nodes. In this way the interior stiffness matrix $\mathbf{K}_I$ can be used to solve the system. However the boundary condition will be incorporated first to the right hand side forcing function, simply by multiplying the last entries of $\hat{u}_{BC}$ with the Matrix $\mathbf{K}_{Int-BC}$ and this can be taken to the right hand side forcing function. Then the bottom rows in the matrix system are now superfluous and are omitted, leaving equation (3.14) to:

$$\left[\begin{array}{c} \mathbf{K}_{Int} \end{array}\right] \left(\begin{array}{c} \hat{u}_{Int} \end{array}\right) = \left(\begin{array}{c} \hat{f}_{Int} - \mathbf{K}_{Int-BC}\hat{u}_{BC} \end{array}\right) \tag{3.15}$$

This new system can be solved without much trouble by solving (3.15) and remaining with the desired vector $\hat{u}_I$.

Chapter 4

Discretising a Model Problem: Poisson Equation

4.1 The Poisson Equation

The Poisson equation results by applying the Laplacian operator to a potential field, say $\varphi(x, y)$ in two dimensions. This can be written as:

$$\Delta\varphi = \nabla \cdot \nabla\varphi \tag{4.1}$$

This equation can be rewritten as a system of two coupled first order differential equation. The divergence and the gradient operator have to be applied within the coupled system:

$$\begin{aligned} \vec{u} &= \nabla\varphi, \\ \nabla \cdot \vec{u} &= f \end{aligned} \tag{4.2}$$

The first equations represents the gradient operator acting on a potential field producing a vector field. The second equations of (4.2) is the divergence operator acting on this vector field returning a scalar field, which can be expressed as the right hand side forcing function. The Poisson equation has to be solved by means of the LSQSEM method. The right hand side forcing function f is known together with four boundary conditions [10]. In the present chapter Neumann boundary conditions are used at all four sides, i.e. the normal components of $\vec{u}$ are prescribed.

4.2 Writing Poisson as a coupled linear system

For the Poisson equation a new linear operator for applying least squares has to be created. The tools for construction are described in the previous chapter. The first line of (4.2), in two dimensions can be written as:

$$\frac{\partial\varphi}{\partial x} = u, \tag{4.3}$$

$$\frac{\partial\varphi}{\partial y} = v. \tag{4.4}$$

and the second line of the said equation gives one more equation,

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = f(x, y). \tag{4.5}$$

Therefore combining the three equation from (4.3),(4.4) and (4.5) into a linear system - taking the unknowns to the left-handside and the known functions to the right-handside yields:

$$\begin{bmatrix} \partial/\partial x & \partial/\partial y & \\ 1 & 0 & -\partial/\partial x \\ 0 & 1 & -\partial/\partial y \end{bmatrix} \begin{pmatrix} u \\ v \\ \varphi \end{pmatrix} = \begin{pmatrix} f(x,y) \\ 0 \\ 0 \end{pmatrix}, \tag{4.6}$$

Therefore the LO for the Poisson equation reads:

$$\mathcal{L} = \begin{bmatrix} \partial/\partial x & \partial/\partial y & 0 \\ 1 & 0 & -\partial/\partial x \\ 0 & 1 & -\partial/\partial y \end{bmatrix} \tag{4.7}$$

4.3 Discretising the Poisson Equation

The LS operator applied to the basis functions is equal to $\mathcal{L}$ of equation (4.7). Recall that (3.6) holds:

$$\int_\Omega (\mathcal{L}\phi)^T(\mathcal{L}u - f)d\Omega = 0 \qquad \forall \quad \phi$$

with $\mathcal{L}^T$ being:

$$\mathcal{L}^T = \begin{bmatrix} \partial/\partial x & 1 & 0 \\ \partial/\partial y & 0 & 1 \\ 0 & -\partial/\partial x & -\partial/\partial y \end{bmatrix}. \tag{4.8}$$

All of the unknowns and the lefthand-side function are expanded using spectral elements, similar to (3.3). Applying the LO (4.7):

$$\mathcal{L}\phi_i(x)\phi_j(y) = \begin{bmatrix} \frac{\partial\phi_i(x)}{\partial x}\phi_j(y) & \frac{\partial\phi_j(y)}{\partial y}\phi_i(x) & 0 \\ \phi_i(x)\phi_j(y) & 0 & -\frac{\partial\phi_i(x)}{\partial x}\phi_j(y) \\ 0 & \phi_i(x)\phi_j(y) & -\frac{\partial\phi_j(y)}{\partial y}\phi_i(x) \end{bmatrix} \tag{4.9}$$

and

$$\mathcal{L}^T\phi_k(x)\phi_l(y) = \begin{bmatrix} \frac{\partial\phi_k(x)}{\partial x}\phi_l(y) & \phi_k(x)\phi_l(y) & 0 \\ \frac{\partial\phi_l(y)}{\partial y}\phi_k(x) & 0 & \phi_k(x)\phi_l(y) \\ 0 & -\frac{\partial\phi_k(x)}{\partial x}\phi_l(y) & -\frac{\partial\phi_l(y)}{\partial y}\phi_k(x) \end{bmatrix}. \tag{4.10}$$

Then the complete system to be solved (4.11) is obtained by merely inserting the LS operator. The Poisson equation is hence solved in one go, as a linear system. The solutions for the velocity field and the potential field are thus obtained and can be used for various applications.

$$\sum_{p=0}^{N}\sum_{q=0}^{N}\sum_{i=0}^{N}\sum_{j=0}^{N}\begin{bmatrix} \frac{\partial\phi_k(x_p)}{\partial x}\phi_l(y_q) & \phi_k(x_p)\phi_l(y_q) & 0 \\ \phi_k(x_p)\frac{\partial\phi_l(y_q)}{\partial y} & 0 & \phi_k(x_p)\phi_l(y_q) \\ 0 & -\frac{\partial\phi_k(x_p)}{\partial x}\phi_l(y_q) & -\phi_k(x_p)\frac{\partial\phi_l(y_q)}{\partial y} \end{bmatrix}$$
$$\left\{ \begin{bmatrix} \frac{\partial\phi_i(x_p)}{\partial x}\phi_j(y_q) & \phi_i(x_p)\frac{\partial\phi_j(y_q)}{\partial y} & 0 \\ \phi_i(x_p)\phi_j(y_q) & 0 & -\frac{\partial\phi_i(x_p)}{\partial x}\phi_j(y_q) \\ 0 & \phi_i(x_p)\phi_j(y_q) & -\phi_i(x_p)\frac{\partial\phi_j(y_q)}{\partial y} \end{bmatrix} \begin{pmatrix} \hat{u}_{ij}(x_p,y_q) \\ \hat{v}_{ij}(x_p,y_q) \\ \hat{\varphi}_{ij}(x_p,y_q) \end{pmatrix} \right.$$
$$\left. - \begin{pmatrix} \hat{f}_{ij}(x_p,y_q)\phi_i(x)\phi_j(y) \\ 0 \\ 0 \end{pmatrix} \right\} w_p w_q = 0 \qquad \forall \quad k,l \tag{4.11}$$

Chapter 5

Error Analysis on the Model Problem: Poisson equation

As a matter of verifying the proper implementation and accuracy of the Poisson equation it is inevitable to perform an error analysis. Therefore this chapter will deal with the detailed error analysis. For more details on error analysis consult Appendix C. First two different forcing functions are presented which are used in the two-dimensional Poisson calculations. Thereafter both a p-refinement and a h-refinement error analysis are performed. The respective speed of convergence is discussed. Finally, some remarks are given.

5.1 Problem Definition

To validate the discretisation of the Poisson equation known functions are studied. The respective derivatives can be calculated easily. Hence, the obtained forcing function is taken as an input right-hand side function for the Poisson code. The calculations should yield the originally taken known functions. Therefore, taking an arbitrary function,

$$\varphi_1 = x^6 y^2 + y^3 \tag{5.1}$$

will give that

$$\begin{aligned} u_1 &= 6x^5 y^2, \\ v_1 &= 2x^6 y + 3y^2, \\ f_1 &= 30x^4 y^2 + 2x^6 + 6y, \end{aligned} \tag{5.2}$$

The functions u_1, v_1 and f_1 are calculated using equations (4.3, 4.4, 4.5) of Chapter 4. With the right hand side forcing function f_1 and the four boundary conditions (one for every derivative [10]) the system can be solved for the desired function φ_1. The error is expected to converge fast to the exact solution (machine error) because the function φ_1 is of a polynomial nature - just as the basis functions are. Since the function φ_1 is of sixth order in x-direction one would expect that there exists a linear combination of the basis functions up to sixth order which produces the exact solution. It is therefore advisable to look for a function which cannot as easily be presented by a final linear combination of polynomial functions for a solidly founded error analysis. A good choice would be a trigonometric function composed out of sines and cosines.
Thus a second function could look like:

$$\varphi_2 = sin(x)sin(y) \tag{5.3}$$

The respective derivatives are computed as:

$$\begin{aligned} u_2 &= cos(x)cos(y), \\ v_2 &= -sin(x)sin(y), \\ f_2 &= -2sin(x)cos(y). \end{aligned} \tag{5.4}$$

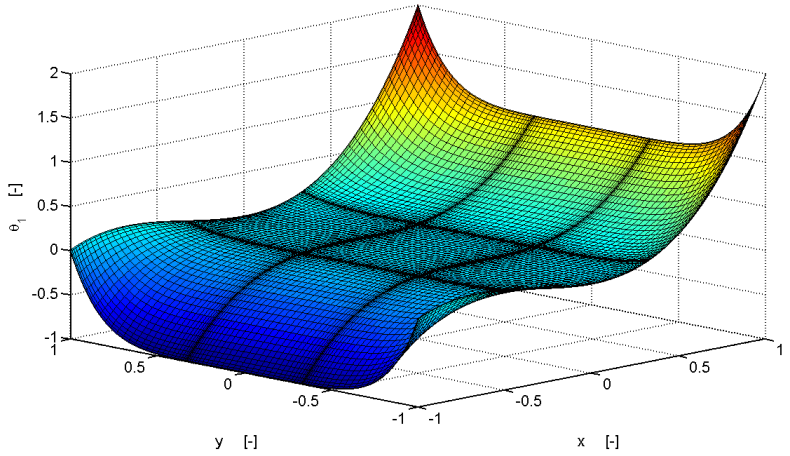


Figure 5.1: Potential field $\varphi_1 = x^6y^2 + y^3$, $m = 9$, $n = 5$

On both functions a h- and p-refinement will be performed and the respective speeds of convergence

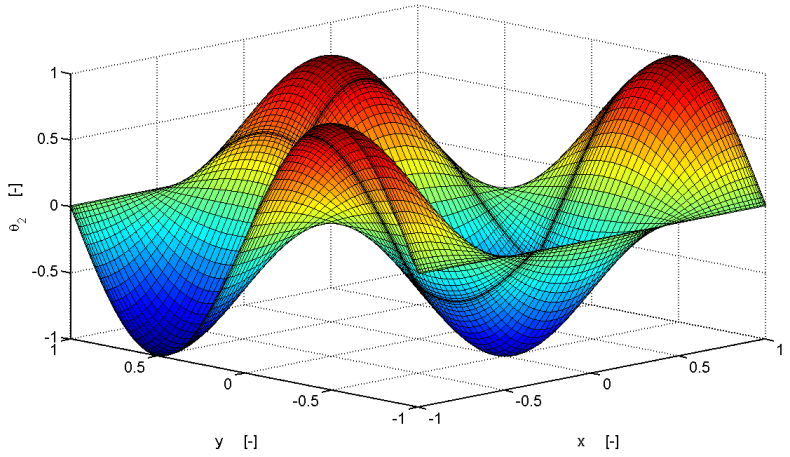


Figure 5.2: Potential field $\varphi_2 = sin(x)cos(y)$, $m = 9$, $n = 5$

are presented in the following sections. For details on error analysis and a comparison of h- and p-refinement consult Appendix C.

5.2 p-Refinement and speed of convergence

For the p-refinement of the first function φ_1 the polynomial order (DOF) is increased while keeping the number of elements m constant. The convergence rate of the error can be determined and will give an indication on how fast the error decreases. As has been pointed out in Appendix C the error decreases exponentially fast. Hence, an exponentially decreasing function is assumed.

$$ln|\epsilon|^2 = a \cdot DOF + b, \tag{5.5}$$

can be used to calculated the error for a certain polynomial order assuming a and b are known constants. a can be identified as the slope of the error curve. Rewriting relation (5.5) one can find the norm of the actual error. This implies that

$$|\epsilon|^2 = c \cdot e^{a \cdot DOF}. \tag{5.6}$$

The result for φ_1 in Figure 5.3 shows clearly that the solution has converged to the exact solution at a degree of freedom of $N = 6$. The error drops to machine accuracy for $N = 6$. This due the fact that φ_1 incorporates a highest order of 6 and this can be represented by polynomials of order $N = 6$. After order $N = 6$ that some increase is visible due to numerical calculation error and truncation errors at machine precision.
The second function φ_2 returns a slower convergence because in this case a sine function can not be exactly represented by polynomials of any finite order. Figure 5.6 shows this with the amount of elements being set to $m = 1$. Further, one sees again that there is a clear 'wiggly' behavior visible in the convergence. A characteristic that is typically found for such trigonometric functions (Appendix C). This is due to the fact that the function chosen φ_2 is point-symmetric with respect to the origin. The approximation of the solution at an even order consists of basis functions that are symmetric. Purely point-symmetric functions can not be approximated by a purely axis-symmetric function. Therefore on every even order it will show a plateau-like shape in the error curve, because the error will hardly decrease. For the even orders a skew-symmetric function should be added. Then the error will decrease smoothly.

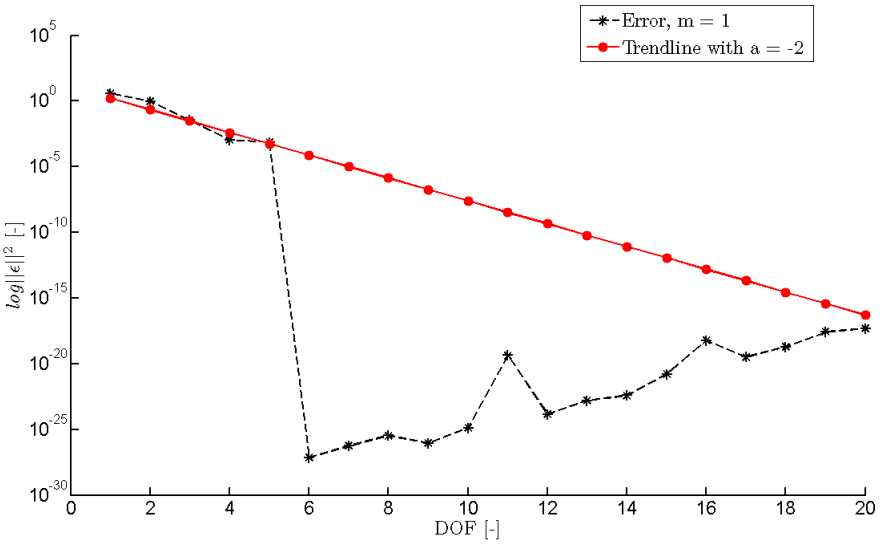


Figure 5.3: Error behavior for $m = 1$, $\varphi_1 = x^6y^2 + y^3$

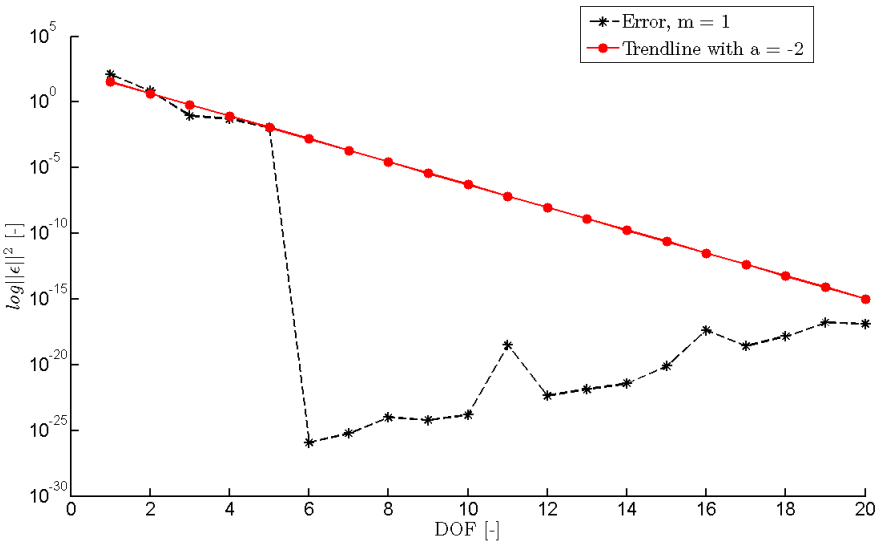


Figure 5.4: Error behavior for $m = 1$, u_1

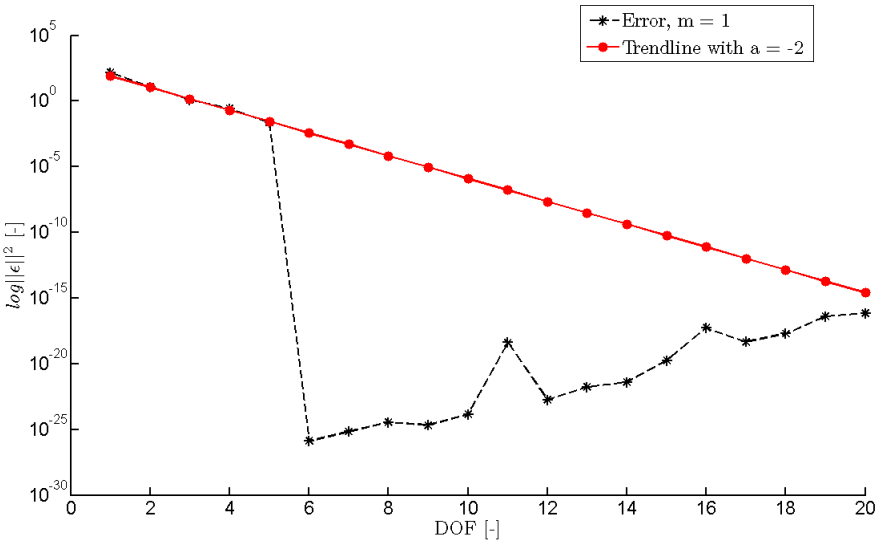


Figure 5.5: Error behavior for $m = 1$, v_1

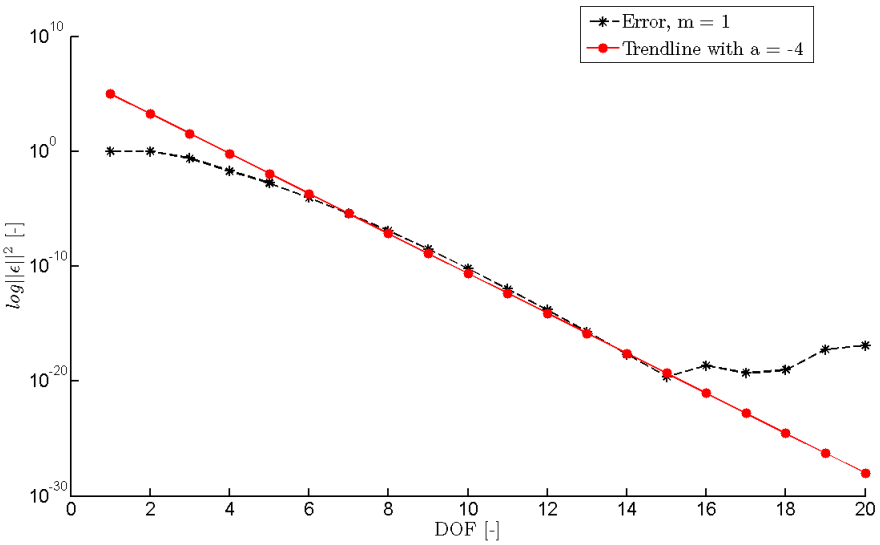


Figure 5.6: Error behavior for $m = 1$, $\varphi_2 = sin(x)cos(y)$

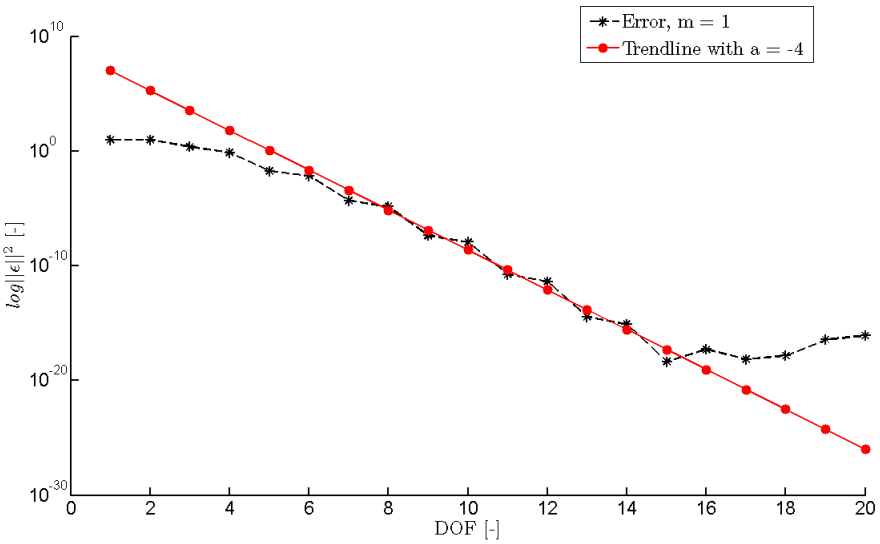


Figure 5.7: Error behavior for $m = 1$, u_2

5.3 h-Refinement and speed of convergence

The real strength of the LSQSEM is the fast convergence by increasing the polynomial order of the basis functions. The previous section revealed exponential convergence of the error for an increase of the polynomial order. The speed of the convergence for the h-refinement however, is different than the one obtained earlier. Equation 5.7 presents the simple relation with whom the error can be evaluated as:

$$ln\|\epsilon\|^2 = a \cdot lnh + b, \tag{5.7}$$

Again a and b are constants, presenting the slope of the error curve and the intersection point with the y-axis, respectively. This implies that

$$\|\epsilon\|^2 = c \cdot h^a. \tag{5.8}$$

With a the slope of the error curve. As the cell size h is proportional to one over the number of elements m:

$$h \propto \frac{1}{m}, \tag{5.9}$$

one expects with a polynomial degree of N that:

$$\|\epsilon\| = c \cdot \left(\frac{1}{m}\right)^{N+1}. \tag{5.10}$$

To illustrate the speed of convergence of the h-refinement, the amount of elements m on the computational domain is increased and the order kept constant. Figure 5.8 shows this for an order of $N = 2$ for φ_1. The trend line for the expected order for $a = 3$ and slowly the solution will converge to this steepness. The trend line in Figure 5.9 shows that the the solution for u approaches an order of $a = 3$ slower compared to the potential. Figure 5.10 converges with the expected slope. This is partly due the fact that in y-direction the order of the function for v_1 is three orders lower compared with the x-direction u_1.

Similar results are obtained for the second potential field φ_2. Figure 5.11 shows slow convergence. Of course a sine-cosine function is more difficult to represent by polynomials. Only the u component is plotted in Figure 5.12, because u_2 and v_2 yield the same function.

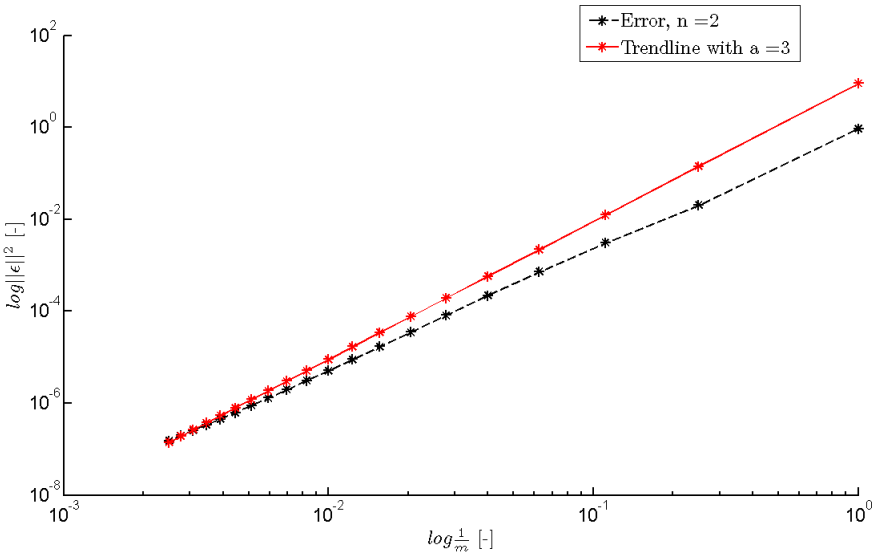


Figure 5.8: Error convergence with trend-line fit for $\varphi_1 = x^6y^2 + y^3$,$N = 2$

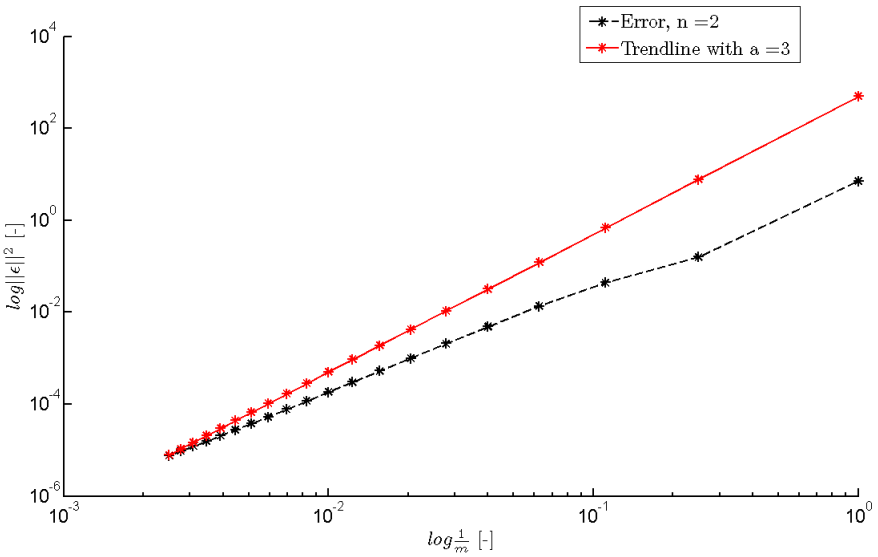


Figure 5.9: Error convergence with trend-line fit for u_1, $N = 2$

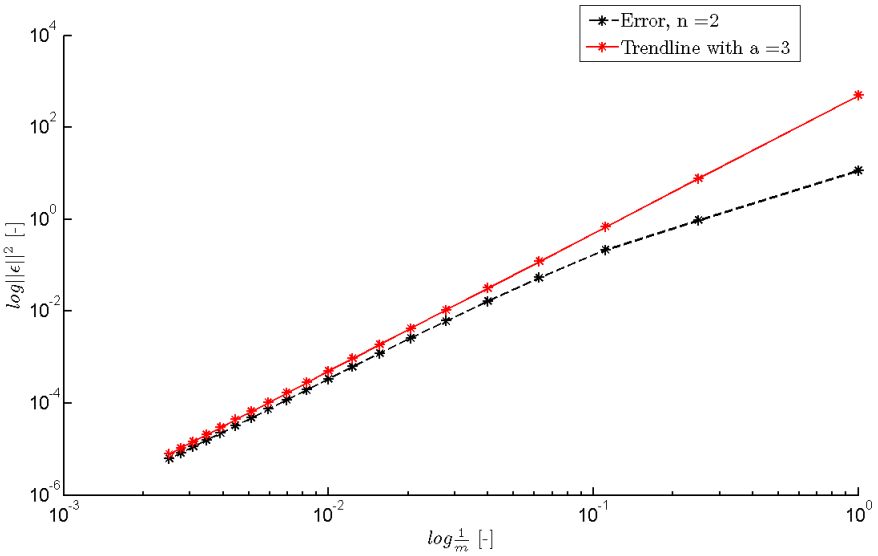


Figure 5.10: Error convergence with trend-line fit for v_1, $N = 2$

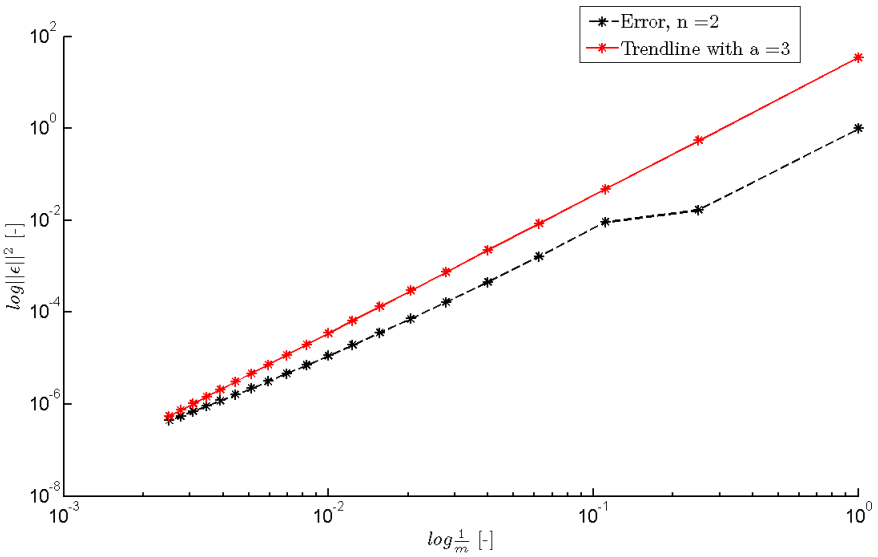


Figure 5.11: Error convergence for $\varphi_2 = sin(x)cos(y)$,$N = 2$

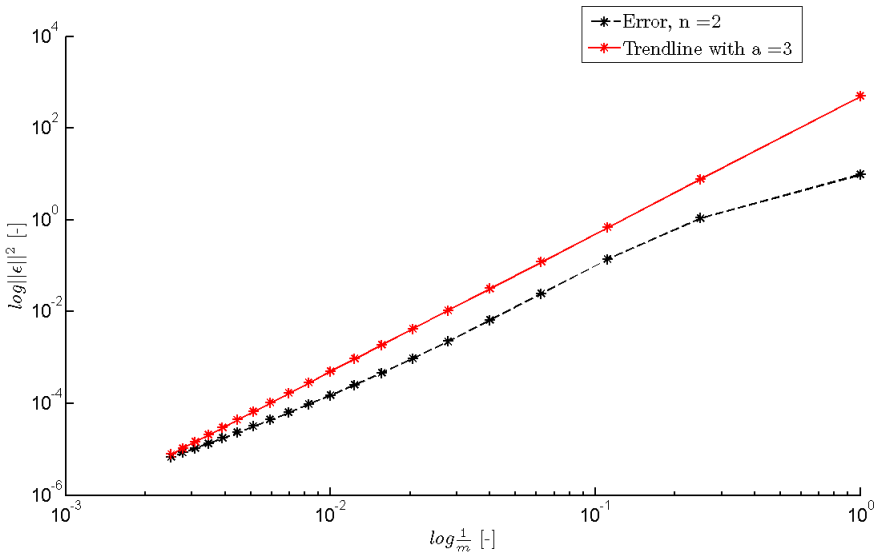


Figure 5.12: Error convergence for u_2, $N = 2$

Chapter 6

Free Surface Flow Tracking

Free surface Flows appear all around us. Think of tidal movement of the oceans, storms, ships in the water, rivers, waves and many more. All of these have one thing in common: two or more media interact. In physics a free surface is the surface of a body that is subject to neither perpendicular normal stress nor parallel shear stress such as the boundary between two homogenous fluids. Unlike liquids, gasses can not form a free surface on their own [32]. To be able to have a free surface or interface, there need to be two fluids of a different kind, e.g. different flow properties or phase. They should therefore have different density ρ, pressure p and even temperature T. When the flow system is at rest or in an equilibrium position the interface will not move and then the interface of the free surface can be easily determined. Mostly however the free surface is driven by a force. This could be wind, a current, or just simply a gravity force. For now the last possibility is taken as an example for the remainder of the report (Figure 6.1).

Especially when a fluid is placed in a gravitational field, as on earth, the fluid will always form a free surface positioned in such a way that the gravitational force will be perpendicular to the surface. Thus an equilibrium position is reached when the gravity vector with the normal of the free surface are aligned [32].

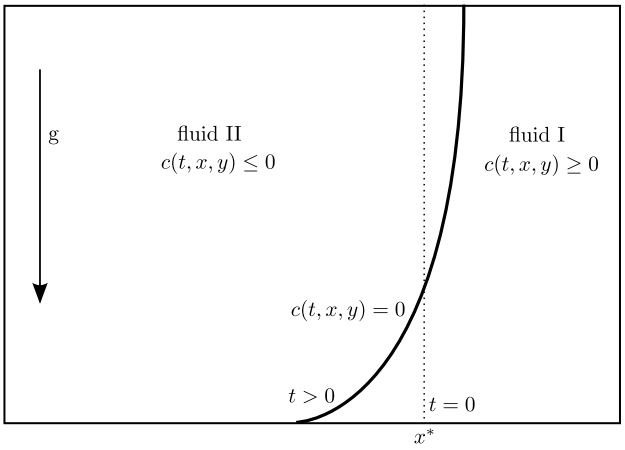


Figure 6.1: Flow Field with two fluids I and II seperated by the free surface $c(t,x,y) = 0$

6.1 Fluid Interface Capturing

In computational fluid dynamics the tools for capturing a flow phenomenon are limited. It will usually consist of flow equations in the form of differential equations and a computational domain on which a grid will be expanded, on which a solver (e.g. LSQSEM) can be released on the flow equations. Merely with these three the work has to be performed in a satisfying manner.

Simulating a similar flow problem in an experimental setup, one can use the eye, color differences, cameras, sensors, lasers etc., but then that is less challenging (PIV is not always the solution).

Assuming that there will be a free surface like depicted in Figure 6.1 then the surface is described by a tracer function $c(t, x, y) = 0$. The tracer function itself can be evaluated on all points in the flow domain. The time dependence is introduced when the surface starts moving, obviously the level of $c(t, x, y)$ will change.

When one knows the solution of the tracer function $c(t, x, y)$, the separation boundary between fluid I and II can thus be found by searching for zero values, one knows then where the fluids resides in space. For the trace function it must hold that when moving within fluid I or II that,

$$\begin{array}{ll} \text{fluid I:} & c(t, x, y) \geq 0 \\ \text{fluid II:} & c(t, x, y) \leq 0. \end{array} \tag{6.1}$$

An other approach of knowing where one resides in the computational domain is for example,

$$\begin{array}{ll} \text{fluid I:} & c(t, x, y) = 1 \\ \text{fluid II:} & c(t, x, y) = 0, \end{array} \tag{6.2}$$

but this has the effect that at the the free surface contour a discontinuity is imposed. Therefore the method described by (6.1) seems to be the favorable one as this implies a linear transition between the fluids.

Performing numerical simulations of two fluid flows with moving free surfaces imposes two problems. One is keeping track of the current location of the surface at a specific time. Next it is a challenge to compute the new position at a next time step. This advancing in time introduces the time dependence of the free surface. Updating the free surface position per new time step requires an iterative numerical scheme.

6.2 Mesh Generation

Before starting a numerical simulation of the flow problem a choice has to be made on the type of computational mesh to be used. The choice depends on the problem considered.

6.2.1 Moving Mesh

The latter section indicated clear that the whole flow field can be described as a function of the tracer function of the free surface flow $c(t, x, y)$, by evaluating it in all nodes. Intuitively one would suggest therefore to chose a computational mesh that adapts to the free surface, as depicted in Figure 6.2. That would be aligning the cells of the mesh orthogonal with respect to the surface. Accuracy can be expected doing it this way, but the great disadvantage is that for every time step the mesh has to be restructured to meet the free surface boundary again. This procedure will be costly in cpu-time and strongly distorted elements can be the result. A further problem exist with a dynamic mesh when possible waves would roll over and collapse. In this case the cells would collapse as well and intersect, which is not possible [29].

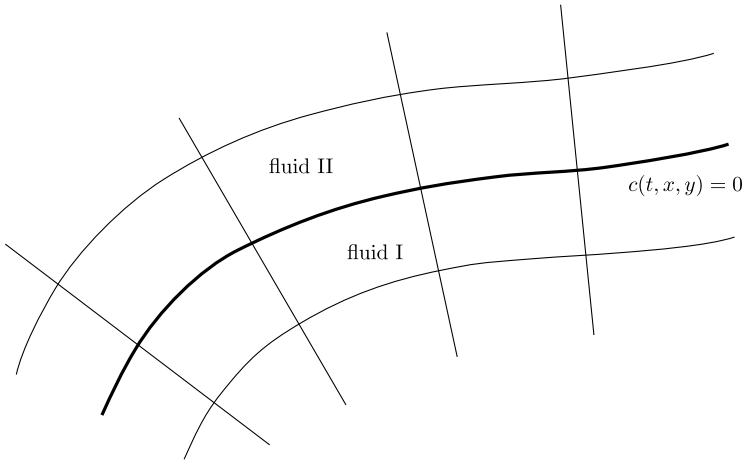


Figure 6.2: Moving mesh adapting to the free surface contour $c(t, x, y) = 0$

6.2.2 Fixed Mesh

A fixed or static mesh will not move with the free surface, see Figure 6.3. The meshes can be structured or unstructured according to preference or the problem definition. The amount of elements has to be chosen in advance and with this the calculation will be performed. The free surface moves within the elements and thus will spare re-meshing after every time step. This is more interesting, clearly because it is easier to implement when programming and it is fast concerning cpu-times. In addition one does not have to worry about cells that deform badly, or cells that collapse. For the course of this chapter the cells are chosen to be fixed. [29]

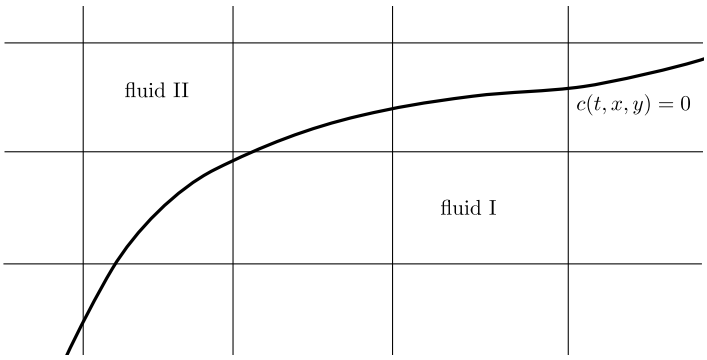


Figure 6.3: Fixed mesh to compute the 2D free surface $c(t, x, y)$

Chapter 7

Solving the Tracer Function for the Free-Surface

For numerical calculations one can not get around the fact that the flow equations have to be discretised. The time dependent tracer function is no exception. The tracking definition will be according to the description in the previous Chapter (6.1). In general the tracer function of an arbitrary free surface can be described by the following convective differential equation [2, 34, 29]:

$$\frac{\partial c(t,x,y)}{\partial t} + (\vec{u}, \nabla)\, c(t,x,y) = 0, \tag{7.1}$$

with $c(t,x,y)$ describing the movement of the free surface in time, subject to the velocity field $\vec{u} = (u,v)$ at time t or,

$$\frac{\partial c(t,x,y)}{\partial t} + u\frac{\partial c(t,x,y)}{\partial x} + v\frac{\partial c(t,x,y)}{\partial y} = 0. \tag{7.2}$$

Previously the spatial discretisation was performed with the spectral elements and least squares minimization to form the LSQSEM, see Chapter 2. The spatial discretisation in equation 7.1 will therefore be done with the LSQSEM. But there is also time dependency that has to be discretised. This does not need to be spectral elements as well, merely a faster and easier to implement scheme could be used. Among these time schemes there are endless possibilities. Well known ones are Backward Euler (explicit), or higher order schemes like Adam-Bashforth2 (explicit), Adam-Moulton (implicit) and many other or others.

7.1 Time-Stepping formulations

Time stepping is of importance when introducing time dependance to the problem. The previous chapters where concentrating entirely on spatial discretisation. Assume a system of an ordinary differential equation:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{F}(\mathbf{u}). \tag{7.3}$$

There are different possibilities to solve equation (7.3) with time-stepping methods. One can use one-step-θ like Euler and Crank-Nicolson or multi-step methods like Adam-Bashforth and Adam-Moulton [6, 19].

7.1.1 One-Step-θ Schemes

The one-step-θ schemes group together time-stepping schemes with θ as a dependent variable. In this way multiple time schemes are summarized in one generic formula i.e.,

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = (1-\theta)\mathbf{F}(\mathbf{u}^n) + \theta\mathbf{F}(\mathbf{u}^{n+1}) \tag{7.4}$$

Taking different values for θ will simplify into different one-step schemes.

Explicit Euler by setting $\theta = 0$, (7.4) gives,

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \mathbf{F}(\mathbf{u}^n) \tag{7.5}$$

Implicit Euler by setting $\theta = 1$, (7.4) gives,

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \mathbf{F}(\mathbf{u}^{n+1}) \tag{7.6}$$

Crank-Nicolson by setting $\theta = \frac{1}{2}$, (7.4) gives,

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \frac{1}{2}\mathbf{F}(\mathbf{u}^n) + \frac{1}{2}\mathbf{F}(\mathbf{u}^{n+1}) \tag{7.7}$$

These are the most common one-step time schemes but others are possible as well.

7.1.2 Adams Forward Multi-Step Schemes

The Adams Family consist of Adam-Bashforth and Adam-Moulton and can both be summarized into,

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \sum_{q=0}^{J-1} \beta_q \mathbf{F}(\mathbf{u}^{n+1-q}) \tag{7.8}$$

Adam-Bashforth is an explicit scheme that can be obtained by setting $\beta_0 = 0$ and setting the order of integration by changing J accordingly. Taking order $J = 1$ is equal to the explicit Euler scheme.

Taking $\beta_0 \neq 0$ gives the implicit Adams-Moulton family. Again order of integration can be chosen to adapt the problem considered. An order $J = 1$ will return implicit Euler and $J = 2$ the Crank-Nicolson scheme. It can therefore be concluded that Crank-Nicolson is second order accurate although it is a one-step time integration scheme. Therefore Crank-Nicolson is an attractive scheme to use because a one-step scheme requires only the storage of the previous time-step. Using increasing orders for Adam-Bashforth reduces the stability region compared to implicit Euler and thus smaller time-steps are needed to keep the scheme from blowing up. This fact turn makes higher order schemes costly to use.

7.2 Space and Time Discretisation of the convective Free Surface equation

7.2.1 The 1D case

Equation (7.2) in 1D will reduce to;

$$\frac{\partial c(t,x)}{\partial t} + u(t,x)\frac{\partial c(t,x)}{\partial x} = 0, \tag{7.9}$$

the contour $c(t,x)$ of the FS can be discretised with spectral elements with a truncated series solution as discussed in chapter 2;

$$c(x,t) = \sum_{i=0}^{N} \hat{c}_i(t)\phi_i(x) \tag{7.10}$$

The advantage of this is that $\hat{c}_i(t)$ is only time dependent as the constants will chance only per new time-step. Likewise, $\phi_i(x)$ is only dependent on the spatial coordinate, x. Discretising like this gives a similar effect as separation of variables. Equation (7.1) can be integrated with respect

to space x first using LSQSEM and then with respect to time solve the semi-discreet system. Combining therefore (7.9) with (7.10) returns

$$\sum_{i=0}^{N} \frac{\partial \hat{c}_i(t)}{\partial t}\phi_i(x) + \sum_{i=0}^{N} u(t,x)\ \hat{c}_i(t)\frac{\partial \phi_i(x)}{\partial x} = 0. \tag{7.11}$$

Then discretising with respect to time with implicit Euler for simplicity reasons,

$$\sum_{i=0}^{N} \frac{\hat{c}_i^{n+1}(t) - \hat{c}_i^{n}(t)}{\Delta t}\phi_i(x) + \sum_{i=0}^{N} u(t,x)\ \hat{c}_i^{n+1}(t)\frac{\partial \phi_i(x)}{\partial x} = 0 \tag{7.12}$$

where $\hat{c}_i^{n+1}$ is the constant at the new time step, just as $\hat{c}_i^{n}$ is the solution for the current time. Re-arranging the knowns to the right hand side and unknowns to the left hand side,

$$\sum_{i=0}^{N} \frac{\hat{c}_i^{n+1}(t)}{\Delta t}\phi_i(x) + \sum_{i=0}^{N} u(t,x)\ \hat{c}_i^{n+1}(t)\frac{\partial \phi_i(x)}{\partial x} = \sum_{i=0}^{N} \frac{\hat{c}_i^{n}(t)}{\Delta t}\phi_i(x) \tag{7.13}$$

or,

$$\sum_{i=0}^{N} \hat{c}_i^{n+1}(t)\left[\frac{\phi_i(x)}{\Delta t} + u\frac{\partial \phi_i(x)}{\partial x}\right] = \sum_{i=0}^{N} \frac{\hat{c}_i^{n}(t)}{\Delta t}\phi_i(x), \tag{7.14}$$

According to least squares (Chapter 2),

$$\int_{\Omega} (\mathcal{L}v)^T(\mathcal{L}u)d\Omega = 0 \qquad \forall \quad \phi,$$

must hold, so meaning that the LS operator will look like:

$$(\mathcal{L}v)^T = \frac{\phi_j(x)}{\Delta t} + u(t,x)\frac{\partial \phi_j(x)}{\partial x}. \tag{7.15}$$

This has to be multiplied with equation 7.14 resulting in:

$$\sum_{i=0}^{N} \hat{c}_i^{n+1}(t)\left[\frac{\phi_j(x)}{\Delta t} + u(t,x)\frac{\partial \phi_j(x)}{\partial x}\right]\left[\frac{\phi_i(x)}{\Delta t} + u(t,x)\frac{\partial \phi_i(x)}{\partial x}\right] = \tag{7.16}$$

$$\sum_{i=0}^{N} \hat{c}_i^{n}(t)\left[\frac{\phi_j(x)}{\Delta t} + u(t,x)\frac{\partial \phi_j(x)}{\partial x}\right]\frac{\phi_i(x)}{\Delta t},$$

Lastly integrating with respect to the GLL quadrature (Appendix A.5.1)

$$\sum_{i=0}^{N}\sum_{p=0}^{N} \hat{c}_i^{n+1}(t)\left[\frac{\phi_j(x_p)}{\Delta t} + u(t,x)\frac{\partial \phi_j(x_p)}{\partial x}\right]\left[\frac{\phi_i(x_p)}{\Delta t} + u(t,x)\frac{\partial \phi_i(x_p)}{\partial x}\right]w_p = \tag{7.17}$$

$$\sum_{i=0}^{N}\sum_{p=0}^{N} \hat{c}_i^{n}(t)\left[\frac{\phi_j(x_p)}{\Delta t} + u(t,x)\frac{\partial \phi_j(x_p)}{\partial x}\right]\frac{\phi_i(x_p)}{\Delta t}w_p \qquad \forall \quad j$$

yields the following system:

$$\mathbf{A}\begin{bmatrix} \hat{c}_0^{n+1}(t) \\ \vdots \\ \hat{c}_N^{n+1}(t) \end{bmatrix} = \mathbf{B}\begin{bmatrix} \hat{c}_0^{n}(t) \\ \vdots \\ \hat{c}_N^{n}(t) \end{bmatrix}. \tag{7.18}$$

Taking the inverse of $\mathbf{A}$ changes the system so that the new time step can be computed:

$$\tilde{c}^{n+1} = \mathbf{A}^{-1}\mathbf{B}\ \tilde{c}^{n}. \tag{7.19}$$

7.2.2 Incorporating BC 1D

Next step is to incorporate the boundary condition. Per time step the new boundary value has to be known. As discretisation is nodal and applied to a Lagrangian framework (see Appendix 2) the solution of the constants $\hat{c}_i^n$ is the solution on the node. Say one defines the boundary value thus at the start of the domain for each time step, this would imply that:

$$\hat{c}_0^{n+1} \hat{=} BC,$$

creating a system like:

$$\left[\begin{array}{c|c} & \\ BC & I_A \\ & \end{array}\right] \left[\begin{array}{c} \hat{c}_0^{n+1} \\ \hline \\ \hat{c}_{1..N}^{n+1} \\ \\ \end{array}\right] = \left[\begin{array}{c} \\ \hline \\ I_B \\ \\ \end{array}\right] \left[\begin{array}{c} \hat{c}_0^{n} \\ \hline \\ \hat{c}_{1..N}^{n} \\ \\ \end{array}\right] \tag{7.20}$$

With BC being the boundary nodes and I_A and I_B the interior nodes. The first line can be eliminated and the reduced system looks then as follows:

$$\left[\begin{array}{c} \\ \hat{c}_{1..N}^{n+1} \\ \\ \end{array}\right] = \left[\begin{array}{c} \\ I_A^{-1} \\ \\ \end{array}\right] \left[\begin{array}{c} \\ I_B \hat{c}_{1..N}^{n} - BC\hat{c}_0^{n+1} \\ \\ \end{array}\right]. \tag{7.21}$$

It is now possible to solve the reduced system to find the new time step solution. Iteration of this procedure will yield the flow field solution at every time step.

7.2.3 The 2D case

The 1D case was an interesting starting point in investigating the problem definition. However a step further to the real world, 2D solutions are trivial. Expanding the convective differential equation of the tracer function from equation (7.2) means also changing the spectral discretisation by one more basis function in y-direction compared to the 1D case, meaning

$$c(t,x,y) = \sum_{i=0}^{N}\sum_{j=0}^{N} \hat{c}_{ij}(t)\phi_i(x)\phi_j(y). \tag{7.22}$$

This is the new truncated series solution expanded on a Lagrangian basis. Note again that $\hat{c}_{ij}(t)$ is the time dependent parameter as this value will change when advancing in time. Similarly as in 1D section 7.2 the problem can be split in a spatial and time dependent integration as the spectral element discretisation yields a variable separating characteristic. The time integration can be performed again with schemes like Euler, Crank-Nicolson, Adam-Bashforth or Adam-Moulton etc Section 7.1.1.
Discretising the domain means that equation (7.22) will have to be substituted with equation (7.2) yielding,

$$\sum_{i=0}^{N}\sum_{j=0}^{N} \frac{\partial \hat{c}_{ij}(t)}{\partial t}\phi_i(x)\phi_j(y) + \sum_{i=0}^{N}\sum_{j=0}^{N} u(t,x,y)\ \hat{c}_{ij}(t)\frac{\partial \phi_i(x)}{\partial x}\phi_j(y) + \tag{7.23}$$

$$\sum_{i=0}^{N}\sum_{j=0}^{N} v(t,x,y)\ \hat{c}_{ij}(t)\frac{\partial \phi_j(y)}{\partial y}\phi_i(x) = 0,$$

Then discretising with respect to time with backward (implicit) Euler,

$$\sum_{i=0}^{N}\sum_{j=0}^{N} \frac{\hat{c}_{ij}^{n+1}(t) - \hat{c}_{ij}^{n}(t)}{\Delta t}\phi_i(x)\phi_j(y) + \tag{7.24}$$

$$\sum_{i=0}^{N}\sum_{j=0}^{N} \left[u(t,x,y)\ \hat{c}_{ij}^{n+1}(t)\frac{\partial \phi_i(x)}{\partial x}\phi_j(y) + v(t,x,y)\ \hat{c}_{ij}^{n+1}(t)\frac{\partial \phi_j(y)}{\partial y}\phi_i(x)\right] = 0$$

where again $\hat{c}_i^{n+1}$ is the constant at the new time step, just as $\hat{c}_i^n$ is the solution for the current time, then,

$$\sum_{i=0}^{N}\sum_{j=0}^{N}\hat{c}_{ij}^{n+1}(t)\left[\frac{\phi_i(x)\phi_j(y)}{\Delta t}+u(t,x,y)\frac{\partial\phi_i(x)}{\partial x}\phi_j(y)+v(t,x,y)\frac{\partial\phi_j(y)}{\partial y}\phi_i(x)\right]= \tag{7.25}$$

$$\sum_{i=0}^{N}\sum_{j=0}^{N}\frac{\hat{c}_{ij}^{n}(t)}{\Delta t}\phi_i(x)\phi_j(y).$$

The LS operator,

$$(\mathcal{L}v)^T=\frac{\phi_k(x)\phi_l(y)}{\Delta t}+u(t,x,y)\frac{\partial\phi_k(x)}{\partial x}\phi_l(y)+v(t,x,y)\frac{\partial\phi_l(y)}{\partial y}\phi_k(x). \tag{7.26}$$

has to be multiplied with equation 7.26 resulting in:

$$\sum_{i=0}^{N}\sum_{j=0}^{N}\hat{c}_{ij}^{n+1}(t)\left[\frac{\phi_k(x)\phi_l(y)}{\Delta t}+u(t,x,y)\frac{\partial\phi_k(x)}{\partial x}\phi_l(y)+v(t,x,y)\frac{\partial\phi_l(y)}{\partial y}\phi_k(x)\right]$$

$$\left[\frac{\phi_i(x)\phi_j(y)}{\Delta t}+u(t,x,y)\frac{\partial\phi_i(x)}{\partial x}\phi_j(y)+v(t,x,y)\frac{\partial\phi_j(y)}{\partial y}\phi_i(x)\right]= \tag{7.27}$$

$$\sum_{i=0}^{N}\sum_{j=0}^{N}\hat{c}_{ij}^{n}(t)\left[\frac{\phi_k(x)\phi_l(y)}{\Delta t}+u(t,x,y)\frac{\partial\phi_k(x)}{\partial x}\phi_l(y)+v(t,x,y)\frac{\partial\phi_l(y)}{\partial y}\phi_k(x)\right]\frac{\phi_i(x)\phi_j(y)}{\Delta t}$$

Applying the the GLL quadrature (Appendix A.5.1)

$$\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{p=0}^{N}\sum_{q=0}^{N}\hat{c}_{ij}^{n+1}(t)\left[\frac{\phi_k(x)\phi_l(y)}{\Delta t}+u(t,x_p,y_q)\frac{\partial\phi_k(x_p)}{\partial x}\phi_l(y_q)+v(t,x_p,y_q)\frac{\partial\phi_l(y_q)}{\partial y}\phi_k(x_p)\right]$$

$$\left[\frac{\phi_i(x_p)\phi_j(y_q)}{\Delta t}+u(t,x_p,y_q)\frac{\partial\phi_i(x_p)}{\partial x}\phi_j(y_q)+v(t,x_p,y_q)\frac{\partial\phi_j(y_q)}{\partial y}\phi_i(x_p)\right]w_pw_q= \tag{7.28}$$

$$\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{p=0}^{N}\sum_{q=0}^{N}\hat{c}_{ij}^{n}(t)\left[\frac{\phi_k(x)\phi_l(y)}{\Delta t}+u(t,x_p,y_q)\frac{\partial\phi_k(x_p)}{\partial x}\phi_l(y_q)+v(t,x_p,y_q)\frac{\partial\phi_l(y_q)}{\partial y}\phi_k(x_p)\right]$$

$$\frac{\phi_i(x_p)\phi_j(y_q)}{\Delta t}w_pw_q$$

The result can be written in a matrix system similar as described in 1D (7.18), except that the matrices **A** and **B** will be larger.

7.2.4 Incorporating BC 2D

As mentioned the structure of the 2D time dependent system resulting from equation (8.10) is similar to the 1D case (7.18) so also the boundary condition implementation will be similar. The constants are described as a matrix c_{ij}, meaning inevitably that it has to be resorted into a vector to solve the linear system. The indexing of the time dependent vectors will be sorted according to Chapter 3.2. The newly sorted matrices **A** and **B** can then be reduced and the system solved.

Chapter 8

Application: Time Dependent Free Surface Tracking using the LSQSEM

Finally, after a lot of mathematics in the forgoing chapters is it now possible to join all the building blocks together to solve a flow problem. The building blocks include the LSQSEM, Poisson equation and the time dependent Free surface tracer function. Additional characteristics of the flow will be added in this chapter. An overview of the solving procedure is summarized in Section 8.3

8.1 Problem Definition: Gravity Driven Free Surface Flow

Figure 8.1 shows a graphical representation of the flow problem, where an initially excited surface with $h(t,x)$ at $t=0$ is released so that fluid I starts to oscillate in the box due to the presence of a gravity force g. For simplicity fluid II is taken to be vacuum. Fluid I on the other hand will be subject to a potential field φ where density ρ_{fluidI} and p_{fluidI} are the dominating flow properties. Having fluid II as a vacuum and fluid I a fluid makes the solving easier, because the velocity field in the entire domain is only dependent on the movement of fluid I.

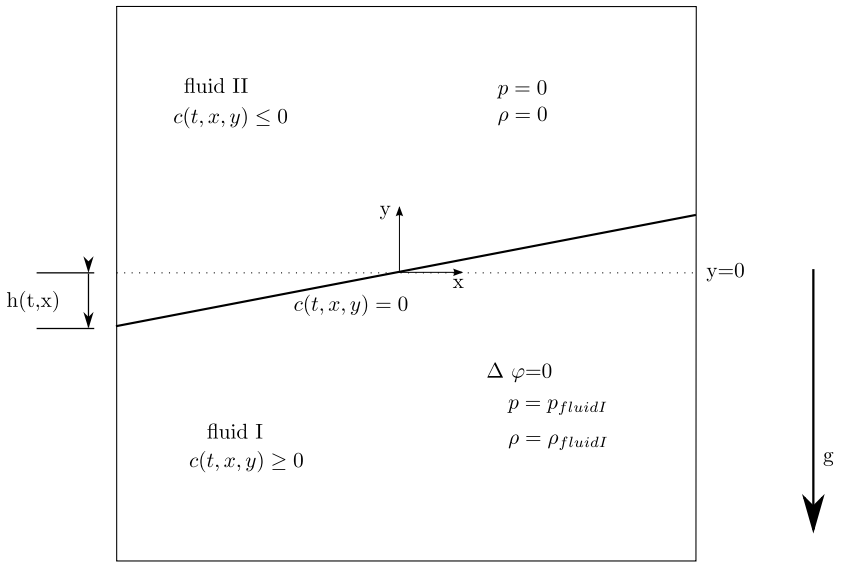


Figure 8.1: Problem definition of an initially excited free surface

8.1.1 Flow Properties

Potential Flow, Irrotanial Inviscid Flow

Recall that the Poisson is defined by taking the Laplacian of the the potential field $\Delta\varphi = f$ (4.2) [28],

$$\vec{u} = \nabla\varphi,$$
$$\nabla \cdot \vec{u} = f.$$

For this flow problem $\Delta\varphi = 0$ is chosen and as conservation of mass must be fulfilled:

$$\nabla \cdot \vec{u} = 0, \tag{8.1}$$

making the righthand-side $f = 0$. Further is the flow to be assumed irrotational so that,

$$\nabla \times \vec{u} = 0. \tag{8.2}$$

The flow is subject to the Eulerian momentum equation, neglecting viscosity:

$$\frac{\partial u}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} + \frac{p}{\rho} = 0. \tag{8.3}$$

Using the vector identity,

$$(\vec{u} \cdot \nabla)\vec{u} = \frac{1}{2}\nabla\vec{u}^2 - \vec{u} \times (\nabla \times \vec{u}), \tag{8.4}$$

and substituting (8.2) into equation (8.3) yielding,

$$\nabla\frac{\partial\varphi}{\partial t} + \frac{1}{2}\nabla(\nabla\varphi)^2 + \frac{\nabla p}{\rho} = 0, \tag{8.5}$$

taking out ∇ gives the momentum equation,

$$\frac{\partial\varphi}{\partial t} + \frac{1}{2}(\nabla\varphi)^2 + \frac{p}{\rho} = const, \tag{8.6}$$

and together with conservation of mass,

$$\Delta\varphi = 0$$

define the physical basis for calculations.

Pressure

Pressure depends on the gravity force and thus on the height of the water column, then,

$$p = p_0 + g\rho y \tag{8.7}$$

substituting in (8.6) gives

$$\rho\frac{\partial\varphi}{\partial t} + \rho\frac{1}{2}(\nabla\varphi)^2 + p_0 + g\rho y = const, \tag{8.8}$$

With $h(t, x)$ being the excitation of the free surface with respect to the x-axis, as depicted in Figure 8.1.

8.1.2 Time-Stepping and Spatial Discretisation

At this point a time-stepping scheme has to be chosen that fits to the flow problem. Section 7.1.1 makes clear that Crank-Nicolson seems to be a good trade-off. It is a one-step scheme and therefore relatively cheap in terms of CPU time because only the previous time has to be stored. Further it is second order accurate, so that leaves some leverage in case higher order integration would be needed to increase accuracy of the computation. As the higher order accuracy comes 'for free' Crank-Nicolson will be used in the remainder of the flow problem. Recall that (7.7) states:

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \frac{1}{2}\mathbf{F}(\mathbf{u}^n) + \frac{1}{2}\mathbf{F}(\mathbf{u}^{n+1}),$$

Then combining the Crank-Nicolson time-stepping and LSQSEM spatial discretisation of the free surface contour equation (7.2):

$$\frac{\partial c(t,x,y)}{\partial t} + u\frac{\partial c(t,x,y)}{\partial x} + v\frac{\partial c(t,x,y)}{\partial y} = 0,$$

will look like:

$$\sum_{i=0}^{N}\sum_{j=0}^{N} \frac{\hat{c}_{ij}^{n+1}(t) - \hat{c}_{ij}^{n}(t)}{\Delta t}\phi_i(x)\phi_j(y) +$$

$$\sum_{i=0}^{N}\sum_{j=0}^{N}\left[\frac{1}{2}u(t,x,y)\ (\hat{c}_{ij}^{n+1}(t) + \hat{c}_{ij}^{n}(t))\frac{\partial\phi_i(x)}{\partial x}\phi_j(y) + \frac{1}{2}v(t,x,y)\ (\hat{c}_{ij}^{n+1}(t) + \hat{c}_{ij}^{n}(t))\frac{\partial\phi_j(y)}{\partial y}\phi_i(x)\right] = 0 \tag{8.9}$$

Further derivation of (8.9) will be left to the reader as an elaborated discretisation of an Euler-LSQSEM space-time derivation of the said equation is performed in Chapter 7.2.3. Merely the end result looks like,

$$\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{p=0}^{N}\sum_{q=0}^{N}\hat{c}_{ij}^{n+1}(t)\left[\frac{\phi_k(x)\phi_l(y)}{\Delta t} + \frac{1}{2}u(t,x_p,y_q)\frac{\partial\phi_k(x_p)}{\partial x}\phi_l(y_q) + \frac{1}{2}v(t,x_p,y_q)\frac{\partial\phi_l(y_q)}{\partial y}\phi_k(x_p)\right]$$

$$\left[\frac{\phi_i(x_p)\phi_j(y_q)}{\Delta t} + \frac{1}{2}u(t,x_p,y_q)\frac{\partial\phi_i(x_p)}{\partial x}\phi_j(y_q) + \frac{1}{2}v(t,x_p,y_q)\frac{\partial\phi_j(y_q)}{\partial y}\phi_i(x_p)\right]w_p w_q = \tag{8.10}$$

$$\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{p=0}^{N}\sum_{q=0}^{N}\hat{c}_{ij}^{n}(t)\left[\frac{\phi_k(x)\phi_l(y)}{\Delta t} - \frac{1}{2}u(t,x_p,y_q)\frac{\partial\phi_k(x_p)}{\partial x}\phi_l(y_q) - \frac{1}{2}v(t,x_p,y_q)\frac{\partial\phi_l(y_q)}{\partial y}\phi_k(x_p)\right]$$

$$\left[\frac{\phi_i(x_p)\phi_j(y_q)}{\Delta t} + \frac{1}{2}u(t,x_p,y_q)\frac{\partial\phi_i(x_p)}{\partial x}\phi_j(y_q) + \frac{1}{2}v(t,x_p,y_q)\frac{\partial\phi_j(y_q)}{\partial y}\phi_i(x_p)\right]w_p w_q$$

8.2 Setting up the Boundary and Initial Condition

The problem can be set up so boundary and initial conditions are needed to get the flow started and flowing. The boundary conditions will be implemented using a 'weak' formulation and this will be discussed elaborately in the course of this section.

8.2.1 Initial Conditions

The initial condition is defined by the tracer function $c(t,x,y)$ described in Chapter 6. It was opted for a tracer function with smooth linear transition between negative and positive values of the tracer function $c(t,x,y)$. At the positions in the domain where $c(t,x,y) = 0$, the free surface is found. Figure 8.2 shows clearly the surface at $t = 0$. The tracer function can be described with an arbitrary linear function as:

$$c(0,x,y) = -ay + bx \qquad \text{for} \quad -1 \le x \le 1 \tag{8.11}$$

With a determining the slope of the tracer plane and b defining the hight of the initial exitations. The velocity potential is initially $\varphi^0 = 0$, hence the velocities in the system are zero as well, this makes sense as the flow is still at rest.

8.2.2 Implementation of Weak Boundary Conditions

Until now only 'strong' implementation of boundary conditions have been used (Section 7.2.4). The advantage is that the prescribed values can be eliminated from the system and taken to the righthand-side of the problem. This reduces the system for faster computing and guarantees that the boundary conditions are met exactly. However in a big and complicated system like the one discussed in this chapter it can happen that not all the boundary conditions can be met in the nodes.

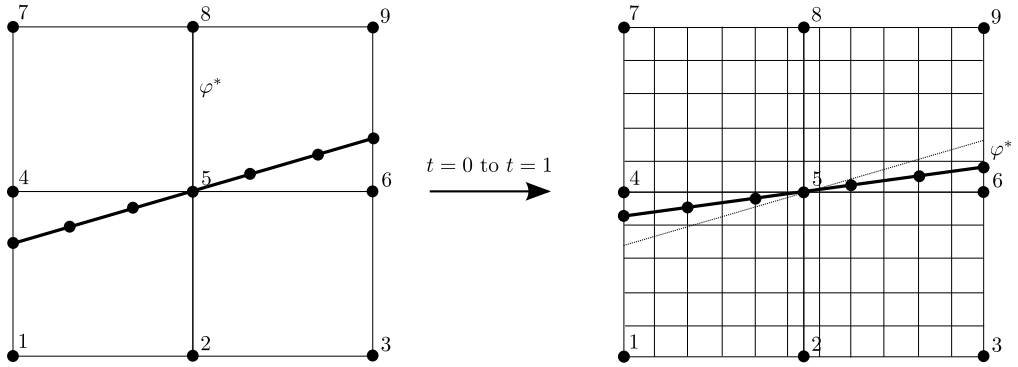


Figure 8.2: Free surface at $t = 0$, nodal to $t = 1$ interpolated

For example the potential on the surface boundary will have to be interpolated as it is unlikely that the surface will always go through any integration points and if so, most likely too little boundary conditions can be met, leaving the system under determined. The weak implementation ensures a coupling between interpolation and nodes and also ensures that all boundary condition will be met in the least squares sense. So if a boundary condition can not be met exactly the system will try to get as close as possible and still satisfy other boundary conditions as well [15]. Equation (3.7) concludes with the LSQSEM system

$$\int_{\Omega} (\mathcal{L}v)^T (\mathcal{L}u - f) d\Omega = 0$$

and extending it to multiple elements as done in Appendix A.4.1,

$$\sum_{e} \int_{\Omega^e} (\mathcal{L}v)^T (\mathcal{L}u - f) d\Omega^e = 0 \tag{8.12}$$

with

$$u^e = \sum_{i} u_i^e \phi_i(x) \tag{8.13}$$

and where $\phi_i(x)$ are basis functions that span the subspace Ω^e. Applying the Gauss-Lobatto Quadrature to the above gives:

$$\sum_{e} \left[\sum_{i} u_i^e \sum_{p} (\mathcal{L}\phi_i(x_p))^T (\mathcal{L}\phi_j(x_p)) w_p \right] = \sum_{e} \sum_{p} f(x_p) \mathcal{L}\phi_i(x_p))^T w_p \tag{8.14}$$

with x_p being the GLL zero and w_p the corresponding GLL weight. Hoitinga et. al. [15] summarized the last equation by defining in each element the matrix $(\mathbf{A}^e)_{pi}$:

$$(\mathbf{A}^e)_{pi} = \mathcal{L}\phi_i(x_p) \tag{8.15}$$

and a seperate weigh matrix $\mathbf{W}^e$

$$(\mathbf{W}^e)_{pp} = \mathbf{w}_p \tag{8.16}$$

into this:

$$\sum_{e} \left[(\mathbf{A}^e)^T \mathbf{W} \mathbf{B}^e \right] u^e = \sum_{e} \left[(\mathbf{A}^e)^T \mathbf{W} \mathbf{F} \right], \tag{8.17}$$

where the vector $\mathbf{F}$ contains the elements $(\mathbf{F})_p = f(x_p)$. So imposing weak boundary conditions will now be simply adding rows below matrix $\mathbf{A}^e$ instead of elimininating as previously done;

$$\tilde{\mathbf{A}}^e = \left[\begin{array}{c} \mathbf{A}^e \\ \hline \\ \end{array}\right] \leftarrow \text{extra BC relations} \quad . \tag{8.18}$$

This means that also the weight matrix $\mathbf{W}^e$ will have to be extended so that it will look like

$$\tilde{\mathbf{W}}^e = \left[\begin{array}{c|c} \mathbf{W}^e & \\ \hline & \alpha I \end{array}\right], \tag{8.19}$$

where α are the boundary condition weights, determining the 'hardness' of the implemented boundary condition. So a high α will return a better met boundary value. I is simply the identity matrix. What is left is now to add the know solution for the boundary value on the right hand side vector;

$$\tilde{\mathbf{F}}^e = \left[\begin{array}{c} \mathbf{F}^e \\ \hline BC \end{array}\right], \tag{8.20}$$

The same LSQSEM system as in equation (8.17) can be solved with weak boundary condition.

8.2.3 Identifying Boundary Conditions

The Poisson equation requires $4N$ (four times polynomial order) number of boundary conditions to not under or over constrain the system when using the LSQSEM. So when using an order of $N = 4$, 16 nodes or surface values should be defined as boundary conditions. The boundary conditions can be separated into two parts. For one there is the walls on which the fluid acts and secondly the free surface itself.

Walls

Calculations are done using the Poisson equation meaning that friction is not taken into account, inviscid flow. The walls are slippery, so there is no need to include the velocities parallel to the wall to the boundary conditions. Nevertheless it still applies that the fluids can not cross the wall boundary and therefore the only wall condition is that the normal velocity to the wall is zero

$$\nabla\varphi \cdot \bar{\mathbf{n}} = V_{wall} = 0, \tag{8.21}$$

shown graphically in Figure 8.3. The free surface divides the domain into two parts of roughly equal size. In this case that means the number of nodes on the walls within the fluid are approximately $\frac{1}{2}N + \frac{1}{2}N + N = 2N$, which is two half side walls and one full wall. This number will roughly stay the same, even during motion of the surface, because mass has to be conserved. When using less or more fluid of course the amount of boundary nodes will reduce or increase. During the movement of the surface it means however that one has to determine in every new time step if the node at the wall is in- or outside of the fluid, by looking at the sign of the tracer function $c(t, x, y)$. Positive was defined to be fluid (6.1), meaning that then a boundary value will apply for the corresponding

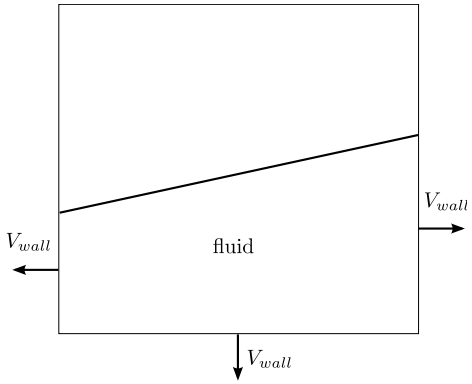


Figure 8.3: Normal Wall Boundary Condition

node and that this node will have to be imposed as weak boundary underneath the system. If the corresponding nodes defined as boundary condition are found they can be imposed in the 'weak' form so that the extra boundary condition relation beneath matrix **A** will look like,

$$\left[\begin{array}{c|c|c} 1 & 0 & 0 \end{array}\right] \left[\begin{array}{c} u_{ij} \\ \hline v_{ij} \\ \hline \varphi_{ij} \end{array}\right], \tag{8.22}$$

meaning that where there is a 1 in the system, the corresponding u_{ij} or v_{ij} is the boundary value known. On the right hand side vector **F** (8.20) a 0 is added below the system as boundary condition.

Surface

First it must hold that all point on the surface stay on the surface, which implies that the kinematic boundary condition looks like:

$$\frac{D\vec{X}}{dt} = \vec{u}. \tag{8.23}$$

such that the interface moves with the speed of the flow. This however is already incorporated in the convection equation of the tracer function (equation 7.2) as always the current velocity field will be included. The convective derivative of the velocity potential;

$$\frac{D\varphi}{dt} = \frac{\partial \varphi}{\partial t} + (\nabla \varphi)^2, \tag{8.24}$$

is also taken care for in the convection equations. Equation (8.6) can be discretised and written as,

$$\varphi^{n+1} = \varphi^n - \frac{1}{2}(\nabla \varphi^n)^2 \, \Delta t - gy \, \Delta t, \tag{8.25}$$

and holds on the free surface, but the potential φ will have to be interpolated, because it is unlikely that the surface will run through exactly the nodes at any time during movement. At a certain position on the surface $P(x^*, y^*)$, not necessary being in a node, the potential at time step n can be found interpolated as:

$$\varphi^{*\,n}(x^*, y^*) = \sum_{i=0}^{N} \sum_{j=0}^{N} \varphi_{ij}^n \phi_i(x^*) \phi(y^*). \tag{8.26}$$

The number of points to be chosen on the surface is roughly $2N$ as about $2N$ points are already been given to the boundary conditions on the walls in the previous part of this chapter. Again the ratio between the number of wall and surface boundary nodes depends on the amount of fluid.

The potential found on the refined grid as on the surface and depicted in Figure 8.2 will be described from now on at time step n as $\varphi^{*\,n}$. At $t = 0$ the system is released from the initial excitation and the free surface contour is positioned in the field so that for the new time step $t = 1$

the corresponding $\varphi^{*\,n+1}$ can be computed by using equation (8.25). The potential at the new time step $\varphi^{*\,n+1}$ is the new surface boundary condition. Therefore the solution for $\varphi^{*\,n+1}$ is put below the system on the right hand side vector **F** (equation 8.20), implemented 'weak'. On the left hand side the basis functions will be positioned so that the extra row of boundary condition relation underneath matrix **A** (8.18) looks like:

$$\left[\begin{array}{c|c|c} 0 & 0 & \phi_i(x^*,y^*)\phi_j(x^*,y^*) \end{array}\right] \left[\begin{array}{c} u_{ij} \\ \hline v_{ij} \\ \hline \varphi_{ij} \end{array}\right] = \left[\begin{array}{c} \varphi^*_{ij} \end{array}\right], \tag{8.27}$$

8.3 Solving Procedure

To get things into perspective it is probably a good idea to graphically demonstrate how the previous can be implemented in an iterative process. In Figure 8.4 one starts by inserting the initial conditions into the Poisson equation. Then the Poisson equation is solved returning u, v and φ in the nodes. Then u, v can be used for the Crank-Nicolson time integration and the tracer function $c(t, x, y)$ will be moved into the new position. For the new time step coming up new boundary condition has to be defined and so the potential φ has to interpolated and on different points on the surface the interpolated potential φ^{*n} will be found. From (8.26) φ^{*n+1} can be gathered as new 'weak' boundary condition. In addition all the nodes on the boundary have to be checked if they are residing within the fluid. If so, they will be added to the 'weak' boundary conditions as well. Now going back to the start will initiate the new iteration.

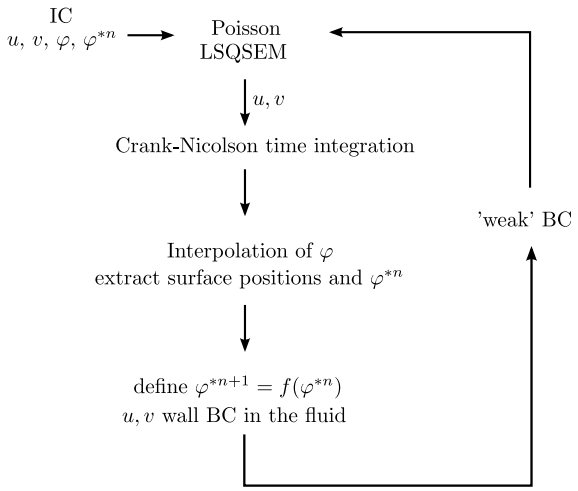


Figure 8.4: Solving scheme for free surface tracking using Crank-Nicolson and LSQSEM

Chapter 9

Results For a Gravity Driven Free Surface Flow

Chapter 8 described all the tools for setting up a basic gravity driven flow problem. That included the definition of all the boundary conditions involved and the physical system, using a Laplacian of the velocity potential. At this stage the only thing missing is seeing the system in action. Therefore presented in this chapter are the results of the gravity driven problem shown in Figure 8.1. In this way the application description in Chapter 8 will be implemented. The only force acting on the system is gravity, so no friction forces or others.

9.1 Standing Wave

The ease of this problem is that it is relatively simple to implement. It can be compared with different orders of convergence and the differences spotted. The flow is given an initial excitation of $h(0, x) = 0.1x$ and released so that gravity can do its work to get the system going. The system should wave up and down forever as friction with the walls is not incorporated due to the inviscid flow assumption. Figure 9.1 is a vector plot of the velocity vector $\vec{u}$ in the fluid, plotted for a polynomial order of $N = 8$ after the fist iteration. Clearly the velocity has a downward motion on the right hand side, where we expect gravity pulling the fluid down. Figure 9.2 shows the results of

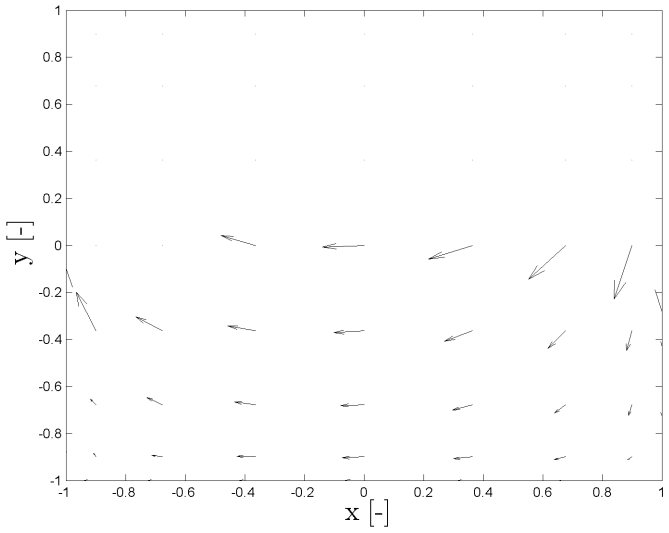
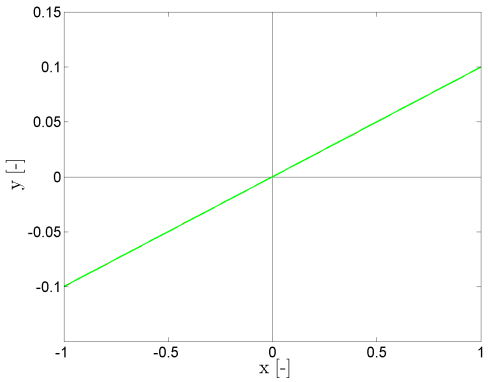


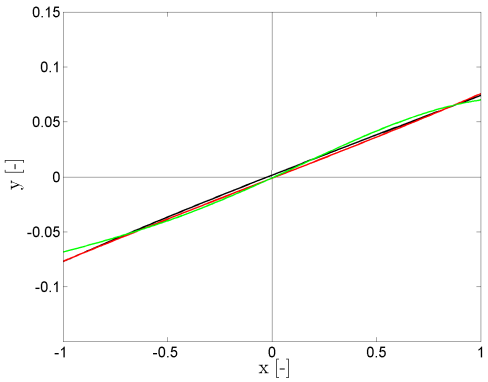
Figure 9.1: Vector plot of velocity vector $\vec{u}$ in the fluid for $N = 8$ after first iteration

using the LSQSEM method for the free surface tracking on a standard element using polynomial orders of $N = 3$ to $N = 5$ in spatial direction. The initial excitation is set to $h(0, x) = 0.1x$ (Figure 9.2(a)). The interpolation mesh, Section 8.2.1, is set to $r = 160$ cells in one spatial direction. The whole computational field is solved at one go with Poisson using a single elemental basis $m = 1$, with m the number of elements. Already at the first slice, figure 9.2(c), one detects that order $N = 5$ is starting to curve into an 's' shape. Also $N = 4$ is showing that behavior, but less than for $N = 5$. The increase in curvature is certainly no accident, it is rather expected. The solution is calculated with the LSQSEM and N being the polynomial order. At low orders like $N = 3$ the polynomials can not follow the actual solution, as the curve can not be represented properly by third order polynomials. Increasing the order increases the ability to do so. Therefore $N = 3$ looks more like a linear function oscillating around the origin. As the problem inherits the characteristics of a standing wave the node at the origin should always be met at any time. $N = 3$ has little problems with that due to a 'near' linear behavior. $N = 4$ clearly does not meet it well as Figures 9.2(d) and 9.2(e) etc demonstrate. $N = 4$ does not show origin symmetry, so that could be the reason it does not meet the origin. Consistently, $N = 5$ is meeting the origin quite well and has a more origin symmetric display.

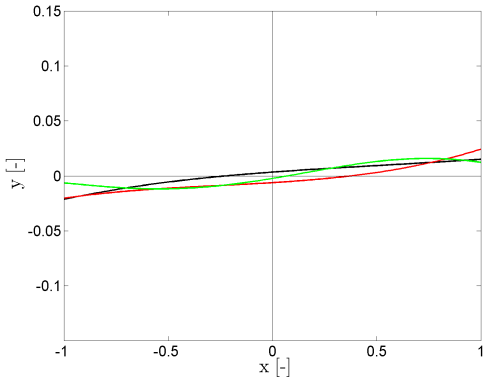
An other characteristic of the stationary wave is that the maxima and minima in the end points at $x = -1$ and $x = 1$ should be reached. The maxima and minima at these points should return the value of the initial excitation $|h(0, x)| = 0.1x$, because there can not be any damping due to viscosity. Investigating Figure 9.2(e) once more, it shows that $N = 4$ has some overshoot going beyond the initial value of $h(0, -1) = 0.1$. $N = 3$ Is meeting h well, Figure 9.2(e). $N = 5$ Can not be taken a trustworthy result as soon in time the solution explodes (figure 9.2(i)). $N = 4$ is consistent in showing overshoot, Figure 9.2(s), however $N = 3$ is damping out a little on the left hand side comparing Figure 9.2(s) and 9.2(t). During the transition at the maximum the value of $h = -0.1$ at the wall has not been met, so some damping is present. It is known that low orders suffer some numerical diffusion, so some damping can be expected.



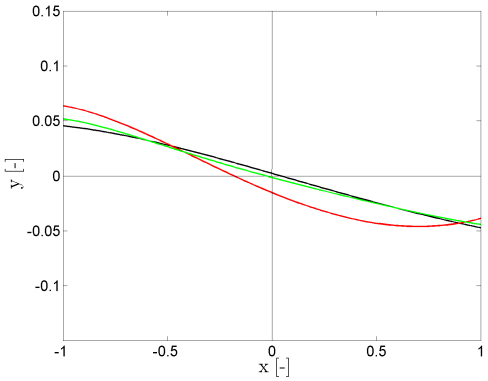
(a) t=0s



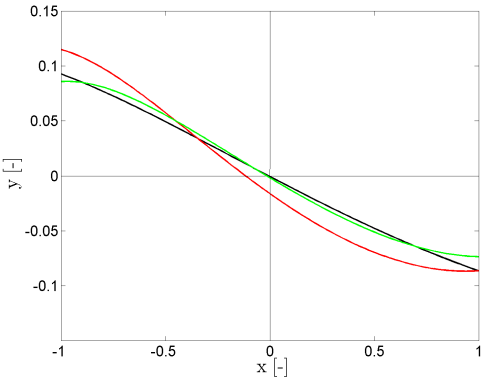
(b) t=0.19s



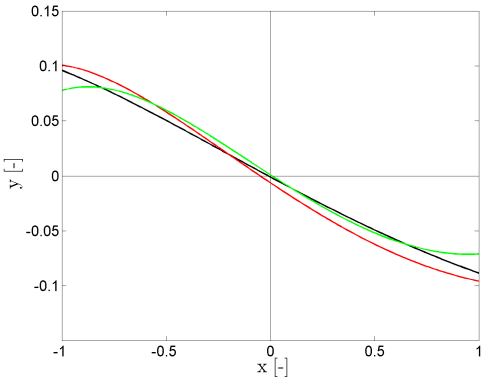
(c) t=0.38s



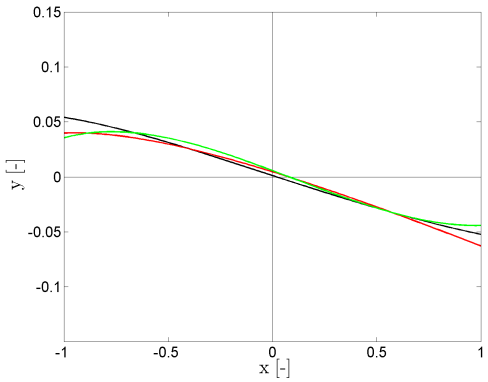
(d) t=0.56s



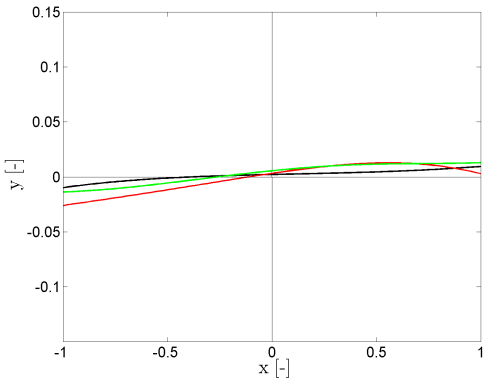
(e) t=0.75s



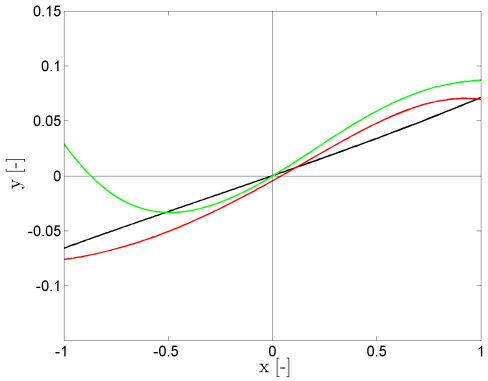
(f) t=0.94s



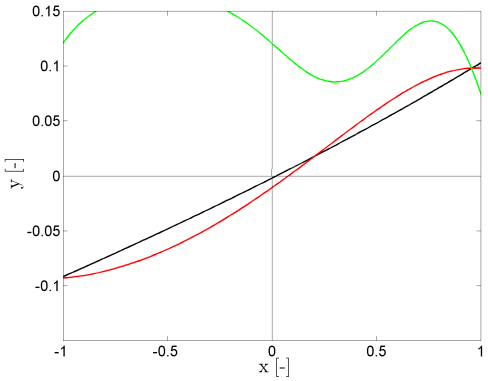
(g) t=1.13s



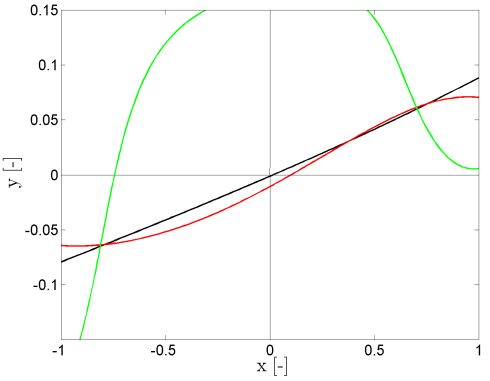
(h) t=1.33s



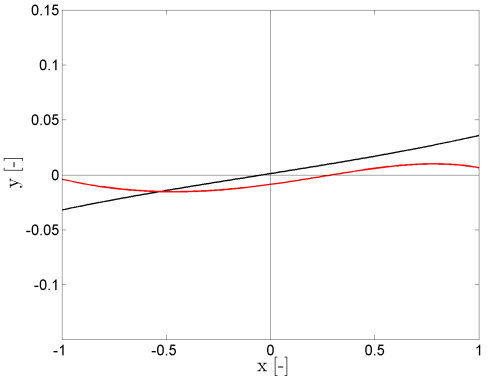
(i) t=1.50s



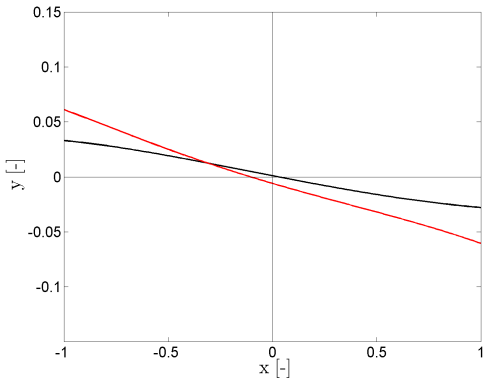
(j) t=1.69s



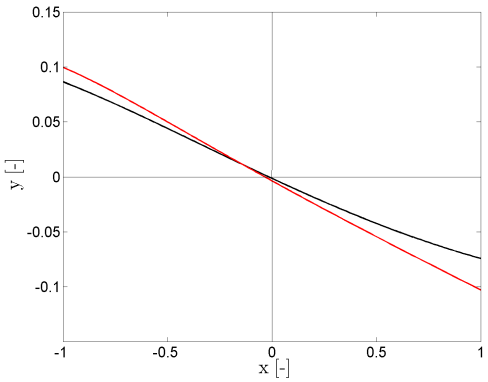
(k) t=1.88s



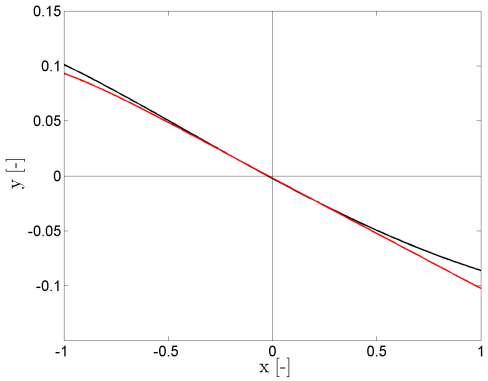
(l) t=2.06s



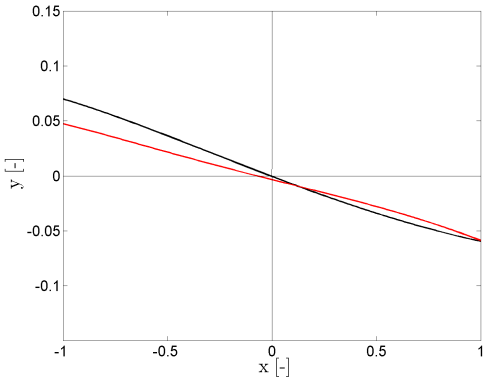
(m) t=2.25s



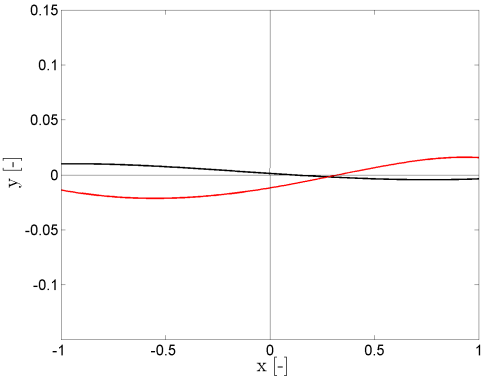
(n) t=2.44s



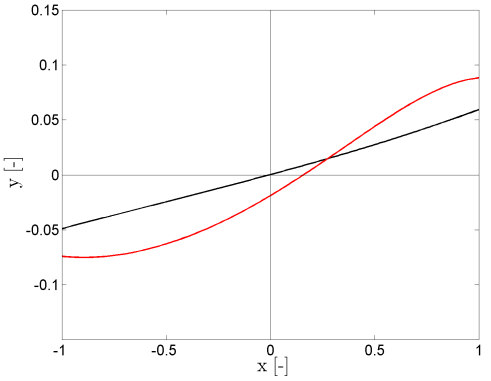
(o) t=2.63s



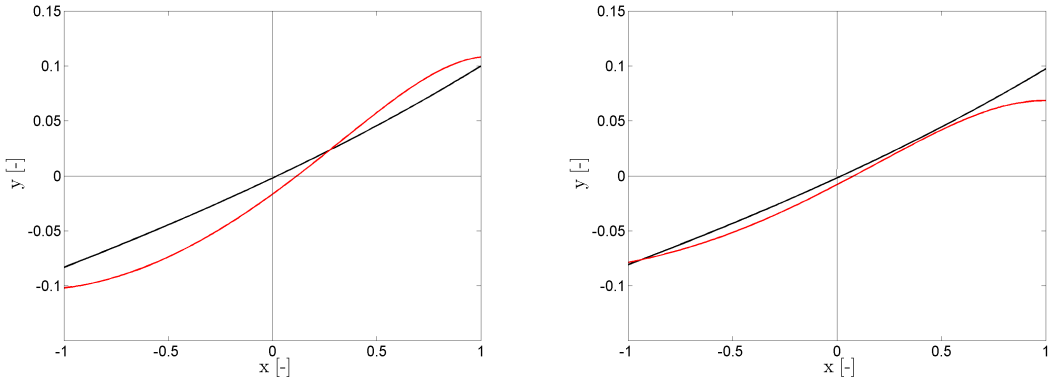
(p) t=2.81s



(q) t=3.00s



(r) t=3.19s



(s) t=3.56s (t) t=3.75s

Figure 9.2: Standing wave, N=3 'black', N=4 'red', N=5 'green'

By now one is wondering why the solution for $N = 5$ is blowing up after only a short time of iterations (Figure 9.2(i)), $t = 1.5s$. This is opposing all expectations, because the LSQSEM should be working very well at high orders. So the expectation would be increased accuracy at higher orders. There are some theories about this behavior. For one the calculations are performed on a single element. Within this element there are two regions, the fluid and the vacuum. The vacuum can not be seen as a fluid. The LSQSEM system solves the whole domain at once, for u, v and φ. The solution of u, v and φ will be zero in the vacuum and have some value within the fluid. This means that the basis functions will have difficulties to fulfill the solution in all nodes on the computational domain. The 'weak' implementation of the boundary conditions ensures that the solution will try and reach the best compromise to fulfill all boundary conditions. The LSQSEM will try and make use of the vacant area in the computational element to get rid of the polynomials in such a way that the solutions in the fluid can be met as good as possible. When using higher orders however the linear effect of low order will slowly vanish. Therefore it will be more difficult to meet the boundary conditions.

Inevitably the next step into tackling this problem is to go to a multiple elemental basis. In this way the basis functions will be able to represent the solution better. In this particular problem an interesting alternative would be introducing elements that deform to meet the free surface contour, because like this only the region in the fluid has to be solved. Everything outside of the fluid can then be neglected. For sure this will yield more accurate results.

Making the time step smaller is an other alternative. This method is generally known to work well. Table 9.1 show what happens when reducing the time step. The solution will blow up later, but the gain is marginal, being 0.1 second. Reaching computing times of 3.75 seconds, like $N = 4$, in this way is unlikely. It can therefore be concluded that this is not the reason the solution blows up.

time step	blow up:
Δt=1/160	t=1.35s
Δt=1/200	t=1.45s

Table 9.1: Convergence comparison blow up time vs. time step

Chapter 10

Conclusion

In this master thesis the emphasis is put on using the LSQSEM to track free surface flows on a standing wave problem. Using spectral elements based on orthogonal Lagrange polynomials as basis function and combining it with the least square method results into the least square spectral element method (LSQSEM). The real strength of the spectral elements is the accuracy of integration by using p-refinement and will result in a fast convergence of the integration error compared to h-refinement. Therefore a linear operator was discretised that can be applied on a variety of functions. This is the basis of setting up the discretisation of flow problems like the Poisson, Navier-Stokes or Euler.
For solving the Poisson equation using the LSQSEM one needs to set up a coupled system defined by the Laplacian of a velocity potential. The Poisson equation is tested on different kind of known function like the sine and cosine and simple polynomial functions. It showed that integration errors (convergence study) go as low as machine error using p-refinement and that the speed of convergence is faster than using classic h-refinement.
To be able to apply the LSQSEM to free surface tracking a tracer function was introduced that can be evaluated in the whole flow field. It ensures that the surface contour will be convected correctly over time. Where the tracer function is identified as being zero, i.e. $c(t, x, y) = 0$ part of the free surface is found. One of the two fluids will inherit negative tracer function values and the other positive values. The tracer function will have a smooth transition between negative and positive side of the the free surface to prevent discontinuities. A fixed mesh was chosen compared to moving mesh as this safes valuable CPU-time in re-meshing for every time step.
The solving of the free surface tracer function requires the discretisation in time direction as well as in spatial direction. Spatial direction will be solved using the LSQSEM, but in time direction a second order accurate scheme is used, Crank-Nicolson. This time discretisation is a one-step scheme that only requires the storage of the previous time step, so that the second order accuracy it inherits comes for 'free'. Valuable CPU-time can be saved.

Finally the tracer function and the Poisson equation are combined to solve a flow problem. Setting up the boundary conditions and how an iterative process can be initiated are explained. For the free surface boundary condition use was made of interpolations of the velocity potential. The interpolated values where then inserted as 'weak' boundary condition for solving the Poisson equation. The wall boundary condition are evaluated at the GLL nodes and idem used as a 'weak' boundary implementation. A standing wave problem is introduced that is initially exited from a reference hight by $h(0, x) = 1$ and then released. Only one side of the free surface is actual fluid. The other side is considered to be a vacuum. Gravity is the only driver of the problem, because friction forces are not taken into account, by taking the right hand side of the Poisson equation to be zero.

The result is that a flow can be simulated with success. On a single elemental basis an oscillating standing wave is simulated. Polynomial orders of the basis function ranging between $N = 3$ and $N = 5$ are shown. Order $N = 3$ has some linear behavior as the the third order polynomials can not represent the solution well enough yet. Increasing to $N = 4$ solves this problem partly, but the flow is still not symmetric in the y-axis as is expected. Initially order $N = 5$ complies very well

with the expectations of a standing wave, but is unfortunately not stable enough and explodes. This is most likely because the polynomials have a hard time representing the flow field, as outside of the fluid, in the vacuum, calculations are performed that do not belong in the fluid solution. The high oscillations of the high order polynomials find it difficult meeting the boundary conditions the quality of the computed results goes down.

10.1 Recommendations

To be able to run calculations on the standing wave, it is essential that the calculation basis is expanded to multiple elements. This will relief the polynomials so that the solution can be represented more accurately. Further improvements on this would be creating elements that adapt to the form of the free surface contour. In this way unnecessary inclusion of the region outside of the flow region is not needed anymore.

Bibliography

[1] N.A. Beishuizen and M.I. Gerritsma. Error estimation for the least squares spectral element method. *16th AIAA Computational Fluid Dynamics Conference*, AIAA 2003-3851, June 2003.

[2] Jean-Michel Campin, Alistair Adcroft, Chris Hill, and John Marshall. Conservation of properties in a free-surface model. *Ocean Moddeling 6 (2004) 221-244*, March 2008.

[3] C. Canuto, M.Y. Hussaini, A. Querteroni, and T.A. Zang. *Spectral Methods: Fundamentals in Single Domains*. Springer, 2006.

[4] B. de Maerschalk. Space-time least-square spectral element method for unsteady flows. Master's thesis, TU-Delft, 2003.

[5] M.O. Deville, P.F. Fischer, and E.H. Mund. *High Order Methods for Incompressible Fluid Flows*. Cambridge University Press, 2002.

[6] Dale R Durran. The third-order Adams-bashforth method: An attractive alternative to Leapfrog time differencing. *Monthly weather review*, 119, March 1991.

[7] A. Galvão, M.I. Gerritsma, and B. de Maerschalck. *hp*-adaptive least squares spectral element method for hyperbolic partial differential equations. *Journal of Computational and Applied Mathematics 215 (2008) 409-418*, 2005.

[8] M.I. Gerritsma. *Computational Fluid Dynamics, Incompressible Flows*. Delft University Press, 2002.

[9] M.I. Gerritsma, R. van der Bas, B. De Maerschalck, B. Koren, and H. Deconinck. Least-squares spectral element method applied to the Euler equations. *International Journal for Numerical Methods in Fluids*, March 2008.

[10] Haberman. *Applied Partial Differential Equations*, volume Fourth Edition. Pearson Prentice Hall, 2004.

[11] W. Heinrichs. An adaptive spectral least-squares scheme for the Burgers equation.

[12] W. Heinrichs. Least-squares spectral collocation for the NavierStokes equations. *Journal of Scientific Computing*, 21(1), August 2004.

[13] W. Hoitinga. Direct minimization of equation residuals in least squares *hp*-finite element methods. Master's thesis, Delft University of Technology, 2004.

[14] W. Hoitinga, R. de Groot, and M.I. Gerritsma. Direct minimization method of the least squares spectral element method using the block-QR decomposition.

[15] W. Hoitinga, R. de Groot, M. Kwakkel, and M.I. Gerritsma. Direct minimization of the least-squares spectral element functional part i: Direct solver. *Journal of Computational Physics 227 (2008) 24112429*, November 2007.

[16] B. Jiang. *The Least-Squares Finite Element Method*. Springer.

[17] G.E.M Karniadakis and S.J.Sherwin. *Spectral/hp Element Methods for CFD*. Cambridge University Press, 1999.

[18] A. Klawonn, O. Rheinbach, and L.F. Pavarino. Exact and inexact FETI-DP methods for spectral elements in two dimensions.

[19] M. Kwakkel. Time dependent flow simulations using the least-square spectral element method. Master's thesis, TU-Delft, June 2007.

[20] M. Kwakkel and M.I. Gerritsma. Time dependent flow simulation using the least squares spectral element method in combination with direct minimization. *European Conference on Computational Fluid Dynamics*, 2006.

[21] B. De Maerschalck and M.I. Gerritsma. Higher-order GaussLobatto integration for non-linear hyperbolic equations. *Journal of Scientific Computing*, 27(13), June 2006.

[22] G. Oldenziel and M.I. Gerritsma. Least-squares spectral element method with implicit time integration for the inviscid Burgers equation. *European Conference on Computational Fluid Dynamics*, 2006.

[23] A.T. Patera. A spectral element method for fluid dynamics-laminar flow in a channel expansion. *Journal of Computational Physics*, 54:458-488, 1984.

[24] R. Peyret. *Spectral Methods for Incompressible Flow*, volume 148. Springer, 2002.

[25] J.P. Pontaza and J.N. Reddy. Spectral/hp least-squares finite element formulation for the navierstokes equations. *Journal of Computational Physics 190 (2003) 523549*, May 2003.

[26] M.M.J. Proot and M.I. Gerritsma. Application of the least-squares spectral element method using Chebyshev polynomials to solve the incompressible Navier-Stokes equations. *Numerical Algorithms 38-155-172*, Springer 2005.

[27] M.M.J. Proot, M.I. Gerritsma, and M. Nool. Application of least-squares spectral element in incompressible flow problems. *16th AIAA Computational Fluid Dynamics Conference*, AIAA 2003-3685, June 2003.

[28] Iain Robertson and Spencer Sherwin. Free-surface flow simulation using *hp/*spectral elements. *Journal of Computational Physics 155, 26-53*, 1999.

[29] Ruben Scardovelli and Stephane Zaleski. Direct numerical simulation of free-surface and interfacial flow. *Annu. Rev. Fluid Mech 31:567-603*, 1999.

[30] L. Timmermans. Analysis of spectral element method. Master's thesis, Eindhoven University of Technology, March 1994.

[31] unknown. *Flows with Free Surfaces. http* : $//www.damtp.cam.ac.uk/user/examples/P2Ld.pdf$, April 2009.

[32] unknown. *Free Surface. http* : $//www.absoluteastronomy.com/topics/Free_surface$, April 2009.

[33] W.R. van Dalen. Spectral methods applied to the lineair advection-reaction equation. 2006.

[34] L. White, V. Legat, and E. Dellersnijder. Tracer conservation for three-dimensional, finite-element, free-surface, ocean modeling on moving prismatic mesheshes. *Monthly weather review*, 136, Jan 2007.

Appendices

Appendix A

Spectral Elements

A.1 Standard Element

A lot of attention will be payed to the computational domain $\Omega = [-1, 1]$ as on this interval the basis functions will be orthogonal. Nevertheless the basis functions can be made orthogonal on all intervals, but doing this for just one interval allows mapping on to all intervals. For distorted domains a transformation has to be defined to make the domain fit the standard element again. Normally for the x-direction the letter ξ is used. For y-direction η will be used and ζ accordingly for z-direction where:

$$\begin{aligned} x &\rightarrow \xi: \quad -1 \leq \xi \leq 1 \\ y &\rightarrow \eta: \quad -1 \leq \eta \leq 1 \\ z &\rightarrow \zeta: \quad -1 \leq \zeta \leq 1 \end{aligned}$$

A.2 High-Order Expansion Basis (Trial) Functions

When using a linear expansion basis the only way to obtain a higher accuracy of the calculation is to increase the amount of elements, also called h-refinement. With the use of Spectral elements, contrary to linear elements, the user is now able to increase the accuracy without increasing the amount of sub-domains, by expanding the truncated series solution with higher polynomial order. This section will deal with the different types of higher order related basis functions that can be used. Also the choice for the expansion basis that will be used in the course of the thesis. Some common basis functions previously derived by others are Jacobi, Lagrange, Legendre, Chebyshev and many others. They all have the common feature that they are all orthogonal on the interval $[-1, 1]$ with respect to a given weight function.

A.2.1 Jacobi/Legendre polynomials

The Jacobi/Legendre polynomials are solutions of a singular Sturm-Liouville problem [17], $P_n^{\alpha,\beta}$ with n being the order of the polynomial. As mentioned above the feature is the orthogonality property they posess on the interval $[-1, 1]$ with respect to a given weight function [13].

To construct the polynomials the following recursive formula can be used:

$$\begin{aligned} P_0^{\alpha,\beta}(x) &= 1 \\ P_1^{\alpha,\beta}(x) &= \tfrac{1}{2}[\alpha - \beta + (\alpha + \beta + 2)] \\ a_p^1 P_{p+1}^{\alpha,\beta}(x) &= (a_p^2 + a_p^3 x) P_p^{\alpha,\beta}(x) - a_p^4 P_{p-1}^{\alpha,\beta}(x) \end{aligned} \tag{A.1}$$

With

$$\begin{aligned} a_p^1 &= 2(p+1)(p+\alpha+\beta+1)(2p+\alpha+\beta), \\ a_p^2 &= (2p+\alpha+\beta+1)(\alpha^2-\beta^2), \\ a_p^3 &= (2p+\alpha+\beta)(2p+\alpha+\beta+1)(2p+\alpha+\beta+2), \\ a_p^4 &= 2(p+\alpha)(p+\beta)(2p+\alpha+\beta+2). \end{aligned} \tag{A.2}$$

Here α and β can be chosen arbitrarily. For example choosing $\alpha = \beta$ one obtains classical symmetric polynomials. When taking $\alpha = \beta = 0$ or $\alpha = \beta = -\frac{1}{2}$ one obtains the Legendre and Chebyshev polynomials, respectively.

For the scope of this Literature review only the Legendre spectral elements will be used as they result in a unit weight function.

A.2.2 Lagrangian Expansion Basis

The expansion of the solution can be done in a Lagrangian framework, meaning on a nodal expansion basis. These nodes are chosen such that they coincide with the zeros of the Gauss-Lobatto-Legendre (GLL) points. This means that one uses the derivative of the highest Legendre Polynomial and adds two zero points, one at the start and one at the end of the computational domain $\Omega = [-1, 1]$. Then it has to be solved:

$$(1 - \xi^2)L'_N(\xi) = 0 \qquad [-1 \leq \xi \leq 1], \tag{A.3}$$

which represents the GLL polynomials. In (A.3) $L'_N(\xi)$ represents the partial derivative with respect to ξ Legendre Polynomials $L_N(\xi)$, with order N. The first five orders are:

N	$L_N(\xi)$
0	1
1	ξ
2	$\frac{1}{2}(3\xi^2 - 1)$
3	$\frac{1}{2}(5\xi^3 - 3\xi)$
4	$\frac{1}{8}(35\xi^4 - 30\xi^2 + 3)$,

(A.4)

and L_0-L_3 are depicted in figure A.1 and can be derived directly by inserting $\alpha = \beta = 0$ and the

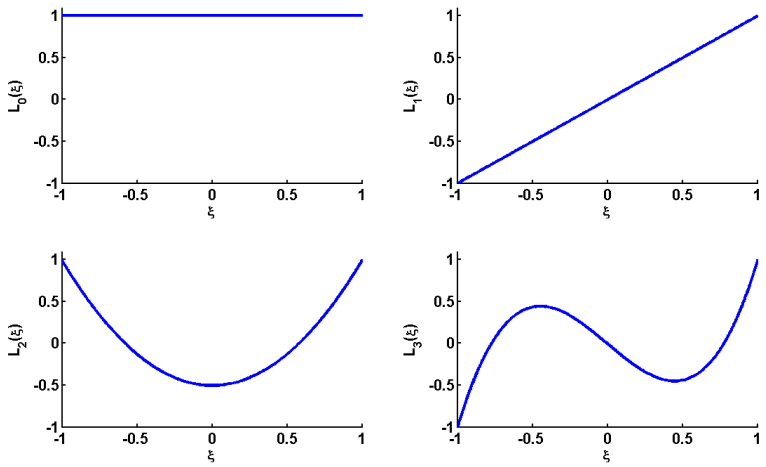


Figure A.1: Legendre polynomials L_0-L_3

order N accordingly, into equation A.1, then,

$$L_N(\xi) = P_n^{0,0}(\xi). \tag{A.5}$$

The zeros of the GLL formulation (A.3) have to be found by evaluating the said equation for every order, thus evaluating the zeros of the Legendre Polynomials and in addition the two endpoints as $(1 - \xi^2)$ will return these values. Therefore the $N + 1$ zeros (nodes) are formulated as $-1 = \xi_0, \xi_1, \cdots, \xi_N = 1$. For convenience the polynomials of Legendre $L_N(\xi)$ can be recursively represented in the form [3]:

$$L_{k+1}(\xi) = \frac{2k+1}{k+1}\xi L_k(\xi) - \frac{k}{k+1}L_{k-1}(\xi), \tag{A.6}$$

with $L_0 = 1$ and $L_1 = \xi$ and k the order of the polynomial. Then taking the derivative returns:

$$L'_{k+1}(\xi) = \frac{2k+1}{k+1}(L_k(\xi) + \xi L'_k(\xi)) - \frac{k}{k+1}L'_{k-1}(\xi) \tag{A.7}$$

with $L'_0 = 0$ and $L'_1 = 1$.

The Lagrangian basis functions can be found using N nodal points $-1 = \xi_0, \xi_1, \cdots, \xi_N = 1$ resulting in an order for the Lagrangian polynomial of $N - 1$. It holds that:

$$\phi_p(\xi_q) = \delta_{pq}, \tag{A.8}$$

thus defined by a Kronecker delta function as the Lagrangian polynomials are unity ξ_q at $p = q$ and zero at $p \neq q$ [3, 13, 17]. Then the Lagrangian base functions can be written as:

$$\phi_p(\xi) = \frac{\Pi^N_{q=0,p\neq q}(\xi - \xi_q)}{\Pi^N_{q=0,p\neq q}(\xi_p - \xi_q)}, \tag{A.9}$$

combined with the GLL points:

$$\phi_p(\xi) = -\frac{1}{N(N+1)}\frac{(1-\xi^2)L'_N(\xi)}{(\xi - \xi_p L_N(\xi_p))} \qquad p = 0, \cdots, N. \tag{A.10}$$

The new set of basis function according to the Lagrangian framework can now be used to expand the solution in a Spectral sense, i.e. replacing the linear basis functions (Fourier, etc.) to form a basis of higher order terms (A.7). The Lagrange polynomials are plotted for $p = 5$ in figure A.2. Therefore in this thesis the Lagrange polynomials are based on Legendre polynomials.

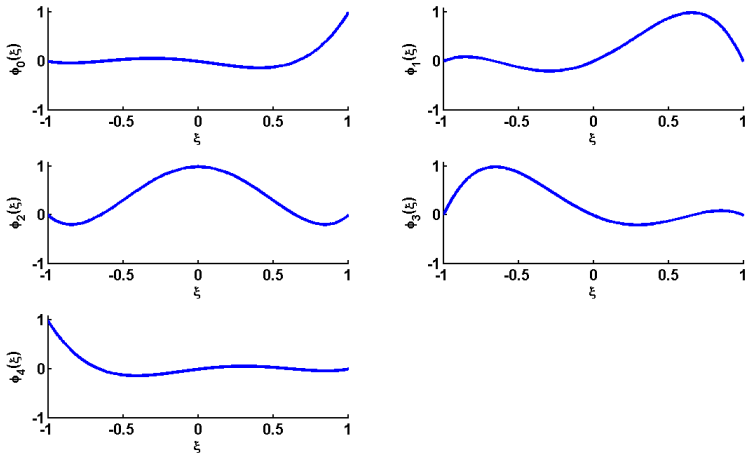


Figure A.2: Lagrange polynomials $\phi_0 - \phi_4$

A.2.3 Differentiation of the Lagrange base function

In many cases it is needed to find a differentation of basis functions. This can be for example:

$$\frac{\partial u}{\partial \xi} = \sum_{n=0}^{\infty} \hat{u}_n \frac{\partial \phi_n(\xi)}{\partial \xi} \tag{A.11}$$

where $\frac{\partial \phi_n(\xi)}{\partial \xi}$ has to be found. Next the said differentiation is also needed when using the Least Square integration method (sect. 2.2), where the test function of the Least Squares method equals

the first derivative of the basis functions. This differential matrix can be written as,

$$d_{pq} = \begin{cases} \frac{L_N(\xi_p)}{L_N(\xi_q)} \frac{-1}{\xi_p - \xi_q} & p \neq q, \\ -\frac{N(N+1)}{4} & p = q = 0, \\ \frac{N(N+1)}{4} & p = q = N, \\ 0 & \text{else.} \end{cases} \tag{A.12}$$

A.2.4 Higher dimensional expansion basis

Higher dimensional expansion basis are very similar to 1-D problems. The expansion can now be written as:

$$\begin{aligned} &2D \quad H_i(\xi,\eta) = \phi_p(\xi)\phi_q(\eta) && i = p + q(N+1) \\ &3D \quad H_i(\xi,\eta,\zeta) = \phi_p(\xi)\phi_q(\eta)\phi_r(\zeta) && i = p + q(N+1) + r(N+1)^2. \end{aligned} \tag{A.13}$$

A.3 Least Squares and Galerkin integration methods

The Least Square and Galerkin methods are ways to integrate an unknown function expanded in the Spectral Element discretization. In Appendix B a step by step implementation of the Galerkin and Least Square method combined with the Spectral Method is incorporated. In this section the fundamental mathematical basis is described of, first, the Galerkin Method and second the Least Squares Method.

A.3.1 Galerkin Method

The Galerkin method is widely used in computational fluid dynamics and belongs to the class of weighted residuals. For the basis of the theory of the Galerkin method a system with a differential equation is considered according to Jiang, [16]:

$$\begin{aligned} \mathbf{Au} &= \mathbf{f} \quad \text{in} \quad \Omega, \\ \mathbf{Bu} &= \mathbf{0} \quad \text{on} \quad \Gamma \end{aligned} \tag{A.14}$$

Where $\mathbf{A}$ is the linear differential operator, $\mathbf{B}$ is the boundary operator, $\mathbf{u}$ is the dependent unknown vector, $\mathbf{f}$ is the forcing function vector, Ω is the domain of interest and Γ represents part of the boundary of Ω. In the Galerkin Method the function $\mathbf{u}$ can be approximated by a set of unknown trial functions (or basis functions) ϕ_i expanded by a truncated series solution, where $\hat{u}_i$ are constants:

$$\mathbf{u}_{appr} = \sum_{i=0}^{N} \hat{u}_i \phi_i \qquad , i = 1, \cdots, n \tag{A.15}$$

Next, the algebraic equation resulting in a numerical solution is formed as a 'weighted residual' and can be defined as:

$$\int_\Omega \mathbf{v_i^T}(\mathbf{Au} - \mathbf{f}) d\Omega + \int_\Gamma \bar{\mathbf{v}}_\mathbf{i}^\mathbf{T} \mathbf{B} \mathbf{u} d\Gamma, \tag{A.16}$$

where $\mathbf{v_i^T}$ and $\bar{\mathbf{v}}_\mathbf{i}^\mathbf{T}$ are the transposes of 'suitable chosen' test functions. In the case of using the classic Galerkin approach usually the test functions are chosen such that:

$$\mathbf{v_i^T} = \bar{\mathbf{v}}_\mathbf{i}^\mathbf{T} = \phi_i. \tag{A.17}$$

The Galerkin method can be applied to self-adjoint operators and systems as well as to non-self-adjoined ones and is thus a good option for general solving of equations. It is applicable to problems in solid mechanics as well as fluid mechanics.

Some known concerns about the Galerkin method in practise are that the solutions to convective problem will show oscillations and 'wiggle' like behavior. This behavior can be reduced by increasing the amount of sub-domains on the computational domain [17], but this would counteract the aim for a faster solving with the help of a high order scheme with low amount of cells. Furthermore it will not behave in hight-speed compressible flow and shallow water wave problems as well as elliptic and mixed problems (Laplace) problems following the Ladyzhenskaya-Babushka-Brezzi (LBB) condition [17] as the minimization results into a difficult to solve saddle-point. According to the LBB-condition there is a distinction between primal and dual variables, where for example in the Navier-Stokes equation the pressure is considered a dual variable as whereas the velocity a primal. This means for the solution a lower order has to be taken for the pressure compared to the velocity and thus a loss in accuracy.

A.4 h/p-refinement

Spectral element methods (SEM) are based on higher order discretisation over a certain domain. This usually means that an order of more than two is used. Higher order polynomials reduces the computed error. The increase in order is also called p-refinement. Another method which is very common in first order FEM, is to increase the amount of elements on the computational domain, because this increases the accuracy as well. This type of refinement is called h-refinement. Appendix C gives a good illustration and analysis what both optimization methods mean for SEM, where first the number of cells is kept constant and polynomial order altered after which the opposite is done, keeping the polynomial order constant and increasing the amount of cells or sub-domains on the computational domain.
It has to be noted that both, h and p type refinement are closely related, because even when only a p-refinement is applied still an elemental basis is needed.

A.4.1 h-refinement

In this section the process of decomposing an expansion into an elemental contribution, that is the h-refinement, will be demonstrated.
The first step in the spectral element discretisation process to be taken is to divide the domain Ω into n_e **non-overlapping** elements denoted by sub-domains, Ω_e [7, 17, 30]. Then the domain Ω can be written as the sum of the sub-domains, hence:

$$\Omega = \cup_{e=1}^{n_e} \Omega_e \quad , \cap_{e=1}^{n_e} = \emptyset. \tag{A.18}$$

For the domain this means that if Ω is a domain of length l, $\Omega = \{x|0 < x < l\}$, a sub-domain, e.g., the e-th element can be defined like

$$\Omega_e = \{x|x_e - 1 < x < x_e\},$$

where $x_0, x_1, \cdots, x_e$ represent the $(e+1)$ nodes at the edges of the elements. Figure A.3 makes clear how division of the solution domain can be represented by the sub-elements. Furthermore it

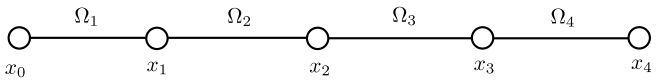


Figure A.3: Sub-domains of global domain Ω and corresponding end element nodes

must hold that the sum of all sub-domains should yield the result of the complete domain as,

$$\int_\Omega (\mathbf{A}\phi)^T(\mathbf{A}\mathbf{u} - \mathbf{f})^2 d\Omega = \sum_{e=0}^{n_e} \int_{\Omega_e} (\mathbf{A}\phi)_e^T(\mathbf{A}\mathbf{u} - \mathbf{f})_e^2 d\Omega_e = 0$$

A.4.2 p-refinement

p-Refinement is the actual strength of the SEM. The accuracy of the solution increases faster compared to the accuracy increase of a h-refinement, Appendix C. Furthermore the implementation of higher orders is relatively simple. It is based on increasing the order of the basis functions (Appendix A.2), where the order is described as the degree of freedom of the approximate solution.

A.5 Error Analysis

Nobody likes to makes errors as it is in human nature to strive for perfection. However discretising a differential equation has the property that one approximates a solution. In the case of LSQSEM the discretisation of a differential equation will have a truncated error and greatness of the error will give a validation of the strength of LSQSEM compared to Galerkin as well as the comparison between p- and h-refinement.

Intuitively an error could be defined as taking the difference between the real solution at a given point and subtract the approximated solution evaluated at the same said point. However in this section another error approach is used according to the L^2-Norm.

A.5.1 Gauss-Lobatto Quadrature

In general form Gauss-Lobatto quadrature will rewrite the integration of a function $g(x)$ as a weighted summation like [7, 3],

$$\int_{\Omega} g(x)dx \approx \sum_{p=0}^{P} g(x_p)w_p, \tag{A.19}$$

With x_p being the Gauss-Lobatto-Legendre GLL zeroes and w_p, the GLL weights,

$$w_p = \frac{2}{N(N+1)} \frac{1}{[L_N(x_p)]^2} \tag{A.20}$$

A.5.2 Error according to the L^2-Norm

Clearly, the remaining term of the series expansion (2.2) can be described as the error. If there exist a space $L^p(\Omega)$ with $1 \leq p \leq \infty$ then this space is a function space for a function u with domain Ω. If $p = 2$ this will result in a Hilbert space with, $L^2(\Omega) = H^0$, [8, 13] or

$$||u||^2_{L^2(\Omega)} = \int_{\Omega} |u|^2 d\Omega. \tag{A.21}$$

Then the error can be defined at the difference of the real and approximated solution. This means the error will look as follows:

$$\epsilon = u_{exact} - u_{appr}. \tag{A.22}$$

Therefore when combining (A.22) and the definition for the L^2-Norm, (A.21), the normed error squared is defined as,

$$||\epsilon||^2_{L^2} = \int_{\Omega} |u_{exact} - u_{appr}|^2 d\Omega. \tag{A.23}$$

A.6 Incorporating Boundary Conditions (BC)

For the solution a differential equation, e.g. (A.11), it was made clear in the above how to expand a solution as a truncated series. Within this series the constants $\hat{u}_n$ have to be found. The expansion can also be written in matrix form as:

$$\mathbf{A}\hat{\mathbf{u}} = \mathbf{f} \tag{A.24}$$

From a mathematical point of view to solve a first order differential equation one needs at least one boundary condition. For every other increase in the order of the differential equation an additional

boundary condition is needed. In case of the first order differential equation the boundary condition is defined somewhere within the matrix system of (A.24). However it is not practical to solve the system this way. It would be easier to get the row and column corresponding to the boundary value to the most right column and the boundary row down to the bottom of matrix $\mathbf{A}$ displayed in (A.25), [13]

$$\left[\begin{array}{c|c} \mathbf{A_I} & \mathbf{A_{IB}} \\ \hline \mathbf{A_{BI}} & \mathbf{A_{BB}} \end{array}\right] \left[\begin{array}{c} \hat{\mathbf{u}}_\mathbf{I} \\ \hline \hat{\mathbf{u}}_\mathbf{B} \end{array}\right] = \left[\begin{array}{c} \mathbf{f_I} \\ \hline \mathbf{f_B} \end{array}\right], \tag{A.25}$$

by making the distinction between interior $\mathbf{I}$ and boundary $\mathbf{B}$ nodes. In this way the interior matrix $\mathbf{A_I}$ can be used to solve the system later. However the boundary condition will be incorporated first to the right hand side forcing function, simply by multiplying the last entry of $\hat{\mathbf{u}}_\mathbf{B}$ with the column $\mathbf{A_I}$ and thus can be taken out of matrix $\mathbf{A}$. Then the bottom row in the matrix system is superfluous and is omitted, leaving equation (A.26) to:

$$\left[\begin{array}{c} \mathbf{A_I} \end{array}\right] \left[\begin{array}{c} \hat{\mathbf{u}}_\mathbf{I} \end{array}\right] = \left[\begin{array}{c} \mathbf{f_I} - \mathbf{A_{IB}}\hat{\mathbf{u}}_\mathbf{B} \end{array}\right] \tag{A.26}$$

This new system can be solved without much trouble by solving (A.26) and remaining with the desired vector $\hat{\mathbf{u}}_\mathbf{I}$.

A.7 Mesh Refinement

To be able to see the computed solution more clearly it is necessary to lay a refined mesh over the computed mesh. With the computed mesh, only the values at the computed GLL nodes resulting in $\hat{\mathbf{u}}$ from the previous section. When just plotting these values one obtains for example figure A.4. Notice that the solution between the GLL-nodes is missing and therefore just a straight line is

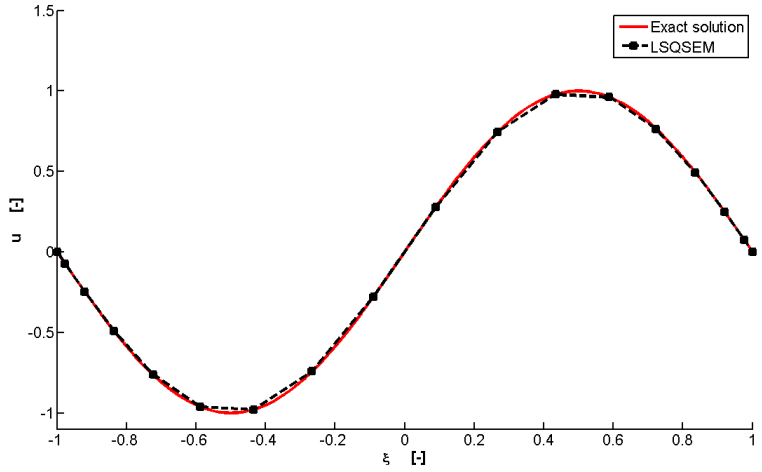


Figure A.4: Course grid GLL-nodes $u(x) = sin(x), n = 17$

drawn. One could make the mistake by meaning to increase the order excessively to make it 'look' better, but in fact wasting computer power and CPU time as it is only a matter of representation. Essentially the following is plotted:

$$u(x_p) = \sum_{n=0}^{N} \hat{u}_n \phi_n(x_p) \tag{A.27}$$

Thus for every GLL-node x_p only the corresponding constant solution $\hat{u}_n$ is plotted as, recall:

$$\phi_n(x_p) = \delta_{np}.$$

Therefore now a fine mesh is chosen, in particular a mesh with a GLL structure, so not a uniform mesh. These points are then inserted in the general solution so that again (A.7) holds:

$$u(x) \approx u_N(x) = \sum_{n=0}^{N} \hat{u}_n \phi_n(x) = \hat{u}_0\phi_0(x) + \cdots + \hat{u}_N\phi_N(x),$$

With the now known constants $\hat{u}_n$ and base functions $\phi_n(x)$ evaluated at any point x with x being a fine GLL mesh node. The contribution of every basis function on the computational domain will now be added together in one single point. Doing this with multiple points will result in the fine GLL mesh. This can be now plotted together with the points from (A.27) and a smooth function can be represented, Figure A.5 resulting from one solution of constants $\hat{u}_n$ without having to increase the order further.

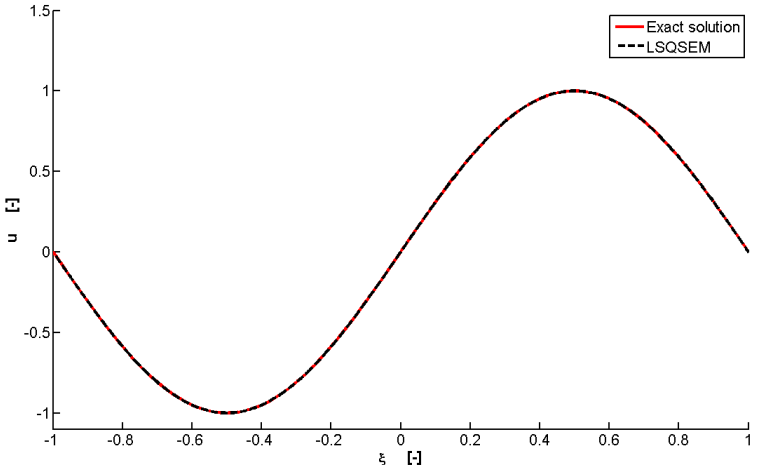


Figure A.5: Refined mesh between GLL-nodes $u(x) = sin(x)$

Appendix B

Test Cases

In this chapter two test cases will be discussed to see how the theory of appendix can be applied on a simple problem. The first test case will concern a step by step implementation of the LSQSEM in one 1D. The second part will deal with a simple 2D test problem.

B.1 Full step by step implementation LSQSEM on a sample problem

Let us now consider a simple sample problem to go through all the steps required to solve an equality. As a simple example the following differential equation is chosen:

$$\frac{\partial u}{\partial x} = f(x), \tag{B.1}$$

with $x = [-1, 1]$ and $f(x) = x$ being the forcing function. One would like to solve (B.1) and of course the exact solution is known to be $u(x) = \frac{1}{2}x^2$ which will help to see if the results of the LSQSEM or Galerkin minimization are correct. It was already previously derived how $u(x)$ can be expanded in a series solution like (2.3):

$$u(x) = \sum_{n=0}^{N} \hat{u}_n \phi_n(x).$$

It is now allowed to take the derivative as $\hat{u}_n$ is a constant, therefore:

$$\frac{\partial u(x)}{\partial x} = \sum_{n=0}^{N} \hat{u}_n \frac{\partial \phi_n(x)}{\partial x}.$$

A choice has to be made on what sort of polynomial is going to be used to evaluate the differential equation. Here a Lagrangian basis is taken. Further the right hand side forcing function has to be expanded as well,

$$f(x) = \sum_{n=0}^{N} f(x)\phi_n(x). \tag{B.2}$$

The right and left hand side of the initial differential equation are now known and thus the equality of B.1 holds now:

$$\sum_{n=0}^{N} \hat{u}_n \frac{\partial \phi_n(x)}{\partial x} = \sum_{n=0}^{N} f(x)\phi_n(x).$$

To obtain an approximated solution the said equation has to be integrated now as:

$$\int_\Omega \sum_{n=0}^{N} \hat{u}_n \frac{\partial \phi_n(x)}{\partial x} v(x) dx = \int_\Omega \sum_{n=0}^{N} f(x)\phi_n(x)v(x)dx,$$

According to the Galerkin and Least Square integration a test function $v(x)$ is needed to be multiplied on both sides for integration. For the Galerkin Method this incorporates to multiply the left and right hand side with $v(x) = \phi_n(x)$ and for the Least Square Method the test function is $v(x) = \phi'_n(x)$. In both cases the integration has to be multiplied with a weighting factor w_p as both types of integration are family of the weighted residuals, see Appendix **??**. Using the GLL quadrature yields:

$$\sum_{p=0}^{P}\sum_{n=0}^{N}\hat{u}_n\frac{\partial\phi_n(x_p)}{\partial x}v(x_p)w_p = \sum_{p=0}^{P}\sum_{n=0}^{N}f(x_p)\phi_n(x_p)v(x_p)w_p,$$

If now the Galerkin method is taken the differential equation can be written as:

$$\begin{aligned}\textstyle\sum_{n=0}^{N}\begin{bmatrix}\hat{u}_1 & \cdots & \hat{u}_N\end{bmatrix}\begin{bmatrix}\phi'_0(x_0)\\ \vdots\\ \phi'_N(x_N)\end{bmatrix}\begin{bmatrix}\phi_0(x_0) & \cdots & \phi_N(x_N)\end{bmatrix}w_p =\\ \textstyle\sum_{n=0}^{N}\begin{bmatrix}f(x_0) & \cdots & f(x_N)\end{bmatrix}\begin{bmatrix}\phi_0(x_0)\\ \vdots\\ \phi_N(x_N)\end{bmatrix}\begin{bmatrix}\phi_0(x_0) & \cdot & \phi_N(x_N)\end{bmatrix}w_p\end{aligned} \tag{B.3}$$

Therefore for the Least Square Method one can write by simply changing the test function from $\phi_n(x)$ to $\phi'_n(x)$, which returns:

$$\begin{aligned}\textstyle\sum_{n=0}^{N}\begin{bmatrix}\hat{u}_0 & \cdots & \hat{u}_N\end{bmatrix}\begin{bmatrix}\phi'_0(x_0)\\ \vdots\\ \phi'_N(x_N)\end{bmatrix}\begin{bmatrix}\phi'_0(x_0) & \cdots & \phi'_N(x_N)\end{bmatrix}w_p =\\ \textstyle\sum_{n=0}^{N}\begin{bmatrix}f(x_0) & \cdots & f(x_N)\end{bmatrix}\begin{bmatrix}\phi_0(x_0)\\ \vdots\\ \phi_N(x_N)\end{bmatrix}\begin{bmatrix}\phi'_0(x_0) & \cdots & \phi'_N(x_N)\end{bmatrix}w_p\end{aligned} \tag{B.4}$$

The general form will now be:

$$M\begin{bmatrix}\hat{u}_0\\ \vdots\\ \hat{u}_n\end{bmatrix} = \begin{bmatrix}f(x_0)\\ \vdots\\ f(x_N)\end{bmatrix}F. \tag{B.5}$$

When using Least Square it will look like:

$$M = \begin{bmatrix}\phi'_0(x_0)\phi'_0(x_0) & \phi'_0(x_0)\phi'_1(x_1) & \cdots & \phi'_0(x_0)\phi'_N(x_N)\\ \phi'_1(x_1)\phi'_0(x_0) & \phi'_1(x_1)\phi'_1(x_1) & \cdots & \phi'_1(x_1)\phi'_N(x_N)\\ \vdots & \vdots & \ddots & \vdots\\ \phi'_N(x_N)\phi'_0(x_0) & \phi'_N(x_N)\phi'_1(x_1) & \cdots & \phi'_N(x_N)\phi'_N(x_N),\end{bmatrix} \tag{B.6}$$

and also,

$$F = \begin{bmatrix}\phi_0(x_0)\phi'_0(x_0) & \phi_0(x_0)\phi'_1(x_1) & \cdots & \phi_0(x_0)\phi'_N(x_N)\\ \phi_1(x_1)\phi'_0(x_0) & \phi_1(x_1)\phi'_1(x_1) & \cdots & \phi_1(x_1)\phi'_N(x_N)\\ \vdots & \vdots & \ddots & \vdots\\ \phi_N(x_N)\phi'_0(x_0) & \phi_N(x_N)\phi'_1(x_1) & \cdots & \phi_N(x_N)\phi'_N(x_N)\end{bmatrix}, \tag{B.7}$$

For the case of using Galerkin as basis functions, the right hand side F will look like:

$$F = \begin{bmatrix}1 & 0 & \cdots & 0\\ 0 & 1 & 0 & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 1\end{bmatrix} \tag{B.8}$$

, simply because it must hold,

$$\phi_n(x_p) = \delta_{np}.$$

then solving to the right side:

$$\begin{bmatrix} \hat{u}_1 \\ \vdots \\ \hat{u}_n \end{bmatrix} = M^{-1} \begin{bmatrix} f(x_0) \\ \vdots \\ f(x_N) \end{bmatrix} F$$

The returned vector $\hat{u}_{1\cdots n}$ contains the unknown constants for the expansion of $u(x)$, (A.7) in all the GLL zeros. Therefore it is know now that:

$$u(x_i) = \hat{u}_i \phi_i(x_i) \qquad , i = 0 \cdots N$$

as follows by combing Equation A.7 and B.2, as all other basis function for one specific zero are all zero as this is the property of the basis functions. Knowing now that $\phi_n(x_p = 1$ for all $n = p$ it results in:

$$u(x_i) = \hat{u}_i \qquad , i = 0 \cdots N$$

Finally, some small remarks on some characteristics of the Galerkin and Least Square formulation should be made to be able to check the correctness of the implementation.
The Galerkin method will always return a diagonal right hand side matrix F whereas Least Square will return a non diagonal one. The row sum of the right hand side matrix M will have to be always zero for both, Galerkin and Least Square. Further matrix M will be symmetric for Least Square and anti-symmetric for Galerkin.

B.2 LSQSEM application in a two-dimensional sample problem

The basic theory used for the two-dimensional analysis is very similar to the one-dimensional case. However, since the one-dimensional elements are expanded to rectangular elements in two dimensions, the major difference is the expansion basis used. As has been pointed out in section A.2 the expansion basis used in two dimensions is a product of two one dimensional expansion bases. The resulting expansion is shown in B.9.

$$u(x, y) = \sum_{i=0}^{N} \sum_{j=0}^{N} \hat{u}_{ij} \phi_i(x) \phi_j(y). \tag{B.9}$$

In case of the first order partial derivative in x-direction the expansion can be written as:

$$\frac{\partial u}{\partial x} = \sum_{i=0}^{N} \sum_{j=0}^{N} \hat{u}_{ij} \frac{\partial \phi_i(x)}{\partial x} \phi_j(y) \tag{B.10}$$

The forcing function can be expanded similarly.

$$f(x, y) = \sum_{i=0}^{N} \sum_{j=0}^{N} \hat{f}_{ij} \phi_i(x) \phi_j(y). \tag{B.11}$$

The two-dimensional expansion of equation B.1 is thus written as:

$$\int_{-1}^{1} \int_{-1}^{1} \left[\sum_{i=0}^{N} \sum_{j=0}^{N} \hat{u}_{ij} \frac{\partial \phi_i(x)}{\partial x} \phi_j(y) - \sum_{i=0}^{N} \sum_{j=0}^{N} \hat{f}_{ij} \phi_i(x) \phi_j(y) \right] \frac{\partial \phi_k(x)}{\partial x} \phi_l(y) dx dy \tag{B.12}$$

In this expression a standard two-dimensional element and the least squares approximation (the test function is equal to the two-dimensional basis function) are used. Using the Gauss-Lobatto quadrature this equation can be rewritten as follows:

$$\sum_{p=0}^{N}\sum_{q=0}^{N}\left[\sum_{i=0}^{N}\sum_{j=0}^{N}\hat{u}_{ij}\frac{\partial\phi_i(x)}{\partial x}\phi_j(y)-\sum_{i=0}^{N}\sum_{j=0}^{N}\hat{f}_{ij}\phi_i(x)\phi_j(y)\right]\frac{\partial\phi_k(x)}{\partial x}\phi_l(y)\Bigg|_{\substack{x=x_p\\y=y_q}} w_p w_q \tag{B.13}$$

This can be rewritten as

$$\sum_{p=0}^{N}\sum_{q=0}^{N}\left[\sum_{i=0}^{N}\hat{u}_{iq}\frac{\partial\phi_i(x)}{\partial x}|_{x=x_p}-\hat{f}_{pq}\frac{\partial\phi_k(x)}{\partial x}|_{x=x_p}\delta_l(q)w_p w_q\right] \tag{B.14}$$

In the last expression use has been made of the fact that the Lagrange polynomials evaluated at the Legendre zeros equal unity at one zero and zero at all the other ones. Hence, the following must hold again

$$\phi_i(x_p)=\delta_{ip}.$$

The Kronecker delta yields an identity matrix. As compared to the one-dimensional case the weights will produce a weight matrix. Clearly, the equations indicate a certain amount of book-keeping is involved with higher dimensional expansion problems. The left hand side matrix in two dimensions is again a symmetric, diagonal matrix. On the diagonal the corresponding one-dimensional matrices are placed. The resulting matrices are shown in equation B.15. The symmetric one-dimensional matrix $\mathbf{d}$ has been discussed in (A.12)

$$M_{2D}=\begin{bmatrix}\mathbf{d} & & & \\ & \mathbf{d} & & \\ & & \ddots & \\ & & & \mathbf{d}\end{bmatrix} \tag{B.15}$$

The general form of the two-dimensional weight matrix is presented:

$$W=\begin{bmatrix} w_p(1)w_q(1) & w_p(2)w_q(1) & \cdots & w_p(N)w_q(1)\\ w_p(1)w_q(2) & w_p(2)w_q(2) & \cdots & w_p(N)w_q(2)\\ \vdots & \vdots & \ddots & \vdots\\ w_p(1)w_q(N) & w_p(2)w_q(N) & \cdots & w_p(N)w_q(N)\end{bmatrix}, \tag{B.16}$$

Certainly, the boundary conditions can be applied as in the one-dimensional case. The procedure is not as trivial as in the one-dimensional case and a detailed description is omitted in this literature review. However, generally the procedure can be summarized as follows. First, a reordering of the left- and right-hand-side matrices is performed. It is recommended to rearrange the matrices as has been outlined in (A.25) section A.6 and the matrix system can be solved to obtain the desired interior values of $\hat{\mathbf{u}}$.

Appendix C

Results Error Analysis

C.1 Examples on error analysis in 1D

This section will deal with the behavior of the error for different type of forcing function f defined by the differential equation in equation B.1.

$$\frac{\partial u(x)}{\partial x} = f(x)$$

For this error analysis known functions are taken for which the primitives and derivatives are known. Therefore the following functions for $u(x)$ are taken for which the forcing function can be determined by simply taking the derivative, so numerically integrating the forcing function should result in an approximate solution u_{appr} of the exact solution u_{exact}, like:

$$\begin{aligned} u_1(x) = sin(\pi x) \quad &\rightarrow \quad f_1(x) = \pi cos(\pi x) \\ u_2(x) = x^8 + 5x^3 \quad &\rightarrow \quad f_2(x) = 8x^7 + 15x^2 \end{aligned} \tag{C.1}$$

In the scope of this literature review first Galerkin will be compared with Least Square method and secondly the h-refinement and p-refinement. A closer look will be taken between the differences of the computed error of the differential equation, for both Galerkin an Least Square method. Then the difference in the refinements are looked at. First the cell size is kept constant and and the polynomial order altered. Subsequently the order will be kept constant and then the number of sub-domains changed, i.e. the amount of elements. This will enable an assessment how the error behaves by changing these mentioned parameters.

C.1.1 Galerkin vs. Least Square p-refinement

Compared to Least Squares, the Galerkin Method is still more popular to be use. As discussed in section A.3.1 the behavior of Galerkin will show 'wiggle' like behavior in the convergence. In this section Galerkin is therefore compared with Least Square to see what this implies for this thesis and to verify the theory [17].

First the function $f_1(x)$ is taken and computed on a single element and $u_1(x)$ will be used as a reference to the computed values as this is the exact solution. Figure C.1 shows the comparison between Galerkin and LS integration methods for an order of $n = 4$. Clearly Least Square is more accurate. However when looking at Figure C.2 it seems that Galerkin and Least Square are roughly equally far converged. Consulting therefore the error evolution, according to the L^2-norm A.5, with increasing the order of both, Galerkin and Least Square in one Figure, shows indeed the 'wiggle' like behavior in the convergence of Galerkin, Figure C.3

Secondly a closer look is taken at a function of higher polynomial order. Both Galerkin and Least Square should perform well as they both incorporate basis functions that are consisting of

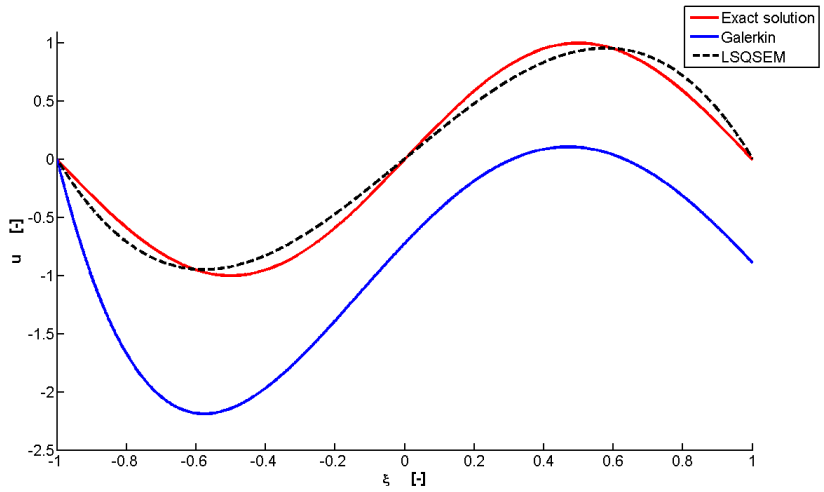


Figure C.1: Comparison $u(x) = sin(x)$ of Galerkin and LS (order n=4)

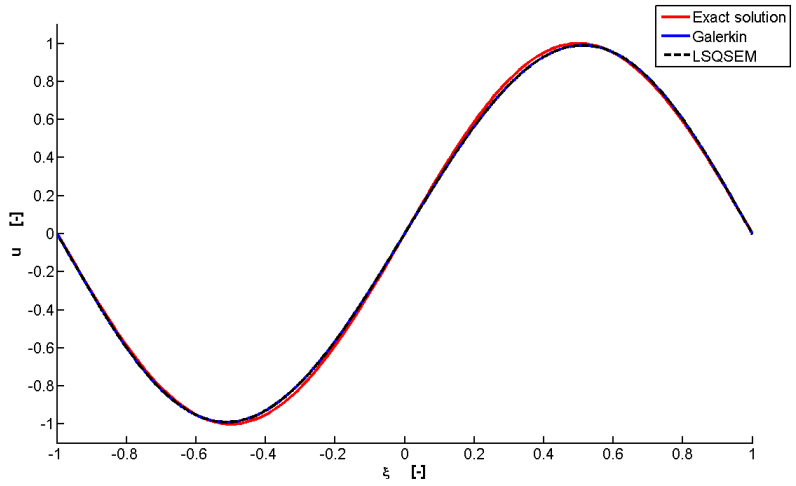


Figure C.2: Comparison $u(x) = sin(x)$ of Galerkin and LS (order n=5)

polynomials themselves. Therefore $u_2(x)$ and $f_2(x)$ are used now and drawn in Figure C.4 for a polynomial order of $n = 5$. Once more Galerkin seems to be lacking behind in convergence to the exact solution. Therefore again the error is analyzed. Figure C.5 displays again the 'wiggles' and an other interesting event, in particular that the solution for the convergence actually suddenly drops down to the exact solution, for both Galerkin and Least Square. The error that still remains after $n = 8$ is just the machine error.

C.1.2 Galerkin vs Least Square h-refinement

First $u_1(x)$ is considered again and the corresponding error shown in figure C.6 and than $u_2(x)$ in figure C.7. The results are similar as in the previous section. Least Squares performs better.

C.1.3 p-refinement vs. h-refinement Least Square

It is clear that a p-refinement, Appendix A.4 converges faster to the exact solution than h-refinement. Nevertheless h-refinement can not be completely neglected. Think about a boundary which is not straight, but one with the presence of a sharp corner or kink. It is then evidently

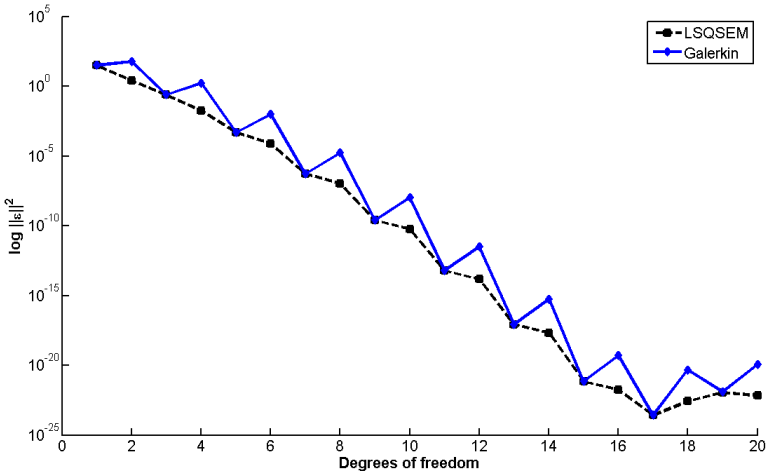


Figure C.3: Comparison error according to the L^2-Norm for $u(x) = sin(x)$

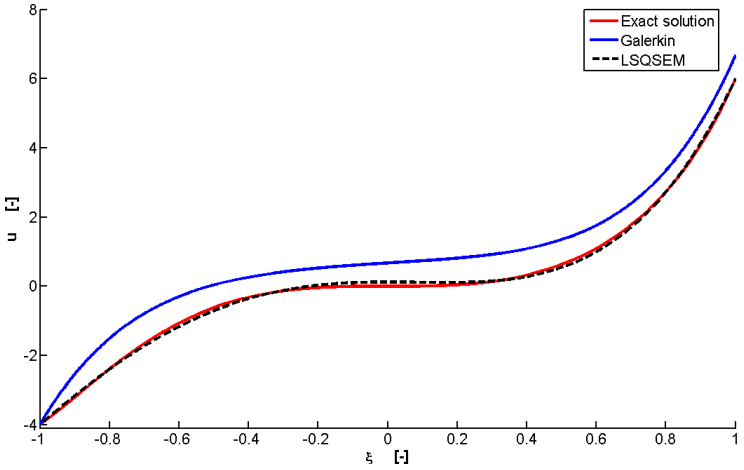


Figure C.4: Comparison $u(x) = x^8 + 5x^3$ Galerkin with Least Square $n = 5$

needed to be able to follow the contour of the boundary with the use of at least more than one cell. Therefore in the remainder of this section p- versus h-refinement is analyzed and once more use is made of the functions $u_1(x)$ and $u_2(x)$

C.2 Error Analysis Results 2D

In this section the results obtained with the two-dimensional calculations are presented. As in the one-dimensional case discussed before a p-refinement is compared with a h-refinement. However, in the two-dimensional case only the LSQSEM is considered.
As in the one-dimensional case a convergence study of the residual error is performed based in two functions:

$$\begin{aligned} u_1(x,y) &= sin(\pi x)cos(\pi y) \quad \rightarrow \quad f_1(x,y) = \pi cos(\pi x)cos(\pi y) \\ u_2(x,y) &= x^9y^2 \qquad\qquad\quad \rightarrow \quad f_2(x,y) = 9x^8y^2 \end{aligned} \tag{C.2}$$

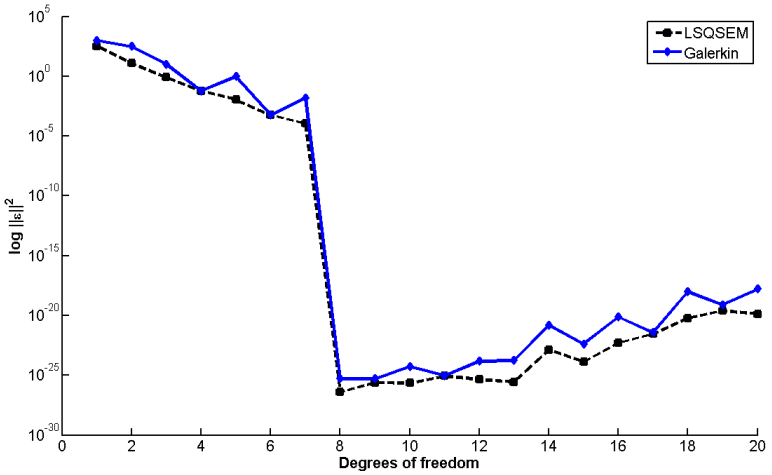


Figure C.5: Comparison error according to the L^2-norm for $u(x) = x^8 + 5x^3$

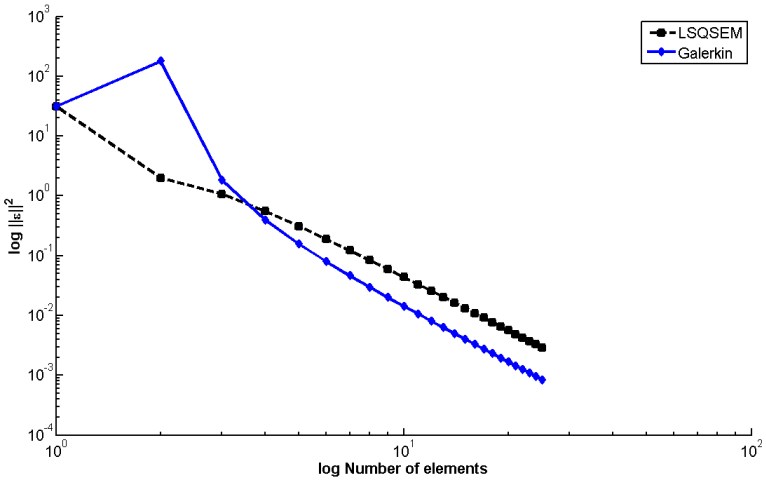


Figure C.6: Comparison error $u(x) = sin(x)$ Galerkin with Least Square h-refinement

C.2.1 2D p-refinement

For the p-refinement the order of the polynomials is increased from $n = 1$ (linear polynomials) to $n = 20$. The number of elements is kept constant at one element - the standard element. Figure C.9 shows the convergence behavior of the function $u_1(x, y)$. It can be seen that with increasing the polynomial order while keeping the number of elements constant the residual error decreases exponentially. At a polynomial order of $n = 16$ the exact solution is reached. Higher orders do not yield more accurate results. In Figure C.10 and C.11 a surface plot is shown of the function $u_1(x, y)$ using second order ($n = 2$) polynomials. In the first figure only the GLL points are used. In the second figure the mesh is refined using just these GLL points. The refinement is needed for a more accurate error analysis. The approximation converges towards the real solution with increasing the polynomial order. Two surface plots showing the converged solution are presented in Figures C.12 and C.13, respectively. The first of these two shows the GLL points are used. The second figure shows the same polynomial order with a refined mesh.

In Figure C.14 the convergence behavior of $u_2(x, y)$ is shown. The sudden drop at $n = 10$ indicates that the actual solution is reached. Again, increasing the polynomial order will not yield a more accurate result.

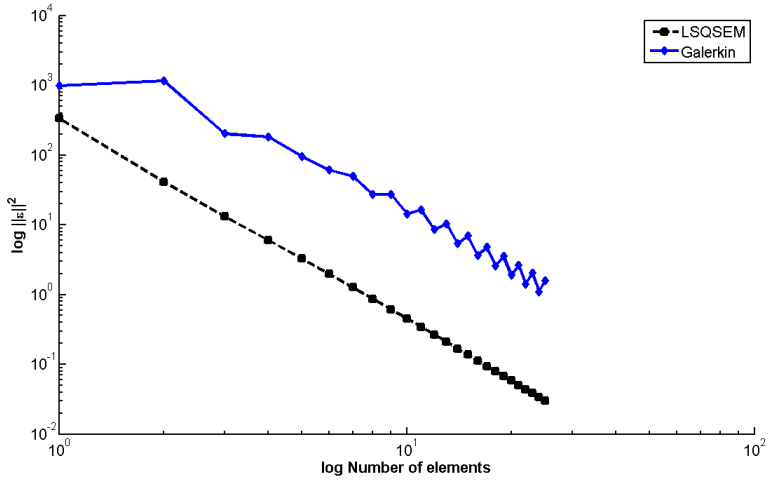


Figure C.7: Comparison error in the L^2-Norm for $u(x) = x^8 + 5x^3$

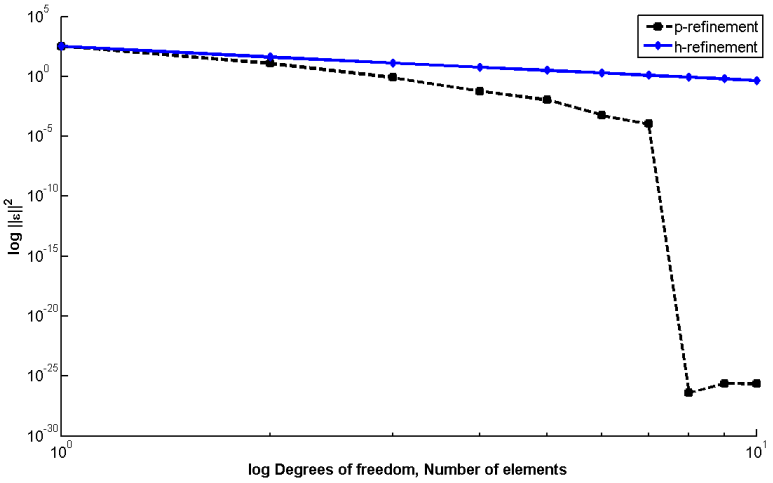


Figure C.8: Comparison error according to the L^2-norm for $u(x) = x^8 + 5x^3$ h- vs p-refinement

C.2.2 2D h-refinement

For the h-refinement the number of square elements is increased from $m = 1$ to $m = 20$ while the polynomial order is kept constant. In the present case linear polynomials are used.

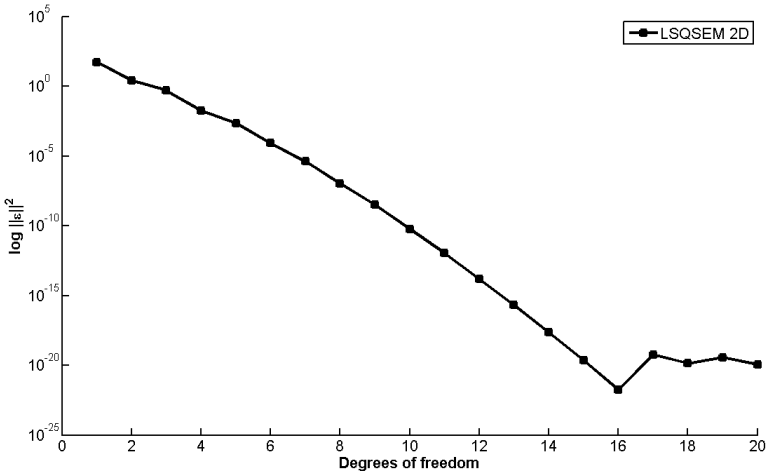


Figure C.9: Error according to the L^2-Norm for $u(x,y) = sin(\pi x)cos(\pi y)$ p-refinement

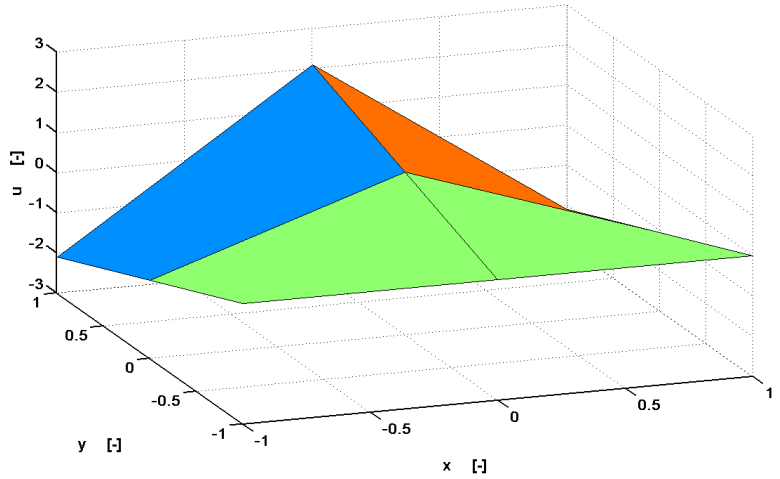


Figure C.10: Surface plot for $u(x,y) = sin(\pi x)cos(\pi y)$, $n = 2$, GLL points

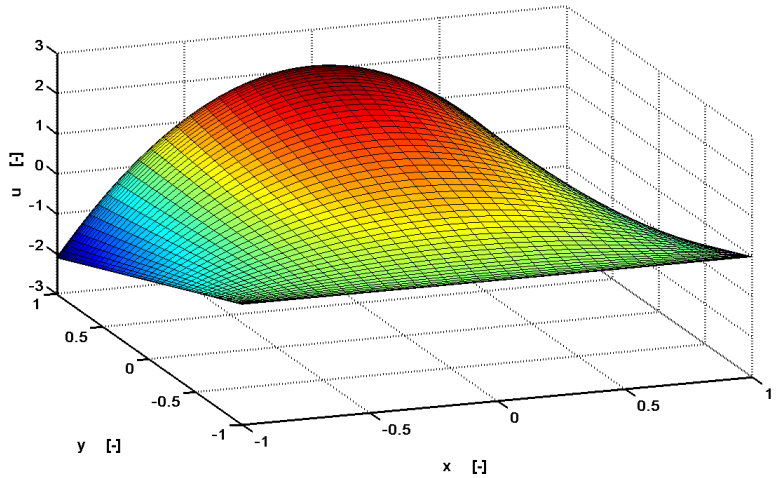


Figure C.11: Surface plot for $u(x,y) = sin(\pi x)cos(\pi y)$, $n = 2$

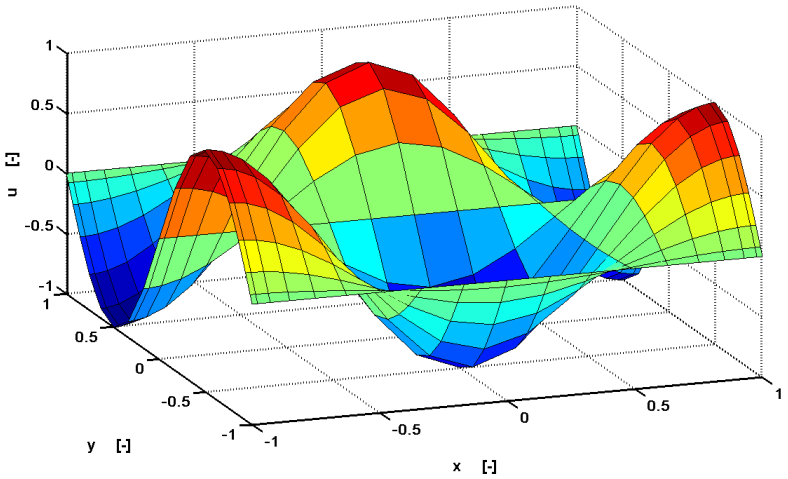


Figure C.12: Surface plot for $u(x, y) = sin(\pi x)cos(\pi y)$, $n = 18$, GLL points

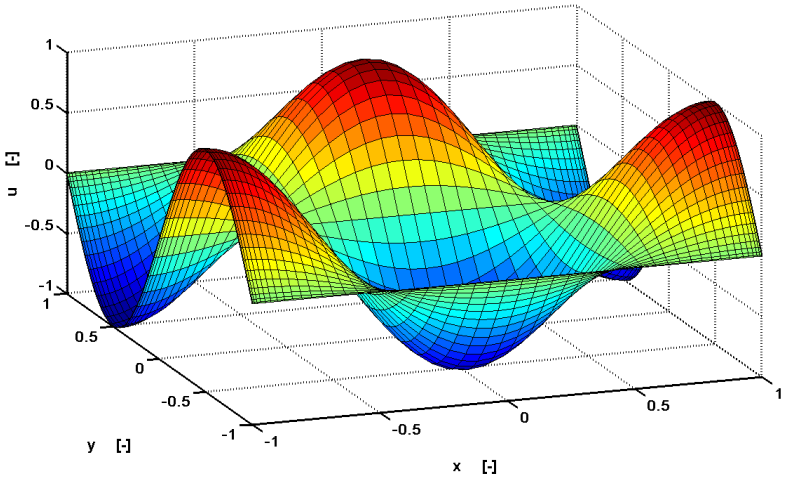


Figure C.13: Surface plot for $u(x, y) = sin(\pi x)cos(\pi y)$, $n = 18$

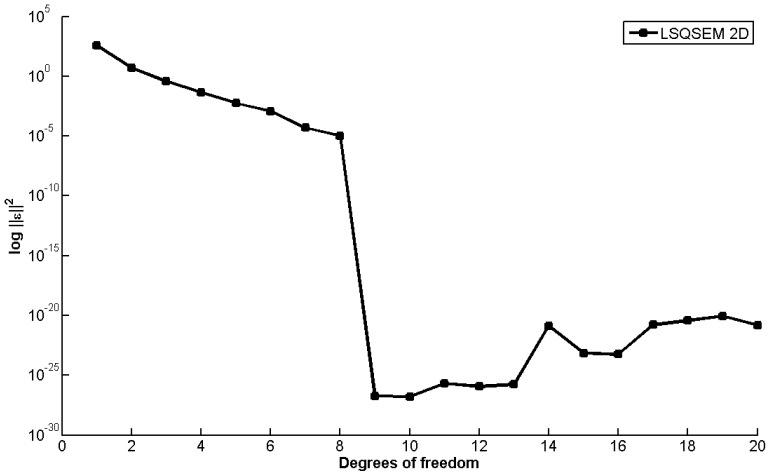


Figure C.14: Error according to the L^2-Norm for $u(x, y) = x^9y^2$ p-refinement