FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT • DEPARTMENT INFORMATIK

Lehrstuhl für Informatik 10 (Systemsimulation)

Efficient generation of Mehrstellenverfahren for elliptic PDEs

Marco Heisig

Bachelorarbeit

Efficient generation of Mehrstellenverfahren for elliptic PDEs

Marco Heisig

Bachelorarbeit

Aufgabensteller:	Prof. Dr. U. Rüde
Betreuer:	D. Ritter
Bearbeitungszeitraum:	18.04 2013 - 18.09 2013

Erklärung:

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Systemsimulation (Informatik 10), wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Bachelorarbeit einschließlich etwaiger Schutzrechte und Urheberrechte eingeräumt.

Erlangen, den 19. September 2013 .....

Abstract

I present several algorithms to generate finite-difference stencils for elliptic partial differential equations. With the given algorithms it is possible to generate stencils of any order of accuracy — including Mehrstellenverfahren — on arbitrary grids and in any number of dimensions. I discuss the efficient implementation of those algorithms in the computer algebra system Maxima and how to integrate a computer algebra system in existing numerical codes. Two model problems are used to validate the generated stencils.

Contents

1 Introduction

Elliptic partial differential equations occur frequently in scientific applications. Notable examples are static heat distributions and electrostatic potentials. Finite-difference methods are simple, yet powerful numerical solution techniques for those cases. Throughout this thesis I will present algorithmic approaches for generating finite-difference methods as well as their implementation.

1.1 Taylor's theorem

Taylor's theorem allows to approximate an arbitrary n-times differentiable function by power series. Such approximations are used to set up finite-difference equations as well as to derive error estimates. The formulation and proof are taken from [For89, p. 174].

Theorem 1.1 (Taylor's theorem)**.** *Let $\mathbf{I} \subset \mathbb{R}$ be an interval and $f : \mathbf{I} \to \mathbb{R}$ a $(n+1)$-times differentiable function. Then for $a \in \mathbf{I}$ and $x \in \mathbf{I}$*

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_{n+1}(x), \tag{1.1}$$

where

$$R_{n+1}(x) = \frac{1}{n!}\int_a^x (x-t)^n f^{(n+1)}(t)\,dt \tag{1.2}$$

Proof. By induction in n. For $n = 0$ we use the fundamental theorem of calculus

$$f(x) = f(a) + \int_a^x f'(t)\mathrm{dt} \tag{1.3}$$

Then with $n \leftarrow n+1$

$$\begin{aligned} R_n(x) &= \frac{1}{(n-1)!}\int_a^x (x-t)^{n-1} f^{(n)}(t)\mathrm{dt} \\ &= -\int_a^x f^n(t)\,\frac{\mathrm{d}}{\mathrm{dt}}\left(\frac{(x-t)^n}{n!}\right)\mathrm{dt} \\ &= \frac{f^{(n)}(a)}{n!}(x-a)^n + \frac{1}{n!}\int_a^x (x-1)^n f^{n+1}(t)\mathrm{dt} \end{aligned}$$

□

It is possible with this theorem to relate values at one point x with derivatives at another point $x - a$. An important property of R_n is that if $f^{(n)}$ is continnuous, the error introduced by omitting R_n is in $O(|x-a|^n)$. So as an example a function $u(x)$ can be approximated by the values at a neighboring point $u(x+h)$ by

$$u(x) = u(x+h) + -h\,u'(x+h) + O(h^2). \tag{1.4}$$

A more general formulation of the theorem is required to deal with mixed derivatives in more than one dimension. To abbreviate the formula I introduce the multiindex notation as such:

Notation. *Let α be an n-tuple $(\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n$ then*

$$\begin{aligned} |\alpha| &:= \alpha_1 + \cdots + \alpha_n \\ \alpha! &:= \alpha_1! \cdot \ldots \cdot \alpha_n! \\ x^\alpha &:= x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n} \\ D^\alpha f &:= \frac{\partial^{|\alpha|} f}{\partial x_1^{a1} \ldots \partial x_n^{a_n}} \end{aligned}$$

Theorem 1.2 (Taylor's theorem in n-dimensions)**.** *Let $U \subset \mathbb{R}^n$ be open, $x \in U$ a point and $\xi \in \mathbb{R}^n$ a vector with $x + t\xi \in U$ for all $t \in [0, 1]$. Furthermore let $f : U \to \mathbb{R}$ be a $(k+1)$-times continuous differentiable function. Then there exists a $\theta \in [0, 1]$ that*

$$f(x + \xi) = \sum_{|\alpha| \leq k} \frac{D^\alpha f(x)}{\alpha!} \xi^\alpha + \sum_{|\alpha| = k+1} \frac{D^\alpha f(x + \theta\xi)}{\alpha!} \xi^\alpha \tag{1.5}$$

Proof. See [For91, p.56].

The considerations about the errors of the truncated series work as in the one dimensional case. The error introduced by truncation at k is in $O(\sum_{|\alpha|=k+1} \xi^\alpha)$ if the relevant derivatives are continuous. More specifically the error for approximations on cartesian grids, with $\xi = (c_1 h, c_2 h, \ldots, c_N h)$, $c_i \in \mathbb{N}$, $i = 1, \ldots, N$ is in $O(h^{k+1})$.

As a simple example consider the approximaion of a function $u(x, y)$ in two variables by the function values of $u(x + h, y + h)$:

$$u(x, y) = u(x + h, y + h) - h\frac{\partial u}{\partial x}(x + h, y + h) - h\frac{\partial u}{\partial y}(x + h, y + h) + O(h^2) \tag{1.6}$$

Of course this relation only holds if the requirements for Taylor's theorem are met. The approximation and error estimate are invalid if the function u is not differentiable enough times. An important special case arises if u is a polynomial of degree N. In this case the Taylor series up to degree N are an exact solution and there is no benefit in adding more terms to the series.

1.2 Finite-difference methods

Finite-difference methods are widely applicable numerical solution techniques for partial differential equations. I will treat only boundary-value problems in second order elliptic partial differential equations and therefore restrict my elaboration to those.

Solving a second order boundary-value problem means finding a function $u(x)$, $x \in \mathbb{R}^n$ that satisfies the differential equation

$$L[u] = \sum_{|\alpha|=2} c_\alpha u^{(\alpha)} + \sum_{|\alpha|=1} d_\alpha u^\alpha + eu = r \tag{1.7}$$

on a domain Ω, where α is the multiindex notation introduced in 1.1 as well as boundary conditions that have to be satisfied on the domain boundary $\partial\Omega$. Equation 1.7 is called *elliptic* if the coefficients c_α of the second order derivatives satisfy the condition

$$\det \begin{pmatrix} c_{2,0,\ldots,0} & c_{1,1,\ldots,0} & \cdots & c_{1,0,\ldots,1} \\ c_{1,1,\ldots,0} & c_{0,2,\ldots,0} & \cdots & c_{0,1,\ldots,1} \\ \vdots & & \ddots & \vdots \\ c_{1,0,\ldots,1} & \cdots & & c_{0,0,\ldots,2} \end{pmatrix} > 0 \tag{1.8}$$

A finite-difference method is now derived by the following steps:

Discretisation of the domain A mesh of points is spanned across the domain Ω. This mesh is often just a cartesian grid where the distance to a points nearest neighbors is always h. But triangular or hexagonal meshes are also common and completely arbitrary meshes are possible. There is also the option to use an adaptive mesh that is finer on those regions where numerical instabilities are expected or where a especially accurate solution is needed.

Discretisation of the differential operator After a suitable mesh is chosen, it is possible to discretize the given differential equation. Therefore a set of neighbor points is chosen for each mesh point and an expression is derived that approximates the operator $L[u]$ as good as possible. The advantage of uniform grids is here, that a single expression can be reused for all other points.

Discretisation of the boundary conditions A similar expression as for the points in Ω has to be derived for all points on $\partial\Omega$. Only in this case the expression has to approximate not only the differential operator, but also the local boundary conditions.

Solving of the linear system An approximate solution to u can now be calculated by solving the linear system of all previously generated expressions. Although Gaussian elimination can be used to solve the equations, it is often infeasible as the system of equations tends to be very large. Instead iterative solvers as the Gauss-Seidel method or the Jacobi method are used. The best results in terms of computational complexity are nowadays obtained by multigrid methods.

It is important to remember that each of the steps introduces a new source of errors. Careful analysis is necessary to guarantee that a finite difference method produces a satisfactory solution.

The name finite-difference method stems from the way the approximations of differential operators are derived. The derivative of a function $\frac{\partial u}{\partial x}$ can be defined by

$$\frac{\partial u}{\partial x}(x) = \lim_{h \to 0} \frac{u(x+h) - u(x)}{h} \tag{1.9}$$

An approximation of the derivative is therefore given by

$$\frac{\partial u}{\partial x}(x) \approx \frac{u(x+h) - u(x)}{h} \tag{1.10}$$

Which is often referred to as *forward difference operator*. Other commonly used operators are the *backward difference operator*

$$\frac{\partial u}{\partial x}(x) \approx \frac{u(x) - u(x-h)}{h} \tag{1.11}$$

and the *central difference operator*

$$\frac{\partial u}{\partial x}(x) \approx \frac{u(x+h) - u(x-h)}{2h}. \tag{1.12}$$

Those formulas can already be used as finite-difference approximations for the first derivative of a function, although not very accurate. Approximations of higher derivatives can be derived by repeatedly applying these operators to a function. This method, as well as a way to extend it to higher order approximations with more points is presented in [KP78].

1.3 Stencil notation

The focus of this thesis is on the fast and accurate approximation of a differential operator $L[u]$ on an expansion point ξ with a set of mesh points $\sigma = \{p_1, p_2, \ldots, p_n\}$. Usually $\xi \in \sigma$. The approximation has the form

$$\sum_{k=1}^{n} w_k u(p_k) = L[u](\xi) + \epsilon \tag{1.13}$$

where ϵ is the error introduced by the approximation. The w_k are the *weights* and the set of points p_k with their corresponding weigths w_k are the *stencil*. On regular grids it is sufficient to denote the stencil by the expansion point ξ and the distance to each point in interval steps. If the same stencil can be applied to all grid points it is even possible to omit the expansion point ξ.

A convenient notation for stencils on regular grids is as a tensor with the weights as elements. The element in the center represents the weight of the expansion point. The other elements represent the weights of the neighbor points. The stencil

$$\begin{bmatrix} 1 & -2 & 1 \end{bmatrix} \tag{1.14}$$

for a function $u(x)$ on a cartesian grid with spacing h is an abbreviation for

$$u(x-h) - 2u(x) + u(x+h). \tag{1.15}$$

With the stencil notation an approximation to the laplace equation $\Delta u = 0$ on a two dimensional cartesian grid can be denoted by

$$\frac{1}{h^2} \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} u = 0 \tag{1.16}$$

as a short form for

$$\frac{u(x-h,y) + u(x+h,y) + u(x,y-h) + u(x,y+h) - 4u(x,y)}{h^2} = 0. \tag{1.17}$$

1.4 The computer algebra system Maxima

Maxima is a computer algebra system with a long history. It started as a MIT project with the name Macsyma (Procect MAC's SYmbolic MAnipulator) already in 1968 with a DEC PDP-6 as a target platform. The MACSYMA program was later connected to the ARPANET, one of the predecessors of the internet, where it became more widely known. The initial assembler code was ported to a wide number of machines and implementations. After 1982 the MIT turned it into a proprietary program. The proprietary versions never took off and slowly vanished, while other systems like Maple and Mathematica emerged and dominated the market. It is due to Dr. William Shelter that the Macsyma code is not lost today. He maintained his own copy under the name Maxima until he got permission in 1998 to distribute the code as free software. Today it is probably the most powerful free computer algebra system. The complete history can be found in [dSFMY04].

I chose this tool in the belief that research should be free and reproducible. Besides there are some features that make Maxima stand out against the competition. While many computer algebra systems are implemented in imperative, compiled languages like C, Maxima is built as an embedded language on Common Lisp [Ste90]. This allows users to access the full Common Lisp language while having access to great computer algebra features. Therefore Maxima exhibits very powerful programming features and a lot of extensibility for the experienced user.

2 Stencil generation via Taylor series

This section describes the generation of stencils by linear combinations of Taylor series. This approach has the advantage that it allows treatment of very general differential equations, as well as arbitrary grids. The downside is that it requires the setup and solution of a linear system of equations. Section 4.2 documents how a implementation can mitigate the use of those costly operations. A faster, but more specific algorithm is treated in section 3.

2.1 Derivation of simple stencils

The poisson equation is the model problem for the following elaborations. It reads

$$\Delta u = f, \tag{2.1}$$

where u is the desired function and f is a given function, the so called *right hand side* of the equation for obvious reasons. The term Δu is in the same manner called *left hand side*. Δ is the Laplace operator and equivalent to $\nabla \cdot \nabla$, or div(grad u).

First I will derive a stencil for the one dimensional case on a domain discretized with a cartesian grid with spacing h. The nearest neighbors of a expansion point $u(x)$ can now be expressed as truncated Taylor series with $u(x)$ as expansion point:

$$\begin{aligned} u(x-h) &= u(x) - hu'(x) + \frac{h^2}{2}u''(x) - \frac{h^3}{6}u'''(x) + O(h^4) \\ u(x+h) &= u(x) + hu'(x) + \frac{h^2}{2}u''(x) + \frac{h^3}{6}u'''(x) + O(h^4) \end{aligned}$$

Both equations are fourth order accurate and relate values of $u(x)$, $u(x+h)$, $u(x-h)$ and $u''(x)$. Combining and reordering the two equations produces the desired stencil.

$$u''(x) = \frac{1}{h^2}\big(u(x-h) + u(x+h) - 2u(x)\big) + O(h^2) \tag{2.2}$$

This approximation is only second order accurate as I had to divide by h^2. The u' and u''' cancel out in the process due to symmetry. Otherwise the accuracy would be worse.

The multidimensional Taylor series are needed to approximate the Laplace operator in more dimensions. The classical *five point stencil* for the Laplace operator in two dimension can be generated via

$$\begin{aligned} u_{-h,0} &= u_{0,0} - hu_x + \frac{h^2}{2}u_{xx} - \frac{h^3}{6}u_{xxx} + O(h^4) \\ u_{h,0} &= u_{0,0} + hu_x + \frac{h^2}{2}u_{xx} + \frac{h^3}{6}u_{xxx} + O(h^4) \\ u_{-h,0} &= u_{0,0} - hu_y + \frac{h^2}{2}u_{yy} - \frac{h^3}{6}u_{yyy} + O(h^4) \\ u_{h,0} &= u_{0,0} + hu_y + \frac{h^2}{2}u_{yy} + \frac{h^3}{6}u_{yyy} + O(h^4) \end{aligned}$$

where $u_{a,b}$ is an abbreviation for u(x + a, y + b) and u_x represents the derivative of u in x at the point (x, y). Combining the equations gives

$$u_{xx} + u_{yy} = \frac{1}{h^2}(u_{-h,0} + u_{h,0} + u_{0,-h} + u_{0,h} - 4u_{0,0}) + O(h^2). \tag{2.3}$$

At this point I only discretized the left hand side of the Poisson equation. For the full finite-difference method, the right hand side must also be discretized. This is luckily straightforward by taking for each point $u(x)$ on the grid the function value $f(x)$. The discrete Poisson equation is then in two dimensions and stencil notation

$$\frac{1}{h^2}\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} u = f. \tag{2.4}$$

2.2 Stencils of higher order

Higher order stencils allow for accurate solutions on relatively coarse grids. The order is increased by eliminating further terms of the Taylor series. Of course this requires more equations and therefore additional information. This information can be introduced by including more neighboring gridpoints.

In the case of the one dimensional Laplace operator is it possible to use the points with distance $-2h$, h, 0, h and $2h$. The equations are then

$$\begin{aligned}
u(x-2h) &= u(x) - 2hu'(x) + 2h^2u''(x) - \frac{4h^3}{3}u'''(x) + \frac{2h^4}{3}u^{(4)} - \frac{4h^5}{15}u^{(5)} + O(h^6) \\
u(x-h) &= u(x) - hu'(x) + \frac{h^2}{2}u''(x) - \frac{h^3}{6}u'''(x) + \frac{h^4}{24}u^{(4)} - \frac{h^5}{120}u^{(5)} + O(h^6) \\
u(x+h) &= u(x) + hu'(x) + \frac{h^2}{2}u''(x) + \frac{h^3}{6}u'''(x) + \frac{h^4}{24}u^{(4)} + \frac{h^5}{120}u^{(5)} + O(h^6) \\
u(x-2h) &= u(x) + 2hu'(x) + 2h^2u''(x) + \frac{4h^3}{3}u'''(x) + \frac{2h^4}{3}u^{(4)} + \frac{4h^5}{15}u^{(5)} + O(h^6)
\end{aligned}$$

and can be combined for the stencil

$$u''(x) = \frac{1}{12h^2}\big(-u_{-2h} + 16u_{-h} - 30u_0 + 16u_h - u_{2h}\big) + O(h^4) \tag{2.5}$$

or in stencil notation

$$u'' = \frac{1}{12h^2}\big[-1, 16, -30, 16, -1\big]u + O(h^4). \tag{2.6}$$

Computation is already for these relatively simple stencils very tedious. Stencils of even higher order and two or three dimensions can be calculated by the same approach, but become infeasible for manual solving. This is the motivation for using computer algebra systems for this task. The additional benefit of having a computer program to generate stencils is that it can be used as a subroutine in software packages to generate finite-difference methods at runtime. This is sometimes required if the differential equations vary over time. See section 4 for some remarks on this topic.

2.3 Mehrstellenverfahren

In the previous examples I introduced further grid points to increase the order of approximation. A different approach to achieve high order was introduced by Lothar Collatz in [Col66]. Instead of approximating only the left hand side of the differential equation, he proposes to take several points of the right hand side as well. Such methods are called *Mehrstellenverfahren*. Other names used in the literature are Hermitian finite-difference methods or implicit methods.

Again I discretize the Poisson equation in one dimension, but this time with two stencils, one for the left hand side Δu and one for the right hand side f. The taylor series are then

$$\begin{aligned} u(x-h) &= u(x) - hu'(x) + \frac{h^2}{2}u''(x) - \frac{h^3}{6}u'''(x) + \frac{h^4}{24}u^{(4)} - \frac{h^5}{120}u^{(5)} + O(h^6) \\ u(x+h) &= u(x) + hu'(x) + \frac{h^2}{2}u''(x) + \frac{h^3}{6}u'''(x) + \frac{h^4}{24}u^{(4)} + \frac{h^5}{120}u^{(5)} + O(h^6) \end{aligned}$$

for the left hand side and

$$\begin{aligned} f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + O(h^4) \\ f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + O(h^4) \end{aligned}$$

for the right hand side. f has to be substituted by Δu from the Poisson equation to combine it with the left hand side approximations and is then of the form

$$\begin{aligned} u''(x-h) &= u''(x) - hu'''(x) + \frac{h^2}{2}u^{(4)}(x) - \frac{h^3}{6}u^{(5)}(x) + O(h^4) \\ u''(x+h) &= u''(x) + hu'''(x) + \frac{h^2}{2}u^{(4)}(x) + \frac{h^3}{6}u^{(5)}(x) + O(h^4). \end{aligned}$$

Linear combination and reformulation give the fourth order accurate Mehrstellenverfahren

$$-\frac{12}{h^2}\big(u(x-h) + u(x+h) - 2u(x)\big) = u''(x-h) + u''(x+h) + 10u''(x) + O(h^4) \qquad (2.7)$$

This Mehrstellenverfahren has the same order of accuracy as the standard finite difference approach in equation 2.5 while accessing only the nearest two neighbors.

A extensive treatment of higher order differential equations with both the standard finite difference mehthod and Mehrstellenverfahren can be found in [Sch00].

2.4 Formulation of a general stencil solver algorithm

Similar steps as in the previous chapter can be used to generate a broad range of stencils. However some problems arise in the general case. I describe now the steps of the algorithm, the problems that arise and how I solved them.

Series expansion To not limit myself to cases of one operator (e.g. the Laplace operator Δu) or Mehrstellenverfahren with two operators (e.g. for the Poisson equation Δu and u) I decided to use a list of any number of operators and stencils as input.

Each operator is expanded in truncated Maclaurin series (Taylor series with expansion point zero) for each stencil point. Each series are weighted with the weight of the point. The weight should be a variable.

Setting up the equations I assume that the distances of all points are roughly on the same order. In this case is the problem of generating the stencil with maximal possible order equivalent to eliminating derivatives up to that order. The order of the derivative is the sum of derivations in each variable. The equations that must be solved are therefore obtained by taking the coefficients of each derivative and setting them to zero.

Successive solving The achievable order of the approximation is initially not known. Therefore is it necessary to solve increasing orders in succession as long as possible. This means that first the derivatives of order zero are eliminated if possible, then the derivatives of order one, and so on until there are not enough weights left to eliminate the next higher order derivatives.

An important observation is that the linear system of equations to set all derivatives to zero is homogeneous. Successive elimination of increasing order will eventually produce the trivial solution that all weights are zero. To avoid this problem is it necessary to introduce a *normalisation condition*. I chose to demand that the sum of all weights on the right hand side of the discretized differential equation is one. This additional equation makes the system of equations inhomogeneous and there is no trivial solution anymore.

Minimisation The successive solver sets derivatives up to the maximal possible order to zero. Often remain some free parameters after the process. They do not suffice to increase the order of accuracy further. A good choice of these parameters can nontheless improve the resulting stencil by a constant factor. I used an unconstrained minimisation tool to minimize the squared sum of the coefficients of all derivatives of the next higher order. The benefits of such optimized stencils can be seen in 5.

The final algorithm can be denoted in the following way:

Algorithm 1: stencil generation algorithm

input : triples of the form $(operator, order, stencil)$, normalisation condition nc
output: the value of each weight in each stencil

$expr \leftarrow 0$
$weights \leftarrow$ **foreach** *triple* **do**
  $op \leftarrow$ first($triple$)
  $ord \leftarrow$ second($triple$)
  $stencil \leftarrow$ last($triple$)
  foreach *entry* **in** *stencil* **do**
    $weight \leftarrow$ first(entry)
    $expr \leftarrow expr + weight \cdot$ maclaurin(op, rest($entry$), ord)
    add *weight* **to** *weights*
  end
end
`/* sucessive elimination and final minimisation */`
$ord \leftarrow 0$
$eqns \leftarrow nc$
while *eqns* **solvable by** *weights* **do**
  add equations eliminating derivatives of order ord in $expr$ to $eqns$
  $ord \leftarrow ord + 1$
end
remove last added equations
solve $eqns$ with $weights$
minimize derivatives of order ord with remaining undetermined $weights$

The second half of the algorithm deals with the elimination and minimisation of derivatives. This process is very dependent on the functionality of the computer algebra system in use and therefore deliberately left vague.

This approach allows to generate everything from Mehrstellenverfahren to interpolation formulae on arbitrary grids and for any number of dimensions. An implementation of the algorithm is given in appendix A.3, together with examples of usage.

3 A fast one dimensional approach

A very efficient way to derive the weights of a finite-difference stencil is given in [For98a].

3.1 The algorithm

The following algorithm was published in [For98a] and treated in detail in [For98b]. It generates weights $w_{n,\nu}^m$ such that the approximations

$$\left.\frac{\partial^m f}{\partial x^m}\right|_{x=\xi} \approx \sum_{\nu=0}^{n} w_{n,\nu}^m f(x_\nu), \quad m = 0, 1, \ldots, M,\ n = m, m+1, N \tag{3.1}$$

are all optimal in the sense that they are exact to an as high degree as possible with the given points. The input is the desired maximal derivative M, a number of points N labeled $x_0, \ldots, x_n$ and an expansion point ξ. Uninitialized values are assumed to be zero for brevity of notation. For an implementation that handles these cases explicitly see Appendix A.4.

Algorithm 2: Fornberg's Algorithm

input : $M, N, \xi, x_0, x_1, \ldots, x_N$
output: the weights $w_{n,\nu}^m$

$w_{0,0}^0 \leftarrow 1, \alpha \leftarrow 1$
for $n \leftarrow 1$ **to** N **do**
  $\beta \leftarrow 1$
  for $\nu \leftarrow 0$ **to** $n-1$ **do**
    $\beta \leftarrow \beta \cdot (x_n - x_\nu)$
    for $k \leftarrow 0$ **to** $min(n, M)$ **do**
      $w_{n,\nu}^k \leftarrow ((x_n - \xi) w_{n-1,\nu}^k - k\, w_{n-1,\nu}^{k-1})/(x_n - x_\nu)$
    end
  end
  for $k \leftarrow 0$ **to** $min(n, M)$ **do**
    $w_{n,\nu}^k \leftarrow \alpha(k\, w_{n-1,\nu-1}^{k-1} - (x_{n-1} - \xi) w_{n-1,\nu-1}^k)/\beta$
  end
  $\alpha \leftarrow \beta$
end
return all weights $w_{n,\nu}^k$
done

The algorithm produces the optimal weights for the points $x_0, \ldots, x_n$ and for the lower order stencils $x_0, \ldots, x_\nu$ with $\nu = 0, \ldots, n$ while requiring a three dimensional array for storage of all weights. However the algorithm can be implemented to safely overwrite lower order stencils with those of higher order. In this case a two dimensional array is sufficient.

3.2 Derivation of the algorithm

The Lagrange interpolation polynomial of a function $f(x)$ in the points $x_0, \ldots, x_k$ is given by

$$p_N(x) = \sum_{k=0}^{N} f(x_k) F_k(x) \tag{3.2}$$

where

$$F_k(x) = \prod_{j=0, j\neq k}^{N} (x - x_j) \Big/ \prod_{j=0, j\neq k}^{N} (x_k - x_j), \quad k = 0, 1, \ldots, N. \tag{3.3}$$

B. Fornberg [For98b] derives the algorithm by expressing the best polynomial-based approximations to $\frac{\partial^k f}{\partial x^k}$ at a point ξ in the points $x_0, \ldots, x_N$ and leading subsets $x_0, \ldots, x_i$, $i < N$ by differentiating the corresponding Lagrange polynomials:

$$\left.\frac{\partial^k p_i(x)}{\partial x^k}\right|_{x=\xi} = \sum_{j=0}^{i} \left.\frac{\partial^k F_{i,j}(x)}{\partial x^k}\right|_{x=\xi} \cdot f(x_j), \quad i = 0, \ldots, N \tag{3.4}$$

Without loss of generality one may assume $\xi = 0$ by shifting the coordinate system accordingly. The finite-difference weights are then given by

$$w_{i,j}^k = \left.\frac{\partial^k F_{i,j}(x)}{\partial x^k}\right|_{x=0} \tag{3.5}$$

Finally a relation between the weights $w_{i,j}^k$ and the points $x_0, \ldots, x_k$ is obtained by applying Taylor's theorem to $F_{i,j}(x)$:

$$F_{i,j}(x) = \sum_{k=0}^{i} \frac{w_{i,j}^k}{k!} x^k = \frac{(x - x_0) \ldots (x - x_{j-1})(x - x_{j+1}) \ldots (x - x_i)}{(x_j - x_0) \ldots (x_j - x_{j-1})(x_j - x_{j+1}) \ldots (x_j - x_i)} \tag{3.6}$$

Equations can now be found that satisfy 3.6 and relate expressions of higher derivatives and more points with less accurate ones:

$$F_{i,j}(x) = \frac{x - x_i}{x_j - x_i} F_{i-1,j}(x) \tag{3.7}$$

$$F_{i,i}(x) = \frac{\prod_{l=0}^{i-2}(x_{i-1} - x_l)}{\prod_{l=0}^{i-1}(x_i - x_l)} (x - x_{i-1}) F_{i-1,i-1}(x) \tag{3.8}$$

Algorithm 2 is now obtained by substituting 3.5 in 3.7 and 3.8 and using the starting value $w_{0,0}^0 = 1$

3.3 Extension to Mehrstellenverfahren

By default algorithm 2 approximates operators by function values only. But there is a simple algorithm to generate Mehrstellenverfahren from standard finite-difference stencils which is the method of fictuous gridpoints.

Let $x_0, \ldots, x_{M-1}$ be some points that determine a stencil and $\xi_0, \ldots, \xi_{N-1}$ be a sequence of expansion points that will afterwards form the stencil applied to the given operator L. Then the following algorithm creates a Mehrstellenverfahren of higher order than the standard finite approach.

The following example treats the generation of a Mehrstellenverfahren for the first derivative in one dimension x at the points $-h$, 0 and h. Two additional fictuous points are introduced, for example $-2h$ and $2h$, and stencils are generated with each one of the original points as expansion point:

$f''(-h)$	$f''(0)$	$f''(h)$		$f(-2h)$	$f(-h)$	$f(0)$	$f(h)$	$f(2h)$	
1			=	$-\frac{1}{4h}$	$-\frac{5}{6h}$	$\frac{3}{2h}$	$-\frac{1}{2h}$	$\frac{1}{12h}$	$+O(h^4)$
	1		=	$\frac{1}{12h}$	$-\frac{2}{3h}$	0	$\frac{2}{3h}$	$-\frac{1}{12h}$	$+O(h^4)$
		1	=	$-\frac{1}{12h}$	$\frac{1}{2h}$	$-\frac{3}{2h}$	$\frac{5}{6h}$	$\frac{1}{4h}$	$+O(h^4)$

Algorithm 3: fictuous gridpoints

input : points $x_0, \ldots, x_{M-1}$, expansion points $\xi_0, \ldots, \xi_{N-1}$, operator L
output: two stencils, one for each side of the differential equation

Introduce k new gridpoints $x_M, \ldots, x_{M+k}$ *anywhere*
for $k \leftarrow 0$ **to** $M-1$ **do**
  Generate the stencil for L over $x_0, \ldots, x_{M+k}$ at expansion point ξ_k
end
Combine the stencils linearly to eliminate dependencies on $x_M, \ldots, x_{M+k}$

The final Mehrstellenverfahren is obtained by multiplying these stencils by 1, 4 and 1 respectively and adding them together:

$f''(-h)$	$f''(0)$	$f''(h)$		$f(-2h)$	$f(-h)$	$f(0)$	$f(h)$	$f(2h)$	
1	4	1	=	0	$\frac{3}{h}$	0	$-\frac{3}{h}$	0	$+O(h^4)$

The method of fictuous points can extend any set of standard finite difference stencils to a single Mehrstellenverfahren. It is not limited to using algorithm 2 as input, but also the more general stencil generator of section 2.4.

4 Implementation

I discuss in this section the implementation of the previously stated algorithms with the computer algebra system Maxima. Ideas how to speed up computer algebra implementations are presented, as well as a way to integrate the Maxima implementation in existing numerical software.

4.1 From algorithm to executable code

Maxima provides an Algol-like syntax as well as Lisp style expressions. All programming paradigms from functional to imperative are therefore supported. The algorithms in this thesis can be translated almost literally. However I used functional notation with lambda expressions where possible as this leads to shorter and more readable code.

Maxima has native support for equations and variables, but there is no built in representation for stencils. However with Maxima being based on Common Lisp, there is excellent support for working with lists. So I decided to describe stencils simply as a list of point descriptions. Each point description is itself a list with a weight as the first element and the following elements denoting the distance from the expansion point in each dimension. So the three point stencil to approximate the Laplace operator in one dimension can be written as

```
[[1 / h^2, -h], [-2 / h^2, 0], [1 / h^2, h]]
```

and the five point stencil for the Laplace operator in two dimensions can be written as

```
[[1 / h^2, 0, h],
 [1 / h^2, -h, 0], [-4 / h^2, 0, 0], [1 / h^2, h, 0],
 [1 / h^2, 0, -h]].
```

This representation can handle stencils of arbitrary size in any number of dimensions.

The challanges in the general stencil generator are the Taylor expansion, the solving of linear systems of equations and linear minimisation. Luckily Maxima has builtin routines for these problems, namely `taylor`, `solve` and `lbfgs`.

4.2 Making computer algebra fast

There are cases where the compute time for stencil generation becomes relevant. This happens, when new stencils are required for each iteration or for each single grid point. This is likely to happen in applications where the domain or some coefficients change over time. I experimented with several optimisation [1] strategies and present now my experiences.

Write the complete generator in C, C++ or Fortran: This approach seems appealing at first. There are very good compilers for those language and a good implementation will surely achieve the best possible performance.

The downside of that strategy is that one will either loose almost all features of the original computer algebra system or spend a long time reimplementing functionality that is then so complex that it will not outperform a mature tool as Maxima. I came to the conclusion that it is more effective to tune the existing Maxima code.

Translate the Maxima code to C The Lisp Implementation ECL (Embeddable Common Lisp) can translate Common Lisp code such as the source code of Maxima to C code. This code can then benefit from a good C compiler. Another advantage is that it is easy to add the translated code to existing software with support for calling C functions.

Several problems arise in this process that cannot be easily overcome. The maxima sourcecode was not written to be called as a subroutine in another application. This is definitely possible, but hardly with a clean and generic solution. The more severe problem is that the ECL translation does not generate fast code. Indeed many performance tradeoffs must be made to translate the highly dynamic, garbage collected language Common Lisp to static C code. It is unlikely that a good C compiler will compensate for that. The conclusion is that it is better to use a good Lisp Compiler in the first place.

Use Lisp features for optimisation Common Lisp has many powerful features beyond Fortran–/Algol– and C–like languages. For an extensive treatment see [Gra93]. Very important is the possibility to generate, compile and replace code at runtime. This enables experienced programmers to write programs that are both generic and individually tuned for specific problems.

This opens up many possibilities in the case of stencil generation. If many stencils are to be generated, but all of them discretize the same differential operators, then it is a very worthwile optimisation to have a function that — given the differential operators only — returns a function that takes the same number of stencils and determines the weights. The costly operation of expanding the operators in series can be done already at function generation, the generated function merely substitutes the given points and solves for maximal possible order.

Similar optimisations are possible by generating functions with a fixed number of dimensions or for fixed grids.

4.3 Integration of Maxima in numerical codes

The easiest way to integrate the results of a Maxima stencil generator into another application is by including the results in the source code. This can be achieved via C header files, or some text formatting tool.

[1] The computer science term *optimisation* is somewhat misleading. It means improving the code, not finding the best possible sequence of instructions. The latter is called *superoptimisation* and is unfortunately infeasible for problems of this size.

A more elaborate solution is necessary if the stencils are not yet avilable at compile time. I already discussed in the previous sections, that the direct integration of Maxima in languages with C calling conventions is very problematic. Instead and without too much loss in performance is it possible to communicate via standard input and output streams and spawn Maxima in its own process. The program is then responsible for setting up the Maxima process and converting all desired stencil problems in a string with Maxima instructions. Maxima produces the stencil and returns it as another string that must then be parsed by the main application.

A useful side effect of the process based solution is that it offers a simple way to introduce parallelism when many stencils have to be solved at once. The main program can create multiple Maxima processes and distribute the workload among them.

5 Validation

I implemented a simple-finite difference solver to validate the output of the Maxima stencil generator in appendix A.3. The program performs Gauss-Seidel steps with the given stencils until the absolute error does not decrease anymore ($\Delta\epsilon < 10^{-30}$). The absolute error is then printed in the L^2-norm and the ∞-norm.

The model problems are taken from [Sch00].

5.1 Tested stencils

Both problems are based on the two dimensional Poisson equation and can be solved with the same stencils.The following discrete approaches were used:

The five point stencil

$$\frac{1}{h^2}\begin{pmatrix}0 & 1 & 0\\ 1 & -4 & 1\\ 0 & 1 & 0\end{pmatrix} u = f. \tag{5.1}$$

A standard Mehrstellenverfahren from [Col66] with a five point stencil on the right hand side

$$\frac{1}{h^2}\begin{pmatrix}-2 & -8 & -2\\ -8 & 40 & -8\\ -2 & -8 & -2\end{pmatrix} u = \begin{pmatrix}0 & 1 & 0\\ 1 & 8 & 1\\ 0 & 1 & 0\end{pmatrix} f. \tag{5.2}$$

Another Mehrstellenverfahren from [Col66] with a nine point stencil on the right hand side

$$\frac{1}{h^2}\begin{pmatrix}-1 & -4 & -1\\ -4 & 20 & -4\\ -1 & -4 & -1\end{pmatrix} u = \frac{1}{12}\begin{pmatrix}1 & 4 & 1\\ 4 & 52 & 4\\ 1 & 4 & 1\end{pmatrix} f. \tag{5.3}$$

A Mehrstellenverfahren from [Sch00] with a nine point stencil on the right hand side. Some derivatives of the next higher order are eliminated for higher accuracy.

$$\frac{1}{h^2}\begin{pmatrix}-1 & -4 & -1\\ -4 & 20 & -4\\ -1 & -4 & -1\end{pmatrix} u = \frac{1}{12}\begin{pmatrix}1 & 4 & 1\\ 4 & 52 & 4\\ 1 & 4 & 1\end{pmatrix} f. \tag{5.4}$$

An optimized stencil generated from the implementation in appendix A.3

$$\frac{1}{6h^2}\begin{pmatrix}-1 & -4 & -1\\ -4 & 20 & -4\\ -1 & -4 & -1\end{pmatrix} u = \begin{pmatrix}0.00277 & 0.0777 & 0.00277\\ 0.07777 & 0.6777 & 0.07777\\ 0.00277 & 0.0777 & 0.00277\end{pmatrix} f. \tag{5.5}$$

(The entrys of the right hand side have actually more digits of accuracy, but I omitted them for readability.)

5.2 A polynomial test problem in two dimensions

The polynomial test problem is given by the differential equation

$$-\Delta u(x,y) = -12x^2y^2 - 2x^4, \quad \Omega :=]0,1[^2 \tag{5.6}$$

and the boundary conditions

$$\begin{aligned} u(x,0) &= 0 \\ u(x,1) &= x^4 \\ u(0,y) &= 0 \\ u(1,y) &= y^2. \end{aligned}$$

The exact solution is

$$u(x,y) = x^4y^2 \tag{5.7}$$

The following results were obtained:

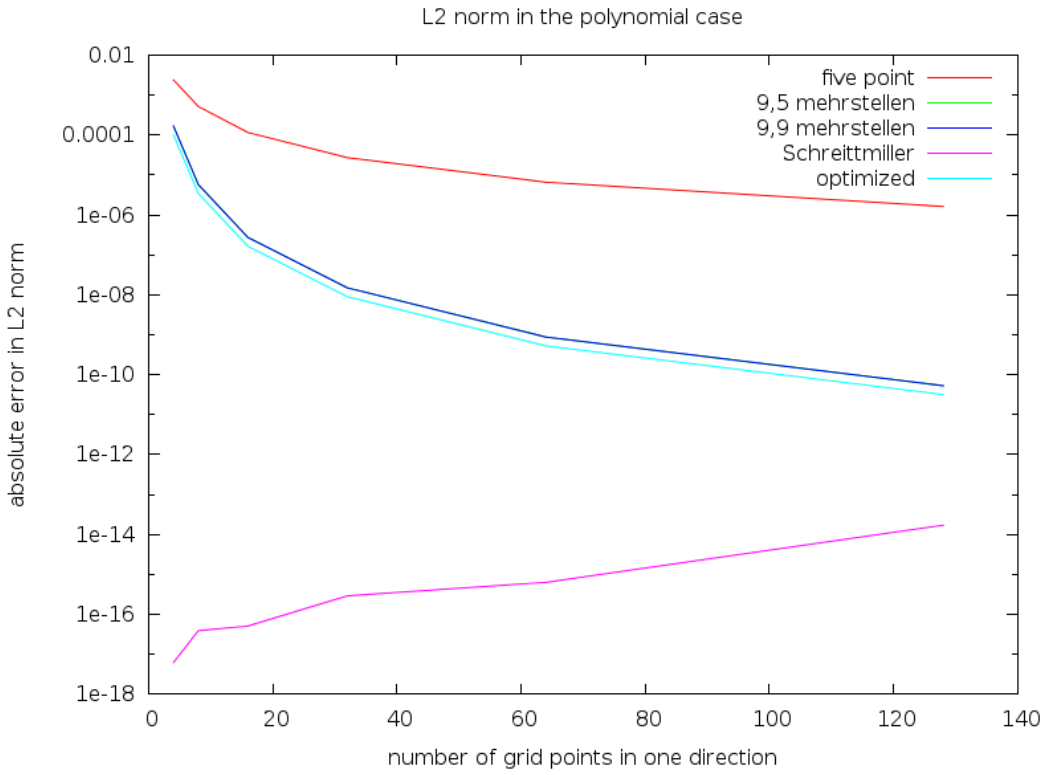

Figure 1: L^2-norms for the polynomial problem

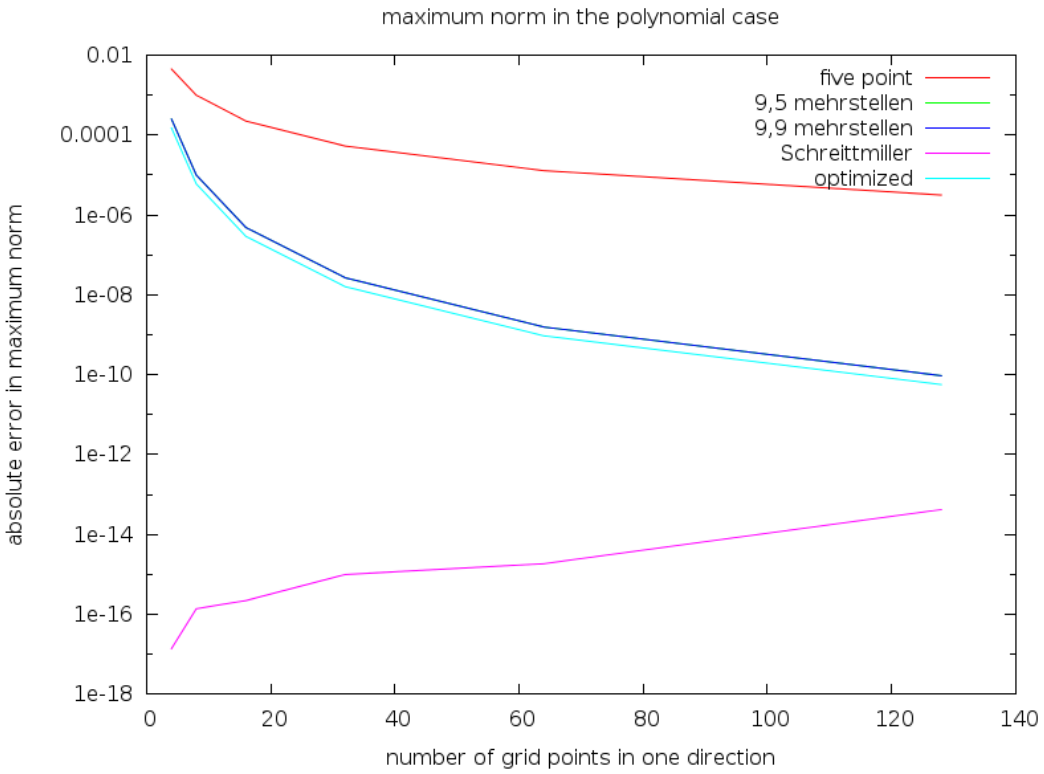

Figure 2: ∞-norms for the polynomial problem

The polynomial problem contains some curiosities. The most notable is that the discretisation of Schreittmiller is already accurate at very coarse grids. This is because this approximation solves the model problem exactly and independent of the grid spacing. A finer grid can therefore only introduce more errors.

The other observation is that the other Mehrstellenverfahren are all of the same quality. The reason is that the solution has only derivatives up to a certain order. High order derivatives are already zero and minimizing them does therefore not yield any improvements.

The five point stencil shows the expected behaviour and is second order accurate.

I presented this example as it shows the important role of the solution when it comes to higher order stencils. There are no benefits from using highly accurate stencils for a problem with limited differentiability.

5.3 A trigonometric test problem in two dimensions

The trigonometric test problem is given by the differential equation

$$-\Delta u(x,y) = -2\pi^2 \sin(\pi x)\sin(\pi y), \quad \Omega :=]0,1[^2 \tag{5.8}$$

and the boundary conditions $u = 0$ on $\partial\Omega$. The exact solution is

$$u(x,y) = \sin(\pi x)\sin(\pi y). \tag{5.9}$$

The following results were obtained:

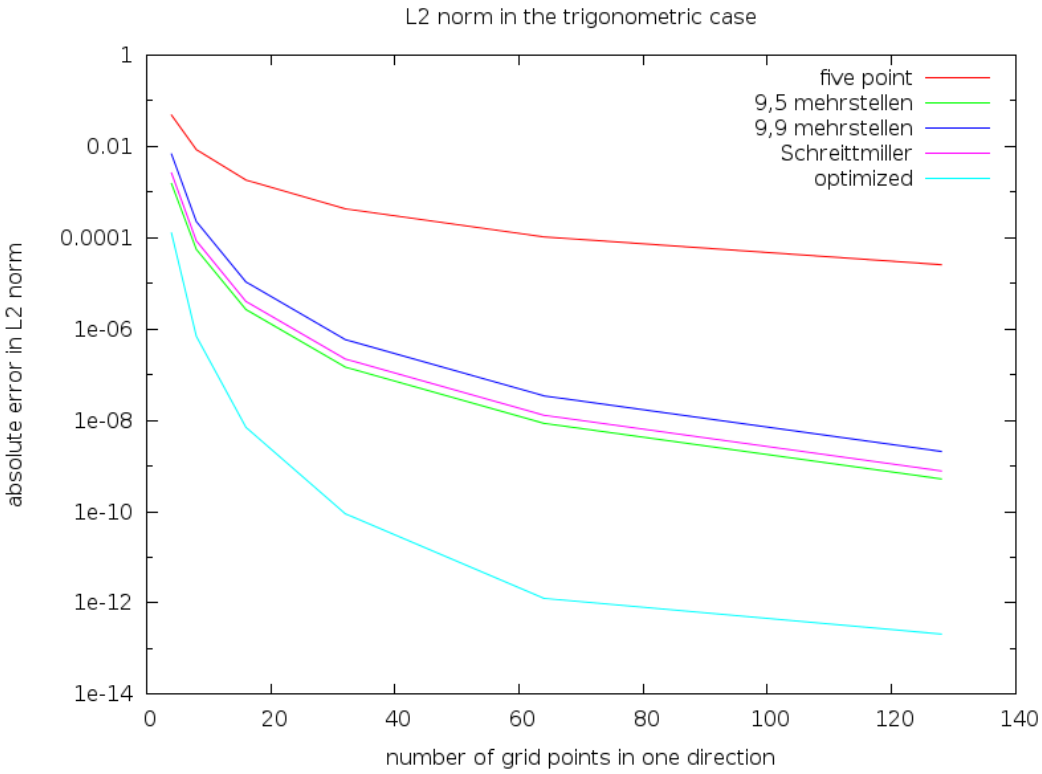

Figure 3: L^2-norms for the trigonometric problem

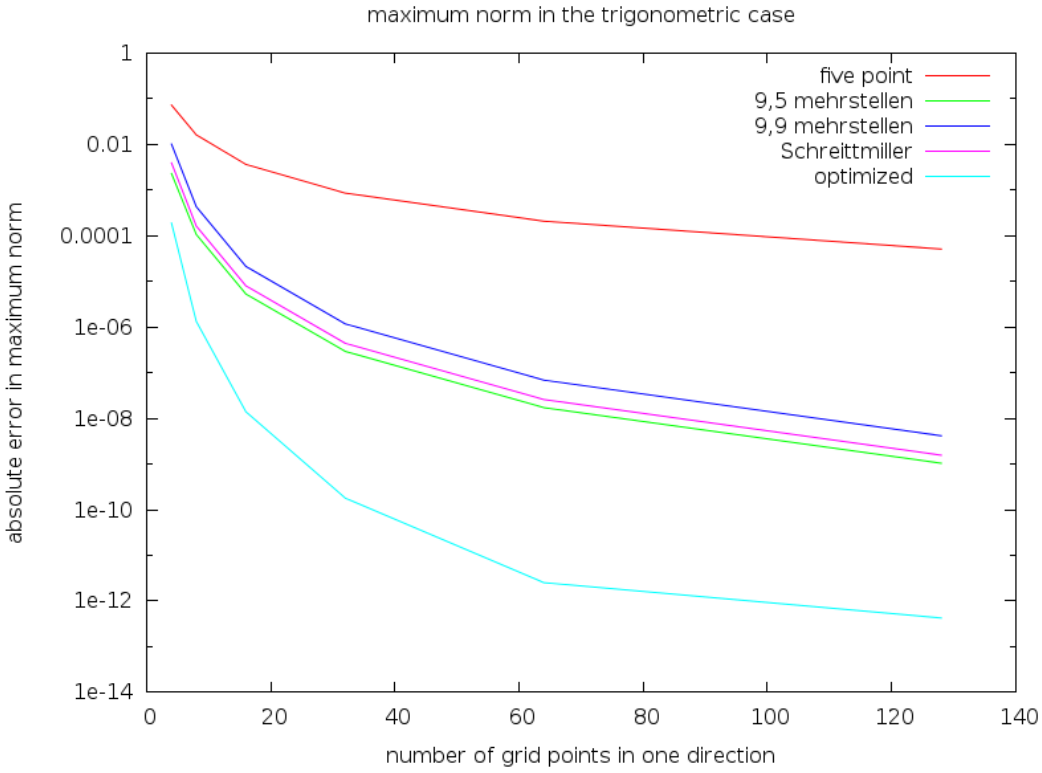

Figure 4: ∞-norms for the trigonometric problem

This example shows impressive gains in accuracy for the optimized stencil. The minimisation of the derivatives of next higher order made the resulting stencils almost sixth order accurate. A possible explanation is that the next higher error terms of the 9,9-Mehrstellenverfahren are roughly

$$\frac{u_{(6,0)}(0,0)\,h^4}{40} + \frac{u_{(4,2)}(0,0)\,h^4}{24} + \frac{u_{(2,4)}(0,0)\,h^4}{24} + \frac{u_{(0,6)}(0,0)\,h^4}{40},$$

while the next higher terms in the optimized version are

$$.00416\,u_{(6,0)}(0,0)\,h^4 - .00416\,u_{(4,2)}(0,0)\,h^4 - .00416\,u_{(2,4)}(0,0)\,h^4 + .00416\,u_{(0,6)}(0,0)\,h^4$$

and therefore smaller by a factor between five and ten.

6 Conclusion

I am aware that the current research tends toward finite-element or finite-volume methods. Nontheless exist applications, where finite a finite-difference approach is favourable. An example are many applications that do already work on structured grids. Mehrstellenverfahren together with multigrid methods are very efficient [Zha98],[ZKO99] while being relatively simple to implement.

I designed my implementations with an extension to immersed interface methods in mind. A description of the immersed interface methods can be found in [LI06]. Unfortunately a proper treatment of such methods is beyond the scope of this thesis. I can only outline a possible way how to adapt algorithm 1 to immersed interface methods. The algorithm allows any number of operators and points, so it is possible to use different operators on each side of the stencil. Jump conditions must then be used in the algorithm to replace the coefficients on one side of the interface. The successive solver can then proceed as usual.

Very complex finite-difference approximations can be computed with the given algorithms. I hope this thesis will encourage experimentation with stencils of higher order, Mehrstellenverfahren and different grids. The approach of minimizing highest order derivatives seems to introduce a significant gain in accuracy, even compared to existing Mehrstellenverfahren.

A Maxima Implementation

A.1 Helper routines

The method `integer_permutations` returns all combinations of `len` positive itegers (or zero) with the sum `n`.

```
integer_permutations(n, len) :=
if n = 0 then [makelist(0, len)]
else listify(
  apply(union,
    listify(map(
        'permutations, integer_partitions(n, len)))))$
```

The method `maclaurin` expands a given operator `expr` in the variables `vars` as truncated Taylor series with expansion point zero and length `maxorder`. `dists` is the distance from the point of interest from the expansion point.

```
load("pdiff")$
maclaurin(expr, vars, dists, maxorder) := block(
  [dims : length(vars), tterm],
  tterm : taylor(expr, vars, makelist(0, dims), maxorder),
  subst(makelist(vars[k] = dists[k], k, 1, dims), tterm))$
```

This snippet takes a list of stencil description triples of the form `(operator, maxorder, stencil)` and returns a list of the weights in all stencils.

```
extract_weights(triples) := flatten(
  map(lambda([triple],
      map(lambda([entry],
          first(entry)),
        triple[3])),
    triples))$
```

The method `rsubst` takes a list of substitutions and applies them successively to the first element.

```
rsubst(list) := (
  if is(length(list) = 1) then list[1] else
  rsubst(cons(subst(list[2], list[1]), rest(list, 2))))$
```

As an example, a call to

```
printf(true, "~a",
  rsubst([[a = 2 * b, c = 5 * d], [b = 4 * e, d = 2 * e]]));
```

gives the relation

```
[a = 8*e,c = 10*e]
```

A.2 Stencil expander

A very useful utility is the `stencil_expand` method. The input is a list of variables, one for each dimension, followed by a sequence of triples. Each triple is of the form `(operator, order, stencil)` The Maclaurin series of all operators at all corresponding stencil entrys are multiplied with the respective weights and added together. The sum of all series is returned.

This code works for any number of dimensions.

```
stencil_expand(vars, triples) := block([
  l        : length(triples),
  dims     : length(vars),
  tmpdists : map(lambda([var], concat('h, var)), vars)],
  expand(
    apply("+",
      flatten(
        map(lambda([triple], block([
              operator : triple[1],
              order    : triple[2],
              stencil  : triple[3],
              series],
              series : maclaurin(operator, vars, tmpdists, order),
              map(lambda([stencil_entry], block([
                    substlist : makelist(
                      tmpdists[k] = stencil_entry[k + 1],
                      k, 1, dims)],
                    stencil_entry[1] * subst(
                      substlist, series))),
                stencil))),
          triples)))))$
```

This is an example invocation that evaluates the accuracy of a Mehrstellenverfahren that relates values of $h^2\Delta u$ and values of u in two dimensions.

```
tex(stencil_expand([x, y],
    [[u(x, y), 6,
    [[-2, -h, h], [-2, h, h], [-2, -h, -h], [-2, h, -h],
    [-8, 0, h], [-8, -h, 0], [-8, h, 0], [-8, 0, -h],
    [40, 0, 0]]],
    [h^2 * (diff(u(x, y), x, 2) + diff(u(x, y), y, 2)), 4,
    [[1, 0, h], [1, -h, 0], [1, h, 0], [1, 0, -h],
    [8, 0, 0]]]]))$
```

The result are the nonzero terms of the Taylor expansion of that expression up to the given order.

$$\frac{u_{(6,0)}(0,0)\,h^6}{20} - \frac{u_{(4,2)}(0,0)\,h^6}{12} - \frac{u_{(2,4)}(0,0)\,h^6}{12} + \frac{u_{(0,6)}(0,0)\,h^6}{20}$$

A.3 Stencil generator

```
load("basic")$
load("lbfgs")$
weights(vars, norm_triples, other_triples) := block([
  dims : length(vars),
  solution, coeffs, s, weights, zerovector, nc],
  zerovector : makelist(0, dims),
  weights : listofvars(
    extract_weights(
      append(norm_triples, other_triples))),
  nc : apply("+", extract_weights(norm_triples)) = 1,
```

```
  expansion : apply(
    stencil_expand,
    [vars, append(norm_triples, other_triples)]),
  linsolvewarn : false,
  solution : [],
  for order : 0 unless is(weights = []) do (
    coeffs : map(lambda([derivop], coeff(expansion, derivop)),
      map(lambda([deriv],
          buildq([zerovector, deriv],
            pderivop(u, splice(deriv))(splice(zerovector)))),
        integer_permutations(order, dims))),
    if is(order = 0) then coeffs : cons(nc, coeffs),
    remarray(%rnum_list),
    s : solve(coeffs, weights),
    if is(s = []) then block(
      [term : expand(apply("+", coeffs))],
      vs : sublist(
        listofvars(term), lambda([item],
          not member(item, %rnum_list))),
      term : subst(
        makelist(vs[k] = 1, k, 1, length(vs)),
        term),
      push(
        lbfgs(term*term, %rnum_list, zerovector, 1e-4, [-1, 0]),
        solution),
      weights : [])
    else (
      weights : %rnum_list,
      push(s[1], solution),
      expansion : subst(s[1], expansion))),
  rsubst(reverse(solution)))$
```

This example invocation

```
printf(true, "␣~{~a~%␣~}",
  weights([x, y, z],
    [[diff(u(x, y, z), x, 2)
    + diff(u(x, y, z), y, 2)
    + diff(u(x, y, z), z, 2), 4,
    [[d, h, 0, 0], [d, -h, 0, 0],
    [d, 0, h, 0], [d, 0, -h, 0],
    [d, 0, 0, h], [d, 0, 0, -h],
    [e, 0, 0, 0]]]],
    [[u(x, y, z), 6,
    [[c, h, 0, -h], [c, -h, 0, -h],
    [c, 0, h, -h], [c, 0, -h, -h],
    [b, 0, 0, -h],
    [c, h, 0, h], [c, -h, 0, h],
    [c, 0, h, h], [c, 0, -h, h],
    [b, 0, 0, h],
    [c, -h, h, 0], [c, h, h, 0],
```

```
    [c, -h, -h, 0], [c, h, -h, 0],
    [b, 0, -h, 0],
    [b, 0, h, 0], [b, -h, 0, 0],
    [b, h, 0, 0],
    [a, 0, 0, 0]]]]));
```

produces the following satisfactory result

```
d = 1/12
e = 1/2
c = -1/(6*h^2)
b = -1/(3*h^2)
a = 4/h^2
```

A.4 Fornbergs algorithm

This implementaiton of Fornberg's algorithm is based on a FORTRAN version from [For98b].

```
fornberg(M, ep, points) := block(
  [N : length(points),
  c1, c2, c3, c4, c5,
  w, x],
  x : make_array(any, N), fillarray(x, points),
  w : make_array(any, M+1, N, N),
  w[0, 0, 0] : 1,
  c1 : 1,
  c4 : x[0] - ep,
  for n : 1 thru N-1 do (
    c2 : 1,
    c5 : c4,
    c4 : x[n] - ep,
    for v : 0 thru n - 1 do (
      c3 : x[n] - x[v],
      c2 : c2 * c3,
      if n <= M then w[n, n-1, v] : 0,
      w[0, n, v] : c4 * w[0, n-1, v] / c3,
      for k : 1 thru min(n, M) do (
        w[k, n, v] : (c4 * w[k, n-1, v] - k*w[k-1, n-1, v])/c3),
      w[0, n, n] : -c1 * c5 * w[0, n-1, n-1] / c2),
      for k : 1 thru min(n, M) do (
        w[k, n, n] : c1 * (k * w[k-1, n-1, n-1]
                           - c5 * w[k, n-1, n-1]) / c2),
      c1 : c2),
    makelist(w[M, N-1, k], k, 0, N-1))$
```

The method can be called in the following way:

```
result : fornberg(2, 0, [-2*h, -h, 0, h, 2*h])$
printf(true, "~a", result)$
```

Which results in the following output:

```
[-1/(12*h^2),4/(3*h^2),-5/(2*h^2),4/(3*h^2),-1/(12*h^2)]
```

A.5 Method of fictuous gridpoints

```
fictuous(sp, fp, ep, L) := block([
  spl : length(sp),
  fpl : length(fp),
  epl : length(ep),
  p, M],
  p : append(sp, fp),
  eqw : makelist(w[k], k, 1, epl),
  M : matrix(),
  for k : 1 thru epl do (
    M : addrow(M, fornberg(L, ep[k], p))),
  eqns : makelist(eqw . col(M, k), k, spl+1, spl+fpl),
  eqw : subst(solve(eqns, eqw), eqw),
  eqw : subst(
    map("=", %rnum_list,
      makelist(1, k, 1, length(%rnum_list))), eqw),
  stencil : makelist(eqw . col(M, k), k, 1, spl),
  [stencil, eqw]
  )$
```

Again a simple example of invocation:

```
print(fictuous([-h, 0, h], [-2*h, 2*h], [-h, 0, h], 2));
```

that generates the following output:

```
  12    24  12
[[--, - --, --], [1, 10, 1]]
   2     2   2
  h     h   h
```

References

[Col66] L. Collatz. *The Numerical Treatment of Differential Equations.* Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen mit besonderer Berücksichtigung der Anwendungsgebiete. Springer Berlin Heidelberg, 1966.

[dSFMY04] P. N. de Souza, R. Fateman, J. Moses, and C. Yapp. The maxima book, 2004.

[For89] O. Forster. *Analysis 1 ; Differential- und Integralrechnung einer Veränderlichen.* Friedr.Viehweg Sohn Verlag, 1989.

[For91] O. Forster. *Analysis 2 ; Differental- und Integralrechnung im R^n Gewöhnliche Differentialgleichungen.* Vieweg Studium : Grundkurs Mathematik. Vieweg, 1991.

[For98a] Bengt Fornberg. Calculation of weights in finite difference formulas. *SIAM Rev*, 40:685–691, 1998.

[For98b] Bengt Fornberg. *A Practical Guide to Pseudospectral Methods.* Cambridge University Press, 1998.

[Gra93] P. Graham. *On Lisp.* Prentice Hall, 1993.

[KP78] H. B. Keller and V. Pereyra. Symbolic generation of finite difference formulas. *Math. Comp.*, 32:955–971, 1978.

[LI06] Z. Li and K. Ito. *The Immersed Interface Method: Numerical Solutions of PDEs Involving Interfaces and Irregular Domains.* Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics, 2006.

[Sch00] R. Schreittmiller. Konstruktion von verfahren höherer ordnung für elliptische differentialgleichungen, 2000.

[Ste90] G.L. Steele. *COMMON LISP: The Language.* HP Technologies Series. Digital Press, 1990.

[Zha98] Jun Zhang. Fast and high accuracy multigrid solution of the three dimensional poisson equation. *J. Comput. Phys*, 143:449–461, 1998.

[ZKO99] Jun Zhang, Jules Kouatchou, and Mohamed Othman. On cyclic reduction and finite difference schemes, 1999.