Resolution of Skin Layer in EM Simulation

Master Thesis in Computational Science and Engineering

Michael Spreng
supervised by Prof. Ralf Hiptmair

March 3, 2010

Contents

1 Introduction

1.1 Problem Statement

The following 2D elliptic boundary value problem on domain Ω is the subject of this thesis:

$$\begin{aligned} -\Delta u + \sigma(\vec{x})u &= 0 \qquad \text{in } \Omega \\ \frac{\partial u}{\partial \vec{n}} + zu &= g \qquad \text{on } \partial\Omega \end{aligned} \tag{1.1}$$

with $\sigma(\vec{x}), z > 0$. This differential equation captures the effect of a skin layer, where the parameter σ defines the thinness of the solution at the boundary, which is the skin layer. To illustrate the behaviour, the 1D solution of boundary value problem (1.1) is displayed in figure 1.1 for two values of σ and boundary value $g = 1$ on domain $\Omega = [-1, 1]$

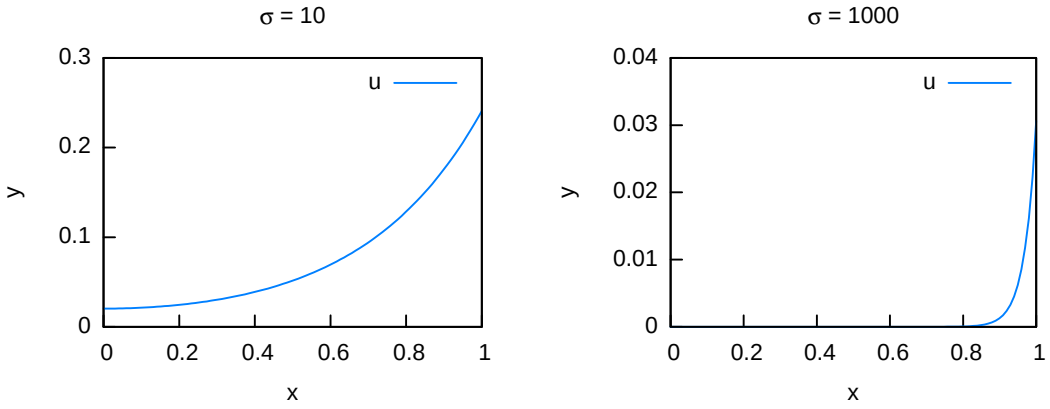


Figure 1.1: Skin layer for two values of σ

1.2 Skin Layers

Alternating electromagnetic fields penetrate good conductors in a layer only at their surface. This layer is called the skin layer. All the electromagnetic field energy is thus stored there and all ohmic power losses due to eddy currents, that may heat the conductor, are concentrated in this layer. If one wants to calculate heating due to ohmic losses, one has to resolve this layer.

An application where heating due to power losses is relevant is an ABB power transformer, which has a diameter of several meters. The skin depth in the copper parts is only approximately 9.4mm at 50Hz. Thus the ratio of the skin layer to the diameter of the transformer is rather small. This poses a problem to standard finite elements because, as a rule of thumb, a minimum of three elements are required to resolve the skin layer.

The electromagnetic fields are computed by the prominent and widely used eddy current model. The E formulation in frequency domain is:

$$\vec{\text{curl}}\frac{1}{\mu}\vec{\text{curl}}\vec{E} + i\omega\frac{1}{\rho}\vec{E} = 0 \tag{1.2}$$

where μ is the permeability, ω the angular frequency, and ρ is the resistivity. The

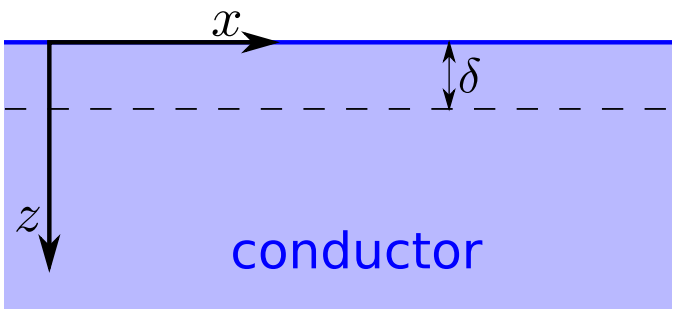


Figure 1.2: Skin layer in a conductor with a flat surface

skin effect can be studied with an idealized situation: At the surface of a flat semi-definite, conducting, permeable, and linear medium, a spatially constant magnetic field, $H_x(t) = H_0 \cos \omega t$, is applied parallel to the surface. See figure 1.2. The magnetic and electric field on the interior of the conductor are given by [1]:

$$H_x = H_0 e^{-z/\delta} e^{-i(z\delta - \omega t)} \tag{1.3}$$

$$E_y = -(1+i)\sqrt{\frac{\mu\omega\rho}{2}} H_0 e^{-z/\delta} e^{-i(z\delta - \omega t)} \tag{1.4}$$

Where δ is the skin depth

$$\delta := \sqrt{\frac{2\rho}{\mu\sigma}} \tag{1.5}$$

Obviously, the decay of the electric and magnetic field is exponential. For surfaces that have a small curvature compared to the skin depth, the above solution is a good approximation of the skin layer. It is popular to use impedance boundary conditions in

this case, but they are only a partial remedy. They fail at resolving corners, which is where the current is concentrated.

Please note that (1.2) leads to the same behaviour like equation (1.1). Therefore, for this study we use (1.1) as an equivalent to (1.2)

1.3 Main Idea

Standard finite elements are not able to resolve the skin layer due to the ratio of the whole computational domain to the skin layer. A method is needed that is able to resolve skin layers with a skin depth of less than an element width.

The idea is now to use basis functions which can fit the exponential decay. For example exponential basis functions, with the proper decay rate could be used. But such basis functions have a drawback: continuity can no longer be guaranteed for them. Therefore a discontinuous Galerkin approach, which allows for discontinuities on edges or faces of the mesh, has to be used to solve the boundary value problem. On the other hand, the exponential basis functions are solutions to the first equation of (1.1), which simplifies some aspects of the discontinuous Galerkin method. This is then called a Trefftz-type method.

1.4 Circular Domain

We use the 2D unit disc as domain Ω for our reference problem, because it has an analytic solution so we can calculate the discretization errors for it. As boundary condition we choose

$$g(\phi) = g_c \cdot e^{ik\phi}, \quad k = 0, 1, 2, \dots \tag{1.6}$$

We also have to constrain σ to be constant on the whole domain, and we chose $z = 1$. The analytic solution can be derived in polar coordinates with separation of variables: $u = R(r)\Phi(\phi)$. The Laplacian in polar coordinates is

$$\Delta f = \frac{\partial^2 f}{\partial r^2} + \frac{1}{r}\frac{\partial f}{\partial r} + \frac{1}{r^2}\frac{\partial^2 f}{\partial \phi^2} \tag{1.7}$$

Applying separation of variables and (1.7) to the given problem (1.1) yields

$$-\Phi\frac{\partial^2 R}{\partial r^2} - \frac{\Phi}{r}\frac{\partial R}{\partial r} - \frac{R}{r^2}\frac{\partial^2 \Phi}{\partial \phi^2} + \sigma R\Phi = 0 \text{ in } \Omega \tag{1.8}$$

$$\frac{\partial R}{\partial r}\Phi + R\Phi = g \text{ on } \partial\Omega \tag{1.9}$$

From (1.9) we can derive an expression for Φ which only depends on g and R at the boundary $r = 1$

$$\Phi(\phi) = \frac{g(\phi)}{\left[\frac{\partial R}{\partial r}\right]_{r=1} + R(1)} = a \cdot e^{ik\phi}, \quad \text{with } a = g_c \left(\left[\frac{\partial R}{\partial r}\right]_{r=1} + R(1) \right)^{-1} \tag{1.10}$$

Inserting this expression for Φ into (1.8):

$$-a \cdot e^{ik\phi}\frac{\partial^2 R}{\partial r^2} - \frac{a \cdot e^{ik\phi}}{r}\frac{\partial R}{\partial r} - \frac{R}{r^2}a(-k^2)e^{ik\phi} + \sigma a R e^{ik\phi} = 0 \text{ in } \Omega \tag{1.11}$$

Multiplying the equation by $-\frac{r^2}{a \cdot e^{ik\phi}}$ gives a partial differential equation similar to the modified Bessel's differential equation.

$$r^2\frac{\partial^2 R(r)}{\partial r^2} + r\frac{\partial R(r)}{\partial r} - (\sigma r^2 + k^2)R(r) = 0 \text{ in } \Omega \tag{1.12}$$

To get the exact same form of the modified Bessel's differential equation, we need to do the variable transformation $\tilde{r} = r \cdot \sqrt{\sigma}$, $\tilde{R}(r\sqrt{\sigma}) = R(r)$. Transforming now equation (1.12) yields the modified Bessel's differential equation:

$$\tilde{r}^2\frac{\partial^2 \tilde{R}(\tilde{r})}{\partial \tilde{r}^2} + \tilde{r}\frac{\partial \tilde{R}(\tilde{r})}{\partial \tilde{r}} - (\tilde{r}^2 + k^2)\tilde{R}(\tilde{r}) = 0 \text{ in } \Omega \tag{1.13}$$

The solutions of it are the modified Bessel function of the first kind, and the modified Bessel function of the second kind. The function of the first kind is:

$$I_k(\tilde{r}) = \sum_{m=0}^{\infty} \frac{1}{m!\,\Gamma(m+k+1)}\left(\frac{\tilde{r}}{2}\right)^{2m+k} \tag{1.14}$$

The modified Bessel function of the second kind is not finite at $r = 0$, but a finite solution is needed at $r = 0$. Therefore only the modified Bessel function of the first kind is used. The final solution for R is then:

$$R(r) = I_k(\sqrt{\sigma}r) \tag{1.15}$$

And we can calculate the expression for a:

$$a = g_c\left(\left[\frac{\partial I_k(\sqrt{\sigma}r)}{\partial r}\right]_{r=\sqrt{\sigma}} + I_k(\sqrt{\sigma})\right)^{-1} \tag{1.16}$$

With help of the identity $I_k'(r) = \frac{1}{2}(I_{k-1}(r) + I_{k+1}(r))$ we get:

$$a = g_c\left[\left(I_{k-1}(\sqrt{\sigma}) + I_{k+1}(\sqrt{\sigma})\right)\frac{\sqrt{\sigma}}{2} + I_k(\sqrt{\sigma})\right]^{-1} \tag{1.17}$$

An example of the solution can be seen in figure 1.3.

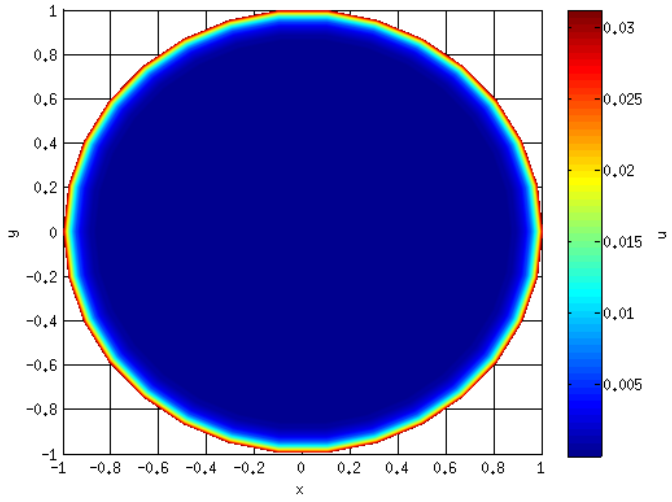


Figure 1.3: example of the solution on a circle for $\sigma = 1000$ and boundary condition $g(\phi) = 1$

2 Numerical Methods

2.1 Finite Elements

The finite element method is a popular and robust method for solving partial differential equations, especially elliptic differential equations. See [2] or [3].

In finite Elements, the domain is decomposed into simple polygons, usually only triangles or tetrahedrons which fill the domain. On each element there is a set of basis functions, which, appropriately scaled, will approximate the solution of the differential equation. The choice of basis functions is important in order to get a good approximation with as few elements as possible. Usually linear or quadratic basis functions are chosen. For the problem of skin layers, they are not that well suited as the skin effect behaves like an exponential. The proper choice of basis functions for something that behaves as such would seem to be exponentials. A nice feature of the linear or quadratic basis functions is, that they are continuous across element boundaries. But this can not be conserved for exponentials as basis functions.

2.2 Discontinuous Galerkin

If we want to choose exponentials as basis functions we can no longer use continuity across element boundaries as for standard finite elements. One way to use discontinuous basis functions is with the DG (discontinuous Galerkin) method. The DG method introduces values of the approximate solution u on the element boundaries, which are calculated from both adjoining elements. This also allows for meshes with hanging nodes. The value of u on the element boundaries is called numeric flux $\hat{u}$. The numerical fluxes are crucial for the simulation as they influence error and convergence ([4], [5]).

2.2.1 Continuous Problem

The DG formulation is derived analogously to [6]. To derive the DG method we will write the second order PDE (1.1) as a system of two first order equations. This is important for introducing the numerical fluxes consistently. Let

$$\vec{\gamma} := \vec{\nabla} u \tag{2.1}$$

be the auxiliary variable to split the second order PDE into two first order PDEs. Then the following set of equations is equivalent to equation (1.1):

$$\begin{aligned} \vec{\gamma} &= \vec{\nabla} u && \text{in } \Omega \\ \sigma(\vec{x}) u &= \vec{\nabla} \cdot \vec{\gamma} && \text{in } \Omega \\ \vec{\gamma} \cdot \vec{n} + z u &= g && \text{on } \partial\Omega \end{aligned} \tag{2.2}$$

Now we introduce the partition $\mathcal{T}_h$ which represents the mesh on which the DG finite elements will be solved. K represents one element of the partition $\mathcal{T}_h$. We now take the first two equations of (2.2), multiply them by smooth test functions and integrate both sides over element K

$$\begin{aligned} \int_K \vec{\gamma} \cdot \vec{\tau} \mathrm{d}V &= \int_K \vec{\nabla} u \cdot \vec{\tau} \mathrm{d}V && \text{in } K, \forall \vec{\tau} \in \vec{H}(\mathrm{div}; K) \\ \int_K \sigma(\vec{x}) u v \mathrm{d}V &= \int_K \vec{\nabla} \cdot \vec{\gamma} v \mathrm{d}V && \text{in } K, \forall v \in H^1(K) \end{aligned} \tag{2.3}$$

Then we integrate by parts both equations to get

$$\begin{aligned} \int_K \vec{\gamma} \cdot \vec{\tau} \mathrm{d}V + \int_K u \vec{\nabla} \cdot \vec{\tau} \mathrm{d}V - \int_{\partial K} u \vec{\tau} \cdot \vec{n} \mathrm{d}S = 0 && \forall \vec{\tau} \in \vec{H}(\mathrm{div}; K) \\ \int_K \sigma(\vec{x}) u v \mathrm{d}V + \int_K \vec{\gamma} \cdot \vec{\nabla} v \mathrm{d}V - \int_{\partial K} \vec{\gamma} \cdot \vec{n} v \mathrm{d}S = 0 && \forall v \in H^1(K) \end{aligned} \tag{2.4}$$

2.2.2 Discretized Problem

The discrete function space $\vec{\Sigma}_h \subset \vec{H}^1(\mathrm{div}; K)$ is introduced for the vector valued functions $\vec{\gamma}_h, \vec{\tau}_h \in \vec{\Sigma}_h$, and $V_h \subset H^1(K)$ is introduced for the scalar functions $u_h, v_h \in V_h$. To couple the equations on different elements K we need the numerical fluxes $\hat{\vec{\gamma}}_h$ and $\hat{u}_h$ across element boundaries. From here on we use the symbol $\vec{\nabla}_h$ for the element wise application of the operator $\vec{\nabla}$. The discretized version of eq. 2.4 is then

$$\begin{aligned} \int_K \vec{\gamma}_h \cdot \vec{\tau}_h \mathrm{d}V + \int_K u_h \vec{\nabla} \cdot \vec{\tau}_h \mathrm{d}V - \int_{\partial K} \hat{u}_h \vec{\tau}_h \cdot \vec{n} \mathrm{d}S = 0 && \forall \vec{\tau}_h \in \vec{\Sigma}_h(K) \\ \int_K \sigma_K u_h v_h \mathrm{d}V + \int_K \vec{\gamma}_h \cdot \vec{\nabla} v_h \mathrm{d}V - \int_{\partial K} \hat{\vec{\gamma}}_h \cdot \vec{n} v_h \mathrm{d}S = 0 && \forall v_h \in V_h(K) \end{aligned} \tag{2.5}$$

This would already be a valid formulation. But we want to transform it again to a single equation. For this we first reverse partial integration in the first equation of (2.5).

$$\int_K \vec{\gamma}_h \cdot \vec{\tau}_h \mathrm{d}V = \int_K \vec{\nabla} u_h \cdot \vec{\tau}_h \mathrm{d}V + \int_{\partial K} (\hat{u}_h - u_h) \vec{\tau}_h \cdot \vec{n} \mathrm{d}S \qquad \forall \vec{\tau}_h \in \vec{\Sigma}_h(K) \tag{2.6}$$

In order to combine the two equations, we assume that $\vec{\nabla}_h V_h \subseteq \vec{\Sigma}_h$ and substitute $\vec{\tau}_h$ with $\vec{\nabla}_h v_h$ in each element. Insert this into (2.6) which reads then

$$\int_K \vec{\gamma}_h \cdot \vec{\nabla}_h v_h \mathrm{d}V = \int_K \vec{\nabla} u_h \cdot \vec{\nabla}_h v_h \mathrm{d}V + \int_{\partial K} (\hat{u}_h - u_h) \vec{\nabla}_h v_h \cdot \vec{n} \mathrm{d}S \qquad \forall v_h \in \vec{V_h}(K) \tag{2.7}$$

Replace $\int_K \vec{\gamma}_h \cdot \vec{\nabla}_h v_h \mathrm{d}V$ in the second equation of (2.5) with the right hand side of eq. 2.7 to get the combined equation

$$\int_K \left(\sigma_K u_h v_h + \vec{\nabla} u_h \cdot \vec{\nabla} v_h\right) \mathrm{d}V + \int_{\partial K} (\hat{u}_h - u_h) \vec{\nabla} v_h \cdot \vec{n} \mathrm{d}S - \int_{\partial K} \hat{\vec{\gamma}}_h \cdot \vec{n} v_h \mathrm{d}S = 0 \quad (2.8)$$

This is equivalent to the system of equations in (2.5).

2.3 Trefftz Methods

A method is called a Trefftz method if the basis functions are a solution to the PDE. We construct a Trefftz method by choosing basis functions solving $-\Delta u + \sigma u = 0$ from differential equation (1.1), where σ can vary from element to element, but is constant on each element. This leads to basis functions matching the exponential decay characteristic for skin layers. With this special choice of the discrete function space V_h we can further simplify eq. 2.8. In order to take advantage of our special choice of basis functions we integrate by parts once more to obtain

$$\int_K (\sigma_K v_h - \Delta v_h) u_h \mathrm{d}V + \int_{\partial K} \hat{u}_h \vec{\nabla} v_h \cdot \vec{n} \mathrm{d}S - \int_{\partial K} \hat{\vec{\gamma}}_h \cdot \vec{n} v_h \mathrm{d}S = 0 \quad (2.9)$$

Where the first integral equals to zero. To obtain a single formula for the whole domain, we sum over all elements. This leads to the variational formulation: Find $u \in V_h$, such that

$$\sum_K \int_{\partial K} \hat{u}_h \vec{\nabla} v_h \cdot \vec{n} \mathrm{d}S - \sum_K \int_{\partial K} \hat{\vec{\gamma}}_h \cdot \vec{n} v_h \mathrm{d}S = 0 \quad \forall v_h \in V_h \quad (2.10)$$

This formulation has only integrals over edges of the mesh, which is another advantage of a Trefftz method, that Integrals over the elements drop out. It can conveniently be calculated by iterating over all edges of the mesh.

2.4 Numerical Flux

For the numerical flux we introduce the same notation as in [4] to get unique values on edges, which belong to two elements. The subscripts $_L$ and $_R$ refer to the left and right hand side element of the edge.

$$\begin{aligned} \text{average} \quad & \{\{u\}\} = \frac{1}{2}(u_L + u_R), \quad \{\{\vec{\gamma}\}\} = \frac{1}{2}(\vec{\gamma}_L + \vec{\gamma}_R) \\ \text{jump} \quad & [[u]] = u_L \vec{n}_L + u_R \vec{n}_R, \quad [[\vec{\gamma}]] = \vec{\gamma}_L \cdot \vec{n}_L + \vec{\gamma}_R \cdot \vec{n}_R \end{aligned} \quad (2.11)$$

Let $\mathcal{F}_h$ be the edges of partition $\mathcal{T}_h$, and $\mathcal{F}_h^{\mathcal{I}} = \mathcal{F}_h \backslash \partial\Omega$ the set of interior edges. Analogously, $\mathcal{F}_h^{\mathcal{B}} = \mathcal{F}_h \cap \partial\Omega$ is the set of boundary edges. The numerical fluxes are chosen the same way as in [6]. For interior edges $\mathcal{F}_h^{\mathcal{I}}$ we choose

$$\begin{aligned} \hat{u}_h &= \{\{u_h\}\} + \vec{\zeta} \cdot [[u_h]]_N - \frac{\beta}{\sqrt{\sigma_K}} \left[\left[\vec{\nabla}_h u_h\right]\right]_N \\ \hat{\vec{\gamma}}_h &= \left\{\left\{\vec{\nabla}_h u_h\right\}\right\} - \alpha\sqrt{\sigma_K} [[u_h]]_N - \vec{\zeta} \left[\left[\vec{\nabla}_h u_h\right]\right]_N \end{aligned} \quad (2.12)$$

and on boundary edges $\mathcal{F}_h^{\mathcal{B}}$ we choose

$$\begin{aligned} \hat{u}_h &= u_h - \frac{\delta}{z}\left(\vec{\nabla}_h u_h \cdot \vec{n} + z u_h - g\right) \\ \vec{\hat{\gamma}}_h &= \vec{\nabla}_h u_h - (1-\delta)\left(\vec{\nabla}_h u_h + z u_h \vec{n} - g\vec{n}\right) \end{aligned} \tag{2.13}$$

The parameter $\vec{\zeta}$ is chosen to be zero, like in [6]. This reduces the complexity of finding appropriate values for the remaining parameters.

For δ, the natural choice is zero, as it corresponds to omitting the introduction of a numeric flux on the boundary terms. Instead, the boundary condition is inserted into the equations (2.4) and the numeric flux is introduced only for interior terms. This leads to the same equations as choosing $\delta = 0$ for the numeric flux on the boundary:

$$\begin{aligned} \hat{u}_h &= u_h \\ \vec{\hat{\gamma}}_h &= -z u_h \vec{n} + g\vec{n} \end{aligned} \tag{2.14}$$

Note that $\delta = 0$ is needed to show ellipticity in section 2.4.2. We will discuss the parameters α and β in section 3.5. Next we discuss the properties consistency and ellipticity.

2.4.1 Consistency

The method is called consistent if, given a smooth solution u which solves the given problem (1.1), u also solves the variational formulation (2.10). For smooth functions $[[\cdot]] = 0$ and $\{\{u\}\} = u$. For boundary edges the term $\vec{\nabla}_h u_h \cdot \vec{n} + z u_h - g = 0$ because u solves $\vec{\nabla}_h u_h \cdot \vec{n} + z u_h = g$ on the boundary. What remains is $\hat{u}_h = u$ and $\vec{\hat{\gamma}}_h = \vec{\nabla} u$ for all edges. Thus it can be seen that this is consistent by considering the derivation in section 2.2 without introducing the discretisation. Then the numerical fluxes correspond to the function u as obtained above.

2.4.2 Ellipticity

For the ellipticity, we need the bilinear form. We derive the bilinear form from equation (2.8) because there we didn't use that our basis functions solve $-\Delta u + \sigma u = 0$ and makes it therefore more general. First we insert the numeric fluxes (2.12) and (2.13) into the

equation:

$$\sum_K \Bigg[\int_K \left(\sigma_K u_h v_h + \vec{\nabla}_h u_h \cdot \vec{\nabla}_h v_h \right) \mathrm{d}S \\ + \int_{\partial K \setminus \partial\Omega} \left(\{\{u_h\}\} + \frac{\beta}{\sqrt{\sigma_K}} \left[\left[\vec{\nabla}_h u_h \right] \right]_N - u_h \right) \vec{\nabla}_h v_h \cdot \vec{n} \mathrm{d}S \\ + \int_{\partial K \cap \partial\Omega} \left(-\frac{\delta}{z} \left[\vec{\nabla}_h u_h \cdot \vec{n} + u_h - g \right] \right) \vec{\nabla}_h v_h \cdot \vec{n} \mathrm{d}S \\ + \int_{\partial K \setminus \partial\Omega} \left(- \left\{ \left\{ \vec{\nabla}_h u_h \right\} \right\} + \alpha \sqrt{\sigma_K} \left[[u_h] \right]_N \right) \cdot \vec{n} v_h \mathrm{d}S \\ + \int_{\partial K \cap \partial\Omega} \left(-\vec{\nabla}_h u_h + (1-\delta) \left(\vec{\nabla}_h u_h - z u_h \vec{n} - g\vec{n} \right) \right) \cdot \vec{n} v_h \mathrm{d}S \Bigg] = 0 \qquad (2.15)$$

In a next step, we want to write the integrals over ∂K in terms of edges $\mathcal{F}_h^{\mathcal{I}}$ and $\mathcal{F}_h^{\mathcal{B}}$ of the mesh. Each interior edge is shared by two triangles. The jump $[[\cdot]]$ and average $\{\{\cdot\}\}$ operators are independent of which triangle we look at. The remaining expressions are of the form $w\vec{n}$ and can be written as a jump term when translating from the sum notation to the integration over $\mathcal{F}_h^{\mathcal{I}}$. The integral $\int_{\partial K \setminus \partial\Omega} u_h \vec{\nabla}_h v_h \cdot \vec{n} \, \mathrm{d}S$ can be written as $\int_{\mathcal{F}_h^{\mathcal{I}}} \{\{u_h\}\} \left[\left[\vec{\nabla}_h v_h \right] \right] \mathrm{d}S + \int_{\mathcal{F}_h^{\mathcal{I}}} [[u_h]] \cdot \left\{ \left\{ \vec{\nabla}_h v_h \right\} \right\} \mathrm{d}S$

$$\sum_K \left[\int_K \left(\sigma_K u_h v_h + \vec{\nabla}_h u_h \cdot \vec{\nabla}_h v_h \right) \mathrm{d}S \right] \\ + \int_{\mathcal{F}_h^{\mathcal{I}}} - [[u_h]]_N \cdot \left\{ \left\{ \vec{\nabla}_h v_h \right\} \right\} + \frac{\beta}{\sqrt{\sigma_K}} \left[\left[\vec{\nabla}_h u_h \right] \right]_N \left[\left[\vec{\nabla}_h v_h \right] \right]_N \mathrm{d}S \\ + \int_{\mathcal{F}_h^{\mathcal{I}}} - \left\{ \left\{ \vec{\nabla}_h u_h \right\} \right\} \cdot [[v_h]]_N + \alpha \sqrt{\sigma_K} [[u_h]]_N \cdot [[v_h]]_N \, \mathrm{d}S \\ + \int_{\mathcal{F}_h^{\mathcal{B}}} \left(-\frac{\delta}{z} \left[\vec{\nabla}_h u_h \cdot \vec{n} + z u_h - g \right] \right) \vec{\nabla}_h v_h \cdot \vec{n} \mathrm{d}S \\ + \int_{\mathcal{F}_h^{\mathcal{B}}} \left(-\vec{\nabla}_h u_h + (1-\delta) \left(\vec{\nabla}_h u_h + z u_h \vec{n} - g\vec{n} \right) \right) \cdot \vec{n} v_h \mathrm{d}S = 0 \qquad (2.16)$$

For the usual $a(u,v) = f(v)$ we have the bilinear form

$$\begin{aligned} a(u,v) &= \sum_K \left[\int_K \left(\sigma_K u_h v_h + \vec{\nabla}_h u_h \cdot \vec{\nabla}_h v_h \right) \mathrm{d}S \right] \\ &+ \int_{\mathcal{F}_h^{\mathcal{I}}} - [[u_h]]_N \cdot \left\{ \left\{ \vec{\nabla}_h v_h \right\} \right\} + \frac{\beta}{\sqrt{\sigma_K}} \left[\left[\vec{\nabla}_h u_h \right] \right]_N \left[\left[\vec{\nabla}_h v_h \right] \right]_N \mathrm{d}S \\ &+ \int_{\mathcal{F}_h^{\mathcal{I}}} - \left\{ \left\{ \vec{\nabla}_h u_h \right\} \right\} \cdot [[v_h]]_N + \alpha \sqrt{\sigma_K} \, [[u_h]]_N \cdot [[v_h]]_N \, \mathrm{d}S \\ &+ \int_{\mathcal{F}_h^{\mathcal{B}}} - \frac{\delta}{z} \vec{\nabla}_h u_h \cdot \vec{n} \, \vec{\nabla}_h v_h \cdot \vec{n} \, - \delta u_h \vec{\nabla}_h v_h \cdot \vec{n} \, \mathrm{d}S \\ &+ \int_{\mathcal{F}_h^{\mathcal{B}}} - \delta \vec{\nabla}_h u_h \cdot \vec{n} \, v_h + (1-\delta) z u_h v_h \mathrm{d}S \end{aligned} \tag{2.17}$$

and the linear form

$$f(v) = \int_{\mathcal{F}_h^{\mathcal{B}}} - \frac{\delta}{z} g \vec{\nabla}_h v_h \cdot \vec{n} \, + (1-\delta) \, g v_h \mathrm{d}S \tag{2.18}$$

The bilinear form is symmetric. Ellipticity is defined as: There exists a constant $C > 0$, such that

$$a(v,v) \geq C \, ||v||_{\mathrm{DG}}^2 \, \forall v \in V$$

A problem arises from the term $-\frac{\delta}{z} \vec{\nabla}_h u_h \cdot \vec{n} \, \vec{\nabla}_h v_h \cdot \vec{n}$ of the boundary integrals, which will always be negative for $a(v,v)$. The problem can be omitted by choosing $\delta = 0$, which was already observed to be the natural choice in section 2.4. Now ellipticity can be shown as in [6], proposition 4.2. From this follow certain restrictions for the parameters:

$$\alpha > \alpha_{\min} > \frac{C_{\mathrm{tinv}}^2}{\sqrt{\sigma} h}, \qquad \beta > 0, \qquad \delta = 0 \tag{2.19}$$

Where C_{tinv} is a trace inverse estimate, a constant such that $||v||_{0,\partial K} \leq C_{\mathrm{tinv}} h_K^{-1/2} \, ||v||_{0,K}$

3 Implementation

3.1 Mesh

The implementation was made in the LehrFEM framework, a finite element framework implemented in MATLAB. One essential component is the mesh, which is stored as a set of vectors and matrices, bundled in a MATLAB struct. The vertices are stored row wise in the $m \times 2$ matrix `Coordinates`. One element is made up of three integers which reference a row in the vertex matrix. The elements are also stored row wise in the $n \times 3$ matrix `Elements`. σ is constant on an element, for that reason a sigma value is stored for each element in the $n \times 1$ vector `ElemSigma`.

There are also some auxiliary matrices which help assembling the equations, like the `Edge` matrix. It is a $o \times 2$ matrix which references row wise the two vertices of an edge, like the element matrix. The vector `BdFlags` is a $o \times 1$ vector. Its entries are negative for edges which belong to the boundary of the domain. The $o \times 2$ `Edge2Elem` matrix stores the two elements which share the respective edge.

3.2 Exponential Basis Functions

We want solutions of the differential equation

$$\Delta u(x, y) - \sigma_k u(x, y) = 0 \tag{3.1}$$

as basis functions for the DGfinite element method. As the laplacian is invariant under rotation, we can add more basis functions by choosing different angles with which the basis functions are rotated. Therefore we solve the 1D differential equation

$$\frac{\partial^2 u}{\partial x^2}(x) - \sigma_k \cdot u(x) = 0 \tag{3.2}$$

A solution of this differential equation is

$$u(x) = Ae^{\sqrt{\sigma_k} \cdot x} + Be^{-\sqrt{\sigma_k} \cdot x} \tag{3.3}$$

which is also a solution to the 2D equation. The second term $Be^{-\sqrt{\sigma_k} \cdot x}$ can be omitted, as it is equivalent to $Ae^{\sqrt{\sigma_k} \cdot x}$ in the opposite direction. Then we introduce the rotation of the basis function as $x \cos(\alpha) + y \sin(\alpha)$. The basis functions should be tied to a point of the element. For convenience, we choose this point to be the first vertex of the element. The final basis function, with one coefficient, direction α and vertex (x_0, y_0) is:

$$A \exp\left[\sqrt{\sigma_k} \begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix} \cdot \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix}\right] \tag{3.4}$$

3.3 Bessel Basis Functions

The exponential basis functions have a disadvantage: If the angles between the basis functions get smaller, the functions get linearly more dependent. Thus if one uses a lot of basis functions, the stiffness matrix gets very ill conditioned. To mitigate this problem, we introduce another set of basis functions, which are based on the modified Bessel functions. We start again with the differential equation (3.1) which can be written in polar coordinates:

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \phi^2} - \sigma_k u = 0 \tag{3.5}$$

Now we choose $u(r,\phi) = R(r)\mathsf{trig}(k\phi)$, where $\mathsf{trig}(k\phi)$ stands for either $\sin(k\phi)$ or $\cos(k\phi)$. And we multiply the equation by $r^2/\mathsf{trig}(k\phi)$.

$$r^2\frac{\partial^2 R(r)}{\partial r^2} + r\frac{\partial R(r)}{\partial r} - k^2 R(r) - \sigma_k r^2 R(r) = 0 \tag{3.6}$$

Now the partial differential equation is only in r. We want to write it in form of the modified Bessel's differential equation and introduce for that the variable transformation: $\tilde{r} = r \cdot \sqrt{\sigma}$, $\tilde{R}(r\sqrt{\sigma}) = R(r)$. The second derivative of $\tilde{R}$ is then

$$\frac{\partial^2 \tilde{R}(\tilde{r})}{\partial r^2} = \frac{\partial^2 \tilde{R}(\tilde{r})}{\partial \tilde{r}^2}\underbrace{\left(\frac{\partial \tilde{r}}{\partial r}\right)^2}_{=\sigma} + \frac{\partial \tilde{R}(\tilde{r})}{\partial \tilde{r}}\underbrace{\frac{\partial^2 \tilde{r}}{\partial r^2}}_{=0} = \frac{\partial^2 \tilde{R}(\tilde{r})}{\partial \tilde{r}^2}\sigma \tag{3.7}$$

Transforming now equation (3.6) yields:

$$\tilde{r}^2\frac{\partial^2 \tilde{R}(\tilde{r})}{\partial \tilde{r}^2} + \tilde{r}\frac{\partial \tilde{R}(\tilde{r})}{\partial \tilde{r}} - (\tilde{r}^2 + k^2)\tilde{R}(\tilde{r}) = 0 \tag{3.8}$$

The solutions of this differential equation are the modified Bessel function of the first kind, and the modified Bessel function of the second kind. The modified Bessel function of the first kind is:

$$I_k(\tilde{r}) = \sum_{m=0}^{\infty} \frac{1}{m!\,\Gamma(m+k+1)}\left(\frac{\tilde{r}}{2}\right)^{2m+k} \tag{3.9}$$

It is finite at $r = 0$. The modified Bessel function of the second kind is not finite at $r = 0$, but a finite solution is needed there. Therefore only the modified Bessel function of the first kind is used. Also we restrict k to $k \in \mathbb{N}_0$ to achieve an independent basis.

$$u(r,\phi) = I_k(\sqrt{\sigma}r)\mathsf{trig}(k\phi) \tag{3.10}$$

The derivative of this function is also needed for calculation. It can be derived with the identity $I'_\nu(z) = \frac{1}{2}(I_{\nu-1}(z) + I_{\nu+1}(z))$

$$\vec{\nabla} r = \frac{1}{\sqrt{x^2+y^2}}\begin{pmatrix} x \\ y \end{pmatrix}, \qquad \vec{\nabla}\phi = \frac{1}{y^2+x^2}\begin{pmatrix} -y \\ x \end{pmatrix},$$

$$k\mathsf{trig}'(k\phi) = \begin{cases} k\sin'(k\phi) = k\cos(k\phi) \\ k\cos'(k\phi) = (-k)\sin(k\phi) \end{cases} \tag{3.11}$$

$$\begin{aligned} \vec{\nabla}u &= (\vec{\nabla}I_k(\sqrt{\sigma}r))\mathsf{trig}(k\phi) + I_k(\sqrt{\sigma}r)k\mathsf{trig}'(k\phi)(\vec{\nabla}\phi) && (3.12) \\ &= \frac{1}{2}\left[I_{k-1}(\sqrt{\sigma}r) + I_{k+1}(\sqrt{\sigma}r)\right]\frac{\sqrt{\sigma}}{\sqrt{x^2+y^2}}\begin{pmatrix} x \\ y \end{pmatrix}\mathsf{trig}(k\phi) \\ &\quad + I_k(\sqrt{\sigma}r)k\mathsf{trig}'(k\phi)\frac{1}{y^2+x^2}\begin{pmatrix} -y \\ x \end{pmatrix} && (3.13) \end{aligned}$$

There is no easy simplification to solve the integrals analytically like for the exponential basis functions. Therefore the integrals are solved numerically for the matrix assembly.

3.4 Matrix Assembly

We start with the variational formulation (2.10). For the purpose of calculation, we split those integrals up into integrals over edges and sum it up afterwards.

3.4.1 Interior Contributions

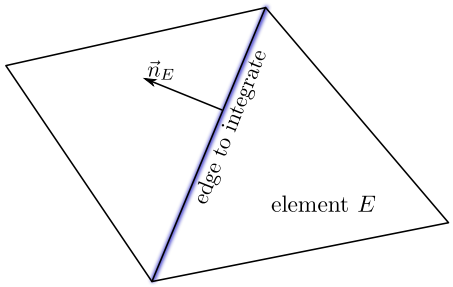
On interior edges assemble

$$\begin{aligned} &\int_{\partial K}\left(\{\{u_h\}\} + \vec{\zeta}\cdot[[u_h]]_N - \frac{\beta}{\sqrt{\sigma_K}}\left[\left[\vec{\nabla}_h u_h\right]\right]_N\right)\vec{\nabla}v_h\cdot\vec{n}\mathrm{d}S \\ -&\int_{\partial K}\left(\left\{\left\{\vec{\nabla}_h u_h\right\}\right\} - \alpha\sqrt{\sigma_K}\,[[u_h]]_N - \vec{\zeta}\left[\left[\vec{\nabla}_h u_h\right]\right]_N\right)\cdot\vec{n}v_h\mathrm{d}S \quad \text{for } \partial K \notin \partial\Omega \end{aligned} \tag{3.14}$$

Let's write v_L^i, v_R^i for the basis function i on the right and the left side of the edge. All basis functions that are not part of the left or the right element will not contribute to the integrals on this edge. u can also be written in terms of the basis functions. For l and r basis functions on the left and right triangle respectively:

$$u_h = \sum_{i=1}^{l}\mu_L^i v_L^i + \sum_{i=1}^{r}\mu_R^i v_R^i$$

Additionally we define v_E^i as one of v_L^i and v_R^i, and the same thing for the normal $\vec{n}_E$ which is pointing out of the left or right element respectively:



The interior term corresponding to test function v_E^i is then:

$$\begin{aligned}
& \int_{\partial K} \left\{\left\{\sum_{i=1}^{l} \mu_L^i v_L^i + \sum_{i=1}^{r} \mu_R^i v_R^i\right\}\right\} \vec{\nabla} v_E^i \cdot \vec{n}_E \mathrm{d}S \\
+ \quad & \int_{\partial K} \vec{\zeta} \cdot \left[\left[\sum_{i=1}^{l} \mu_L^i v_L^i + \sum_{i=1}^{r} \mu_R^i v_R^i\right]\right]_N \vec{\nabla} v_E^i \cdot \vec{n}_E \mathrm{d}S \\
- \quad & \int_{\partial K} \frac{\beta}{\sqrt{\sigma_E}} \left[\left[\sum_{i=1}^{l} \mu_L^i \vec{\nabla} v_L^i + \sum_{i=1}^{r} \mu_R^i \vec{\nabla} v_R^i\right]\right]_N \vec{\nabla} v_E^i \cdot \vec{n}_E \mathrm{d}S \\
- \quad & \int_{\partial K} \left\{\left\{\sum_{i=1}^{l} \mu_L^i \vec{\nabla} v_L^i + \sum_{i=1}^{r} \mu_R^i \vec{\nabla} v_R^i\right\}\right\} \cdot \vec{n}_E v_E^i \mathrm{d}S \\
+ \quad & \int_{\partial K} \alpha\sqrt{\sigma_E} \left[\left[\sum_{i=1}^{l} \mu_L^i v_L^i + \sum_{i=1}^{r} \mu_R^i v_R^i\right]\right]_N \cdot \vec{n}_E v_E^i \mathrm{d}S \\
+ \quad & \int_{\partial K} \vec{\zeta} \left[\left[\sum_{i=1}^{l} \mu_L^i \vec{\nabla} v_L^i + \sum_{i=1}^{r} \mu_R^i \vec{\nabla} v_R^i\right]\right]_N \cdot \vec{n}_E v_E^i \mathrm{d}S
\end{aligned} \tag{3.15}$$

In the next step the jump and average terms are expanded.

$$\begin{aligned}
& \int_{\partial K} \frac{1}{2} \left(\sum_{i=1}^{l} \mu_L^i v_L^i + \sum_{i=1}^{r} \mu_R^i v_R^i\right) \vec{\nabla} v_E^i \cdot \vec{n}_E \mathrm{d}S \\
+ \quad & \int_{\partial K} \vec{\zeta} \cdot \left(\sum_{i=1}^{l} \mu_L^i v_L^i \vec{n}_L + \sum_{i=1}^{r} \mu_R^i v_R^i \vec{n}_R\right) \vec{\nabla} v_E^i \cdot \vec{n}_E \mathrm{d}S \\
- \quad & \int_{\partial K} \frac{\beta}{\sqrt{\sigma_E}} \left(\sum_{i=1}^{l} \mu_L^i \vec{\nabla} v_L^i \cdot \vec{n}_L + \sum_{i=1}^{r} \mu_R^i \vec{\nabla} v_R^i \cdot \vec{n}_R\right) \vec{\nabla} v_E^i \cdot \vec{n}_E \mathrm{d}S \\
- \quad & \int_{\partial K} \frac{1}{2} \left(\sum_{i=1}^{l} \mu_L^i \vec{\nabla} v_L^i + \sum_{i=1}^{r} \mu_R^i \vec{\nabla} v_R^i\right) \cdot \vec{n}_E v_E^i \mathrm{d}S \\
+ \quad & \int_{\partial K} \alpha\sqrt{\sigma_E} \left(\sum_{i=1}^{l} \mu_L^i v_L^i \vec{n}_L + \sum_{i=1}^{r} \mu_R^i v_R^i \vec{n}_R\right) \cdot \vec{n}_E v_E^i \mathrm{d}S \\
+ \quad & \int_{\partial K} \vec{\zeta} \left(\sum_{i=1}^{l} \mu_L^i \vec{\nabla} v_L^i \cdot \vec{n}_L + \sum_{i=1}^{r} \mu_R^i \vec{\nabla} v_R^i \cdot \vec{n}_R\right) \cdot \vec{n}_E v_E^i \mathrm{d}S
\end{aligned} \tag{3.16}$$

One matrix entry, representing equation for test function v_E^i, and solution coefficient μ_D^m, then looks like

$$\begin{aligned}
\mathbf{A}_{(Ei,Dm)} = \int_{\partial K} & \frac{1}{2} v_D^m \vec{\nabla} v_E^i \cdot \vec{n}_E + \vec{\zeta} \cdot v_D^m \vec{n}_D \vec{\nabla} v_E^i \cdot \vec{n}_E - \frac{\beta}{\sqrt{\sigma_E}} \vec{\nabla} v_D^m \cdot \vec{n}_D \vec{\nabla} v_E^i \cdot \vec{n}_E \\
& - \frac{1}{2} \vec{\nabla} v_D^m \cdot \vec{n}_E v_E^i + \alpha\sqrt{\sigma_E} v_D^m \vec{n}_D \cdot \vec{n}_E v_E^i + \vec{\zeta} (\vec{\nabla} v_D^m \cdot \vec{n}_D) \cdot \vec{n}_E v_E^i \mathrm{d}S
\end{aligned} \tag{3.17}$$

3.4.2 Boundary Contributions

On the boundary $\partial\Omega$ assemble

$$\int_{\partial K}\left(u_h-\frac{\delta}{z}\left(\vec{\nabla}_h u_h\cdot\vec{n}+u_h-g\right)\right)\vec{\nabla}v_h\cdot\vec{n}\mathrm{d}S$$
$$-\int_{\partial K}\left(\vec{\nabla}_h u_h-(1-\delta)\left(\vec{\nabla}_h u_h+u_h\vec{n}-g\vec{n}\right)\right)\cdot\vec{n}v_h\mathrm{d}S\quad\text{for}\quad\partial K\in\partial\Omega \tag{3.18}$$

E and D can be dropped from the notation because only one element is involved. Apart from that the same notation as before is used. One element of the Matrix is:

$$\begin{aligned}\mathbf{A}_{(i,m)} &= \int_{\partial K}(1-\frac{\delta}{z})v^m\vec{\nabla}v^i\cdot\vec{n}-\frac{\delta}{z}\vec{\nabla}v^m\cdot\vec{n}\vec{\nabla}v^i\cdot\vec{n}\\ &\quad-\delta\vec{\nabla}v^m\cdot\vec{n}v^i+(1-\delta)v^m v^i\mathrm{d}S\end{aligned} \tag{3.19}$$

and one element of the right hand side is:

$$L_{(i)} = -\int_{\partial K}\frac{\delta}{z}g\vec{\nabla}v^i\cdot\vec{n}+(1-\delta)gv^i\mathrm{d}S \tag{3.20}$$

3.4.3 Matrix Assembly for Exponential Basis Functions

For exponential basis functions, it is possible to solve the integrals analytically. In order to do so we replace v with 3.4 in the expressions for the matrix elements:

$$v_E^i=\underbrace{\exp\left[-\sqrt{\sigma_E}\vec{x}_0^E\cdot\begin{pmatrix}\cos(\alpha_i)\\ \sin(\alpha_i)\end{pmatrix}\right]}_{c_E^i}\exp\left[\sqrt{\sigma_E}\vec{x}\cdot\begin{pmatrix}\cos(\alpha_i)\\ \sin(\alpha_i)\end{pmatrix}\right]$$

Where $\vec{x}=\begin{pmatrix}x\\ y\end{pmatrix}$ and $\vec{x}_0^E=\begin{pmatrix}x_0^E\\ y_0^E\end{pmatrix}$, which is the first Vertex of the Triangle E. The gradient is:

$$\vec{\nabla}v_E^i=c_E^i\exp\left[\sqrt{\sigma_E}\vec{x}\cdot\begin{pmatrix}\cos(\alpha_i)\\ \sin(\alpha_i)\end{pmatrix}\right]\cdot\sqrt{\sigma_E}\begin{pmatrix}\cos(\alpha_i)\\ \sin(\alpha_i)\end{pmatrix}$$

As the formulas get quite big in that notation, the following abbreviations are introduced:

$$\exp\left[\sqrt{\sigma_E}\vec{x}\cdot\begin{pmatrix}\cos(\alpha_i)\\ \sin(\alpha_i)\end{pmatrix}\right]=w_E^i(x,y);\qquad\vec{\theta}_E^i=\sqrt{\sigma_E}\begin{pmatrix}\cos(\alpha_i)\\ \sin(\alpha_i)\end{pmatrix}$$

Applied to v:

$$v_E^i=c_E^i w_E^i(x,y);\qquad\vec{\nabla}v_E^i=c_E^i\vec{\theta}_E^i w_E^i(x,y)$$

Integration

The integrals are of the form $\int w_D^m \cdot w_E^i \mathrm{d}S$ for all terms. The solution to this integral can be derived by simplifying the two exponentials to one, which then can be integrated easily.

$$\begin{aligned}\int_{\partial K} w_D^m(x,y) w_E^i(x,y) \mathrm{d}S &= \int_{\partial K} \exp\left[\vec{x}\cdot\vec{\theta}_D^m\right] \exp\left[\vec{x}\cdot\vec{\theta}_E^i\right] \mathrm{d}S \\ &= \int_{\partial K} \exp\left[\vec{x}\cdot\left(\vec{\theta}_D^m + \vec{\theta}_E^i\right)\right] \mathrm{d}S \end{aligned} \tag{3.21}$$

In order to solve the integral, it is mapped onto the edge: $\vec{\lambda}(t) = \vec{b} + \vec{d}\cdot t$, $t \in [0;1]$ where $\vec{d}$ represents the vector from the start $\vec{b}$ to the end point of the edge.

$$\begin{aligned} &= \int_0^1 \exp\left[\vec{\lambda}(t)\cdot\left(\vec{\theta}_D^m + \vec{\theta}_E^i\right)\right] \left|\left|\frac{\partial\vec{\lambda}(t)}{\partial t}\right|\right| \mathrm{d}t \\ &= \exp\left[\vec{b}\cdot\left(\vec{\theta}_D^m + \vec{\theta}_E^i\right)\right] \int_0^1 \exp\left[\vec{d}\cdot\left(\vec{\theta}_D^m + \vec{\theta}_E^i\right)t\right] \left|\left|\vec{d}\right|\right| \mathrm{d}t \\ &= \frac{\exp\left[\vec{b}\cdot\left(\vec{\theta}_D^m + \vec{\theta}_E^i\right)\right] \left|\left|\vec{d}\right|\right|}{\vec{d}\cdot\left(\vec{\theta}_D^m + \vec{\theta}_E^i\right)} \left\{\exp\left[\vec{d}\cdot\left(\vec{\theta}_D^m + \vec{\theta}_E^i\right)\right] - 1\right\} \end{aligned} \tag{3.22}$$

A problem arises if the exponent in 3.22 goes to zero, namely $\vec{\theta}_D^m + \vec{\theta}_E^i$ goes to zero when the angles are π apart. In this case the numerical evaluation gets unstable. In order to avoid this instability, in case $\delta = \vec{d}\cdot\left(\vec{\theta}_D^m + \vec{\theta}_E^i\right) < \mathsf{tol}$ we use a Taylor expansion around zero for the integral $\int_0^1 e^{\delta t}\mathrm{d}t$:

$$1 + \frac{1}{1!}\frac{1}{2}\delta + \frac{1}{2!}\frac{1}{3}\delta^2 + \frac{1}{3!}\frac{1}{4}\delta^3 + \cdots \tag{3.23}$$

With the integral solved analytically we are ready to assemble the matrix elements. First the interior contributions are calculated. Equation 3.17 written in terms of w_E^i:

$$\begin{aligned} \mathbf{A}_{(Ei,Dm)} &= \frac{1}{2} c_D^m c_E^i \vec{\theta}_E^i \cdot \vec{n}_E \int_{\partial K} w_D^m(x,y) w_E^i(x,y) \mathrm{d}S \\ &+ c_D^m c_E^i \vec{\zeta}\cdot\vec{n}_D \vec{\theta}_E^i \cdot \vec{n}_E \int_{\partial K} w_D^m(x,y) w_E^i(x,y) \mathrm{d}S \\ &- \frac{\beta}{\sqrt{\sigma_E}} c_D^m c_E^i \vec{\theta}_D^m \cdot \vec{n}_D \vec{\theta}_E^i \cdot \vec{n}_E \int_{\partial K} w_D^m(x,y) w_E^i(x,y) \mathrm{d}S \\ &- \frac{1}{2} c_D^m c_E^i \vec{\theta}_D^m \cdot \vec{n}_E \int_{\partial K} w_D^m(x,y) w_E^i(x,y) \mathrm{d}S \\ &+ \alpha\sqrt{\sigma_E} c_D^m c_E^i \vec{n}_D \cdot \vec{n}_E \int_{\partial K} w_D^m(x,y) w_E^i(x,y) \mathrm{d}S \\ &+ \vec{\zeta} c_D^m c_E^i (\vec{\theta}_D^m \cdot \vec{n}_D) \cdot \vec{n}_E \int_{\partial K} w_D^m(x,y) w_E^i(x,y) \mathrm{d}S \end{aligned} \tag{3.24}$$

The Integral $\int_{\partial K} w_D^m(x,y) w_E^i(x,y) \mathrm{d}S$ and the constant $c_D^m c_E^i$ is the same for each term, therefore we calculate it only once. Written in form of a matrix:

$$\mathbf{int}_{(Ei,Dm)} = c_D^m c_E^i \int_{\partial K} w_D^m(x,y) w_E^i(x,y) \mathrm{d}S \tag{3.25}$$

Then each of the six terms is set up as a matrix:

$$\begin{aligned}
&\text{Term 1} && \mathbf{T1}_{(Ei,Dm)} = \frac{1}{2}\vec{\theta^i_E} \cdot \vec{n}_E \\
&\text{Term 2} && \mathbf{T2}_{(Ei,Dm)} = \vec{\zeta} \cdot \vec{n}_D \vec{\theta^i_E} \cdot \vec{n}_E \\
&\text{Term 3} && \mathbf{T3}_{(Ei,Dm)} = \frac{\beta}{\sqrt{\sigma_E}} \vec{\theta^m_D} \cdot \vec{n}_D \vec{\theta^i_E} \cdot \vec{n}_E \\
&\text{Term 4} && \mathbf{T4}_{(Ei,Dm)} = \frac{1}{2}\vec{\theta^m_D} \cdot \vec{n}_E \\
&\text{Term 5} && \mathbf{T5}_{(Ei,Dm)} = \alpha\sqrt{\sigma_E} \vec{n}_D \cdot \vec{n}_E \\
&\text{Term 6} && \mathbf{T6}_{(Ei,Dm)} = \vec{\zeta} \cdot \vec{n}_E \vec{\theta^m_D} \cdot \vec{n}_D
\end{aligned} \tag{3.26}$$

Assembly then looks as follows:

$$A_{(Ei,Dm)} = (\mathbf{T1} + \mathbf{T2} - \mathbf{T3} - \mathbf{T4} + \mathbf{T5} + \mathbf{T6})_{(Ei,Dm)} \cdot \mathbf{int}_{(Ei,Dm)} \tag{3.27}$$

For the boundary contributions, the integration for the right hand side has to be carried out numerically, to keep it as general as possible for boundary data g. The numerical integration is straight forward from (3.20). For the left hand side we write again the integrals of equation 3.19 in terms of w_i:

$$\begin{aligned}
A_{(i,m)} &= (1 - \frac{\delta}{z})\vec{\theta^i} \cdot \vec{n} \int_{\partial K} v^m v^i \mathrm{d}S \\
&- \frac{\delta}{z}\vec{\theta^m} \cdot \vec{n}\vec{\theta^i} \cdot \vec{n} \int_{\partial K} v^m v^i \mathrm{d}S \\
&- \delta\vec{\theta^m} \cdot \vec{n} \int_{\partial K} v^m v^i \mathrm{d}S \\
&+ (1 - \delta) \int_{\partial K} v^m v^i \mathrm{d}S
\end{aligned} \tag{3.28}$$

The matrix $\mathbf{int}_{(i,m)}$ is defined analogously to eq. 3.25. Here we only have four terms which are again set up as a matrix.

$$\begin{aligned}
&\text{Term 1} && \mathbf{T1}_{(i,m)} = (1 - \frac{\delta}{z})\vec{\theta^i} \cdot \vec{n} \\
&\text{Term 2} && \mathbf{T2}_{(i,m)} = \frac{\delta}{z}\vec{\theta^m} \cdot \vec{n}\vec{\theta^i} \cdot \vec{n} \\
&\text{Term 3} && \mathbf{T3}_{(i,m)} = \delta\vec{\theta^m} \cdot \vec{n} \\
&\text{Term 4} && \mathbf{T4}_{(i,m)} = 1 - \delta
\end{aligned} \tag{3.29}$$

Assembly then looks as follows:

$$A_{(i,m)} = (\mathbf{T1} - \mathbf{T2} - \mathbf{T3} + \mathbf{T4})_{(i,m)} \cdot \mathbf{int}_{(i,m)} \tag{3.30}$$

3.5 Parameters for the Numeric Fluxes

The numeric flux has four parameters, α, β, $\vec{\zeta}$ and δ, which influence the quality of the numerical solution. The vectorial parameter $\vec{\zeta}$ is always chosen to be zero, like in [6]. The other three parameters α, β and δ are restricted by the proof of ellipticity, see eq. 2.19. Note that $\delta = 0$, which only leaves α and β to be chosen.

It is not trivial to find stable parameters which work in all cases. After a lot of failed attempts, the choice being appropriate for all simulations presented here is:

$$\alpha = n^2 \frac{1}{h\sqrt{\sigma}}, \qquad \beta = 0.5 \tag{3.31}$$

where n is the number of basis functions per element. In case σ is zero, all parameters are set to zero. As h has the units of length and $\sqrt{\sigma}$ has the units of one over length, all units drop out of the parameters, thus all of them are chosen units free. Another property of α is the scaling with h which enhances convergence for mesh refinement [6].

Figure 3.1 shows how the error behaves with varying values of α and β. For low values

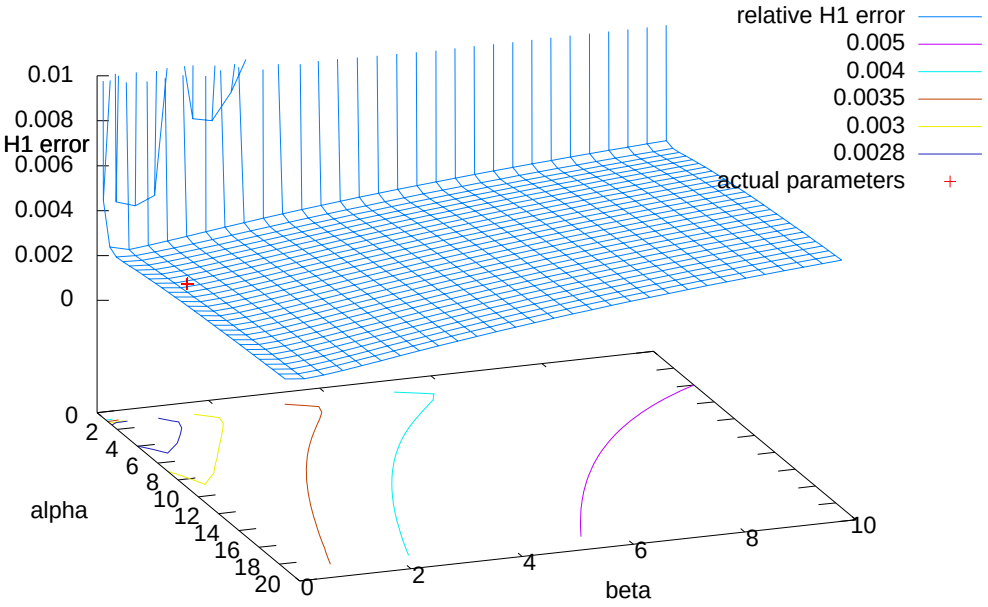


Figure 3.1: H^1 error in the $\alpha \times \beta$ plane. Simulations of a circular domain, with boundary condition $g = 1$, mixed basis functions with number of basis functions $n = 7$, mesh width $h = 0.25$, $z = 1$, $\sigma = 1000$, $\vec{\zeta} = \vec{0}$ and $\delta = 0$

of alpha the error is extremely high. This region should definitely be avoided. But the minimum error is just next to this region, and the error grows from this point on with α. It is quite delicate to find the optimum. Because the error grows only moderately for higher values of α than for lower, it is recommendable to have a too large value of α than a too low one.

On the β axis, the error behaves more smoothly. $\beta = 0$ is already quite close to the minimum. The choice of $\beta = 0.5$ is somehow arbitrary, but it is related to the ultra weak variational formulation [7], as one would have to choose $\beta = 0.5$ to be able to recover it [6]. As the error grows for larger values of β and the minimum is not at zero, it seems to be a reasonable choice.

3.6 Adaptive Choice of Basis Functions

3.6.1 Only Exponential Basis Functions

From section 1.2 the behaviour of the skin layer for smooth boundaries is known. As it is possible to choose the directions of the exponential basis functions per element, we should take advantage of this knowledge. This is straight forward for elements that have a boundary edge. There the direction normal to the boundary, pointing outwards, is most favourable. As the boundary may be curved, or the element is adjacent to a corner, several directions can be used, scattered around the relevant normals.

Listing 3.1 shows an algorithm for choosing a set of exponential basis functions. It does so in four steps, which are illustrated in figure 3.2. The steps are:

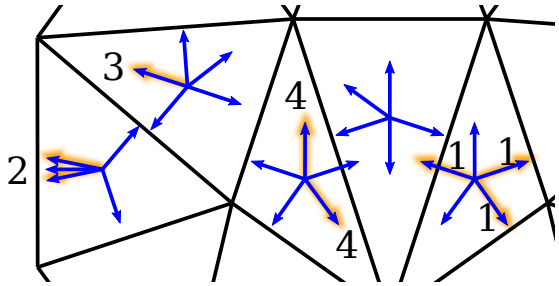


Figure 3.2: Illustration of the algorithm for choosing exponential basis functions adapted to the mesh. Each blue arrow stands for one basis function direction.

1. Line 2: For each edge of the element, choose one basis function in direction of the normal of the edge, pointing outwards.

2. Line 15: For each boundary edge add $2 \cdot$ `n_bd` basis functions to the respective element. Choose the directions symmetrically around the normal of the boundary edge, such that there is always the angle `spreading` in between. The angle should not be too small, or the basis functions will be too similar. It can be scaled with

σ. The higher it is, the closer together the directions may be. `spreading` $= 0.1$ radian was used for the simulations.

3. Line 38: For elements which have just one point on the boundary (not a whole edge), choose an additional basis function with direction of the average normal of the adjoining boundary edges.

4. Line 83: To add further basis functions to the elements, search for the biggest angle between two neighbouring directions. Choose the new direction to be the bisector of those two directions. Repeat this until $\texttt{n}_\texttt{m}\texttt{in}$ is reached.

Listing 3.1: element specific exponential basis functions
file `guess_basis_functions_exp_bdvertex.m`

```
1  % add one direction for each edge to each element
2  for i=1:nElements
3      % set number of basis functions to 3
4      nBfu(i) = 3;
5      % extract vertex coordinates of element i
6      v = Coords(Elements(i,:),:);
7      % angle of the outwards pointing normal of the edge
8      Bfu(i,1) = out_angle(v(2,:) - v(1,:));
9      Bfu(i,2) = out_angle(v(3,:) - v(2,:));
10     Bfu(i,3) = out_angle(v(1,:) - v(3,:));
11 end
12
13 % add n_bd directions for each boundary edge to the
14 % respective element
15 for i = 1:nEdges
16     % if edge belongs to boundary
17     if Mesh.BdFlags(i)<0
18         % get element containing edge i
19         % edgeloc defines where the edge is in the element
20         [elem,edgeloc] = get_elem(Mesh,i);
21         % vertices of edge
22         v_start = Coords(Elements(elem,mod(edgeloc,3)+1),:);
23         v_end = Coords(Elements(elem,mod(edgeloc+1,3)+1),:);
24         % angle of the outwards pointing normal of the edge
25         a = out_angle(v_end - v_start);
26         % add 2*n_bd directions with spaceing spreading
27         for j = 1:n_bd
28             Bfu(elem,3+j*2-1) = a+spreading*j;
29             Bfu(elem,3+j*2) = a-spreading*j;
30         end
31         % update number of basis functions
32         nBfu(elem) = 3+n_bd*2;
33     end
34 end
35
```

```
% add one direction to all elements which have one point
% in common with the boundary
for i = 1:nElements
    for j=1:3
        v = Elements(i,j); % vertex
        if (nBfu(i) == 3 && is_boundary_point(v))
            % find adjoining edges
            edg = Mesh.Edges(:,1) == v;
            edg = edg | Mesh.Edges(:,2) == v;
            % only select boundary edges
            edg = edg & (Mesh.BdFlags < 0);
            % get vertex indices and orientation of found edges
            edg_idx = Mesh.Edges(edg,:);
            edg_orient = sum(Mesh.EdgeOrient(edg,:),2);
            % get angle of normal of edge
            if (edg_orient(1) > 0)
                v_start = Coords(edg_idx(1,1),:);
                v_end = Coords(edg_idx(1,2),:);
            else
                v_start = Coords(edg_idx(1,2),:);
                v_end = Coords(edg_idx(1,1),:);
            end
            a(1) = out_angle(v_end - v_start);
            if (edg_orient(2) > 0)
                v_start = Coords(edg_idx(2,1),:);
                v_end = Coords(edg_idx(2,2),:);
            else
                v_start = Coords(edg_idx(2,2),:);
                v_end = Coords(edg_idx(2,1),:);
            end
            a(2) = out_angle(v_end - v_start);
            % build average direction of both edge normals
            a = sort(a);
            d = a(2) - a(1);
            if (d < pi)
                Bfu(i,nBfu(i)+1) = mod((a(1)+a(2))/2,2*pi);
            else
                Bfu(i,nBfu(i)+1) = mod((a(1)+a(2))/2 + pi,2*pi);
            end
            % update number of basis functions
            nBfu(i) = nBfu(i) + 1;
        end
    end
end

% add directions in the largest angular gap
% until n_min is reached
for i=1:nElements
    if nBfu(i) < n_min
```

```
85          for j = 1:(n_min-nBfu(i))
86              % sort directions to calculate gaps
87              Bfu(i,1:nBfu(i)) = sort(Bfu(i,1:nBfu(i)));
88              gaps = Bfu(i,[2:nBfu(i),1]) - Bfu(i,1:nBfu(i));
89              % modulo takes care of gaps which cross 2*pi boundary
90              % get largest gap
91              [a,idx] = max(mod(gaps,2*pi));
92              % add angle bisector to basis
93              Bfu(i,nBfu(i)+1) = mod(Bfu(i,idx) + a/2,2*pi);
94              nBfu(i) = nBfu(i)+1;
95          end
96      end
97  end
```

The basis functions are stored element wise in the matrix `Bfu`. As not every element has the same number of basis functions, the auxiliary vector `nBfu` stores the number of basis functions per element. The mesh is given by the vertices stored in `Coords` row wise, and the elements in `Elements` which reference the vertex row. The parameters for choosing the basis functions are `n_bd`, which defines how many basis functions are used to spread out in direction of the boundary, and `n_min`, which defines the minimum number of basis functions per element. Step 4 is repeated for each element until `n_min` is reached.

The function `out_angle(e)` calculates the angle of the outwards pointing normal of the edge `e`, which is given as a vector. The triangle must be oriented counter clock wise in order to get the outwards pointing normal. Given an edge index `i` of a boundary edge, `get_elem(Mesh,i)` returns the element index of the corresponding element, and where on this element the edge is located. The `EdgeOrient` matrix stores for each edge, whether it is oriented in the same direction as the element as 1, or if it is in counter direction as -1. Normal edges have two entries for both elements. Boundary edges have only one such entry, the other is zero. The edge orientation is needed on line 51 to get the outwards pointing normal, not the inwards pointing one. Note that MATLAB has a very general implementation of modulo, such that `mod(a,2*pi)` correctly maps the angle `a` to the range $[0, 2\pi)$

`Bfu` and `nBfu` are finally stored in the mesh struct for further processing as `ElemBfuExp` and `ElemNBfuExp` respectively.

This method has one drawback. As the mesh gets finer, the skin depth may drop below an element width. In that case the results get worse due to the missing exponential basis functions in direction of the skin layer. This problem is addressed in the next section, but may be solved in the same way here.

3.6.2 Exponential and Bessel Basis Functions

It is also possible to mix exponential and Bessel basis functions. Exponential basis functions are used where the direction of the skin layer is known. Bessel basis functions are suitable for unknown contributions to the solution, like for interior elements or in vicinity of corners and strongly curved boundaries. For an example see figure 4.19. The

exponential basis functions for elements at the boundary are computed analogously to step 2 in the previous section:

- For each boundary edge, add $2m + 1$ exponential basis functions to the corresponding element. Choose the exponential basis function directions symmetrically around the outwards pointing normal of the boundary edge, such that there is always an angle of about 0.1 radian in between.
- Further exponential basis functions are added to all elements which are less than `layer_width` away from the boundary. Finding the relevant boundary edges for each element is done recursively:

Listing 3.2: finding close boundary edges recursively
file `recursive_boundary_search.m`

```
% initially set the elem_bd_list entry for all bd edges
% on the element they belong to
for i = 1:nEdges
    if Mesh.BdFlags(i)<0
        % find the respective element
        if Mesh.Edge2Elem(i,1) > 0
            elem_bd_list(Mesh.Edge2Elem(i,1),1) = i;
        else
            elem_bd_list(Mesh.Edge2Elem(i,2),1) = i;
        end
    end
end
% recursively find close boundary edges
while true
    % iterate over all elements
    for i = 1:nElem
        % if element does not yet have a elem_bd_list entry
        if sum(elem_bd_list(i,:)) == 0
            % get elem_bd_list entries from neighbours
            bd = get_neighbours_bd(i,Mesh,elem_bd_list);
            % if there are any collected entries
            if ~isempty(bd)
                % store them in an auxiliary list
                aux_elem_bd_list(i,1:length(bd)) = bd;
                % calculate directions
                c = Coords(Elements(i,:),:);
                [temp_dir,dist] = directions(bd, Mesh, c);
                % if there are any directions in layer_width
                if ~isempty(dist) && dist(1) < layer_width
                    n = nBfu(i,1);
                    % number of new basis functions
                    nd = length(temp_dir);
                    % assign basis functions
                    Bfu(i,(n+1):(n+nd)) = temp_dir;
```

```
35                      % update count of basis funtions
36                      nBfu(i,1) = n + nd;
37                  end
38              end
39          end
40      end
41
42      if sum(sum(aux_elem_bd_list)) == 0
43          % if no new entries were generated, exit
44          break
45      else
46          % else store auxiliary list back to elem_bd_list
47          % affected rows
48          r = sum(aux_elem_bd_list,2) > 0;
49          % max size of a row
50          s = size(aux_elem_bd_list,2);
51          % assign new rows to elem_bd_list
52          elem_bd_list(r,1:s) = aux_elem_bd_list(r,:);
53          % clear auxiliary list
54          aux_elem_bd_list = zeros(nElem,10);
55      end
56  end
```

In the first loop (line 3) the matrix of close boundaries `elem_bd_list` is initialized. Each boundary edge gets noted on the row of its respective element index in this matrix, the rest of the matrix is initialized with zeros. In the main loop (line 14) the entries in `elem_bd_list` of neighbouring elements are collected. The function `get_neighbours_bd` finds all elements which have at least one vertex in common with the current element, looks up all boundary edge entries of them and returns that list. See listing 3.3. The found boundary entries are stored in an auxiliary list, which is merged later on line 46 with the normal list. This ensures the selection of boundaries is independent of the element order. `directions` calculates the directions to the closest boundary points. For each boundary supplied in the argument `bd`; it finds the closest point to the element, the distance, and the direction. Then only the directions with the smallest distances (cutoff at 20% of the smallest distance) are considered. Directions which are too close to already included directions are dropped, because they do not improve the basis. The minimum distance to the boundary is compared to the cut off `layer_width` and if it is within the cutoff range, the directions are added to the set of basis functions.

For Bessel basis functions only elements with and without boundary edges are distinguished:

- On interior elements, add $2l + 1$ Bessel functions.
- On Elements which have at least one boundary edge, add only $2(l - m) - 1$ Bessel functions.

Choosing less Bessel basis functions on the boundary assures that all elements have about the same number of degrees of freedom.

Listing 3.3: the neighbour finding function; file `recursive_boundary_search.m`

```
function b = get_neighbours_bd(i,Mesh,elem_bd_list)
    % finds all elements which have at least
    % one vertex in common with i and returns
    % their boundary entries
    elem = Mesh.Elements;
    n = elem(:,1) == elem(i,1);
    for j = 1:3
        for k = 1:3
            n = n | elem(:,j) == elem(i,k);
        end
    end
    % get all boundary entries for elements in n
    b = unique(elem_bd_list(n,:));
    % list also contains 0, but thats not an actual
    % entry, remove it.
    b = b(b ~= 0);
end
```

Note that by combining the basis functions, with increasing number of basis functions per element the risk of ill conditioning increases. About 15 Bessel basis functions can approximate an exponential basis function pretty well.

4 Results and Comparisons

4.1 Linear Finite Elements

Experiment 1: h Convergence

Let's look at standard linear finite elements first. We take them as the reference to compare with the proposed DG method. As the the thickness of the skin layer varies with σ, we try three values of σ.

For the low $\sigma = 1$, and the skin depths equals the whole domain, that means there is no skin layer. Figure 4.1 shows the convergence for mesh refinement on the reference circular domain. Because there is no real skin layer, the finite element method converges quickly.

For a moderate $\sigma = 100$, the skin depth is 10% of the domain such that we do have a skin layer. Figure 4.2 shows a lower convergence rate and overall a higher relative error.

For the high $\sigma = 10^4$, the skin depth is now only 1% of the domain, which is a quite thin skin layer. From figure 4.3 we see that for coarse meshes (< 200 degrees of freedom), there is virtually no improvement. And for finer meshes, there is only a moderate convergence rate. The error stays above 10% for up to 10^4 degrees of freedom. The DG method will use a much coarser mesh with a similar amount of unknowns, because we need a lot more degrees of freedom per element compared to linear finite elements.

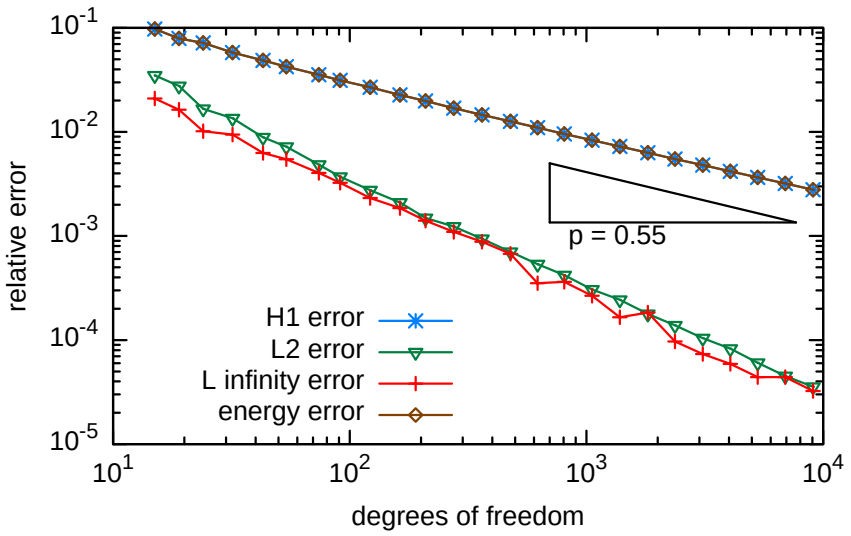


Figure 4.1: Experiment 1: Convergence of standard linear finite elements with $\sigma = 1$

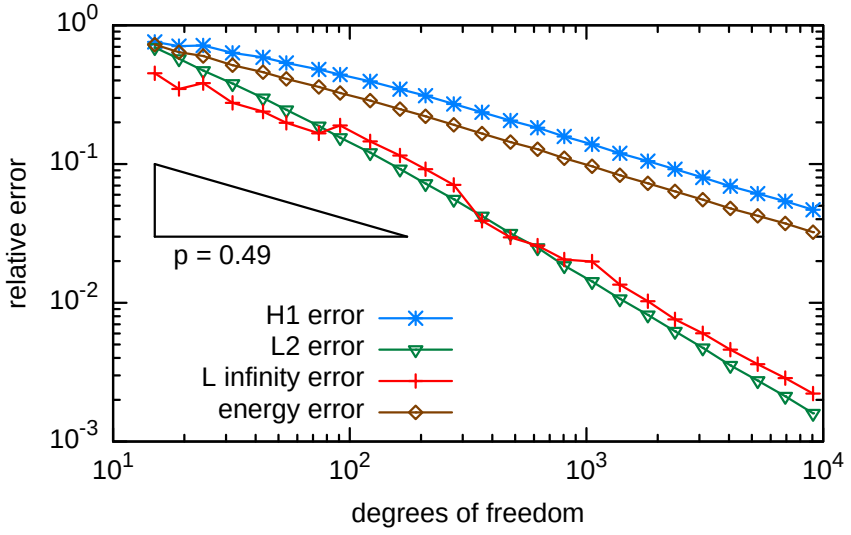


Figure 4.2: Experiment 1: Convergence of standard linear finite elements with $\sigma = 100$

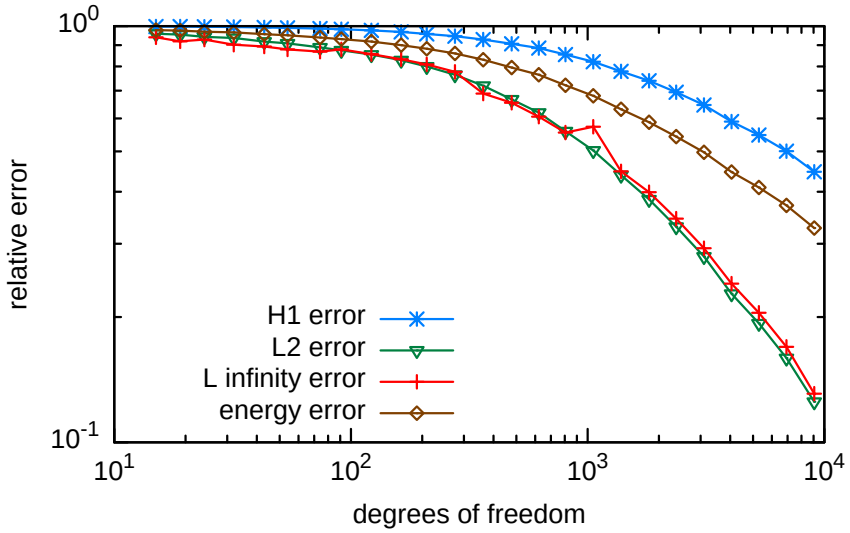


Figure 4.3: Experiment 1: Convergence of standard linear finite elements with $\sigma = 10^4$

4.2 DG with Exponential Basis Functions

Experiment 2: h Convergence

Let's repeat these test cases for the DG method. First for exponential basis functions as in section 3.2 with the same 5 directions on every element.

For $\sigma = 1$ (fig. 4.4) we already observe a higher convergence rate than with linear finite elements. On the next figure (4.5, $\sigma = 100$) the convergence rate has decreased, but it is still better than with linear finite elements. It fails for the high $\sigma = 10^4$ (fig. 4.6). In contrast to linear finite elements, relative errors above 1 are possible. The systems matrix is ill conditioned especially for coarse grids.

The result for $\sigma = 10^4$ can be improved by using more basis functions per element, as in figure 4.7. The one outlier at 418 degrees of freedom is probably caused by a particularly bad mesh and the huge condition number for coarse meshes.

Experiment 3: Convergence for Number of Basis Functions per Element

Convergence when using more basis functions per element on a fixed mesh, is illustrated in figure 4.8. Actually the observed convergence is a lot higher in terms of degrees of freedom than with mesh refinement. But it is not reliable, as the condition of the matrix increases with decreasing angles between the directions of the exponential basis functions. That means solutions will improve only to a certain point and can get really bad beyond that point because of ill-conditioning. This problem is addressed with the Bessel basis functions.

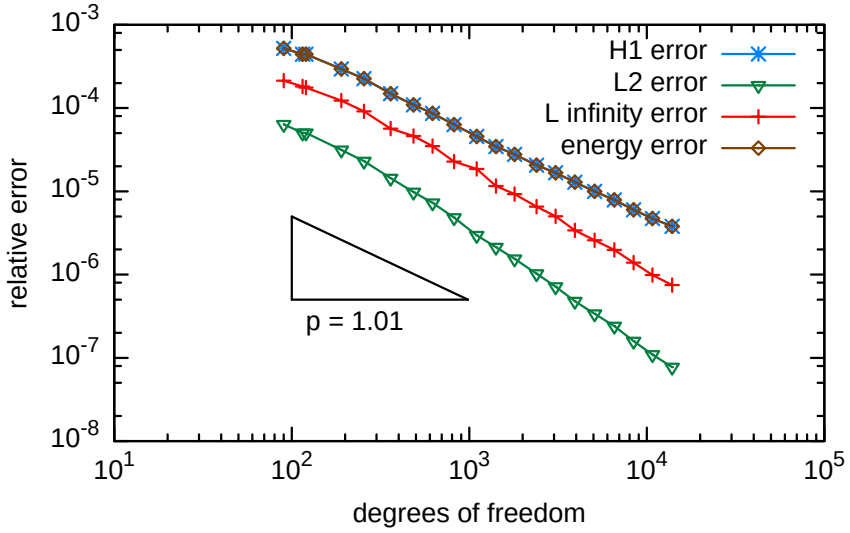


Figure 4.4: Experiment 2: Convergence of DG with exponential basis functions with $\sigma = 1$. Each element has the same set of basis functions. Number of basis functions $n = 5$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

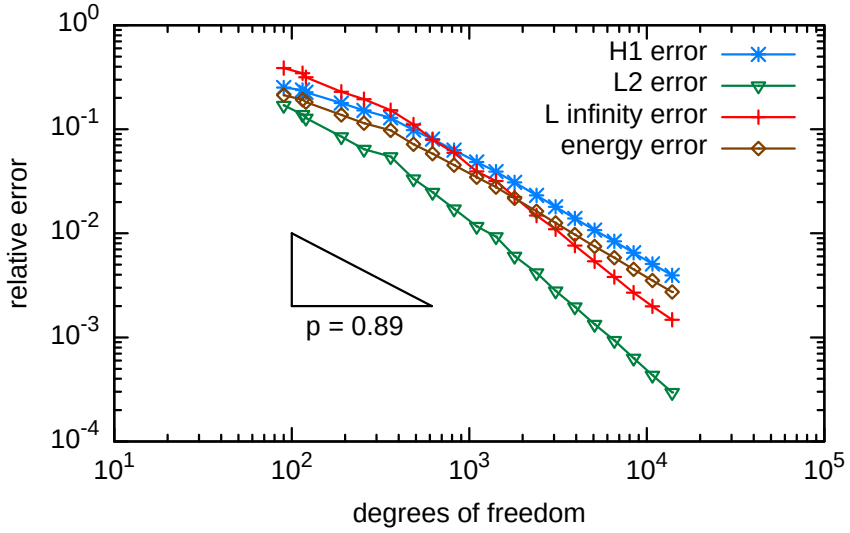


Figure 4.5: Experiment 2: Convergence of DG with exponential basis functions with $\sigma = 100$. Each element has the same set of basis functions. Number of basis functions $n = 5$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

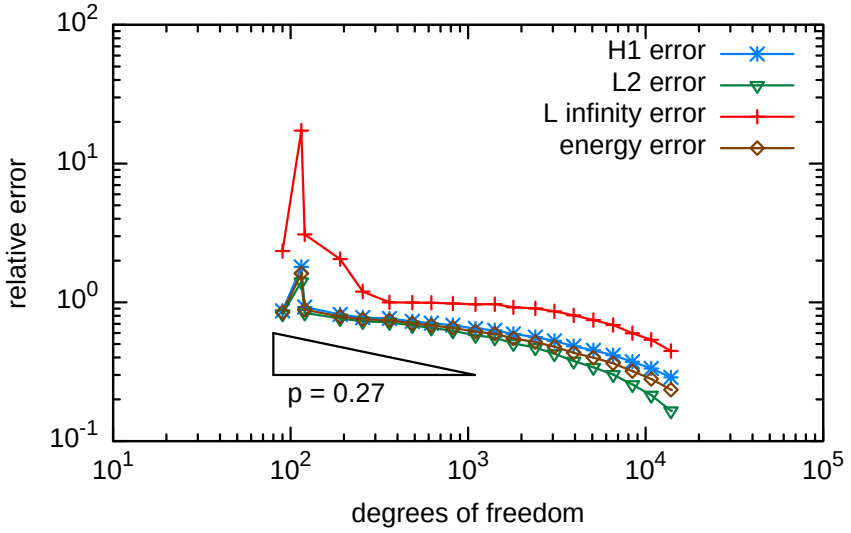


Figure 4.6: Experiment 2: Convergence of DG with exponential basis functions with $\sigma = 10^4$. Each element has the same set of basis functions. Number of basis functions $n = 5$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

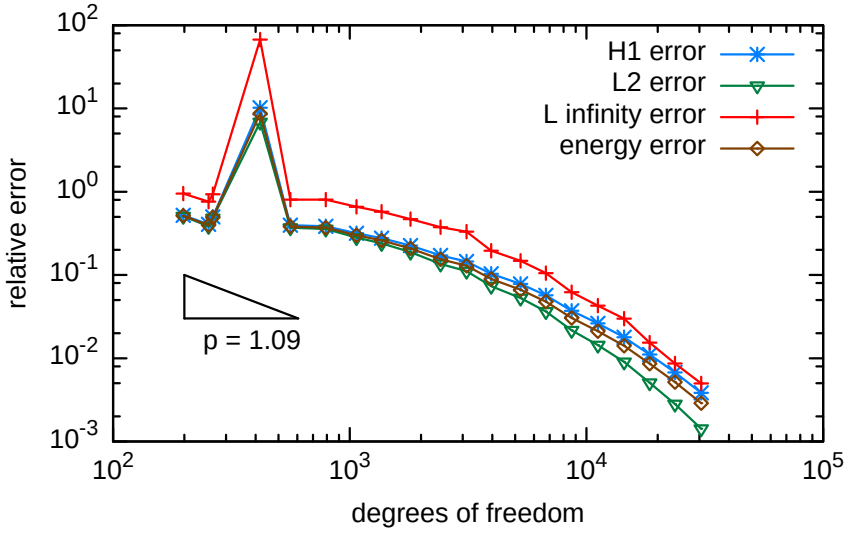


Figure 4.7: Experiment 2: Convergence of DG with exponential basis functions with $\sigma = 10^4$. Each element has the same set of basis functions. Number of basis functions $n = 11$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

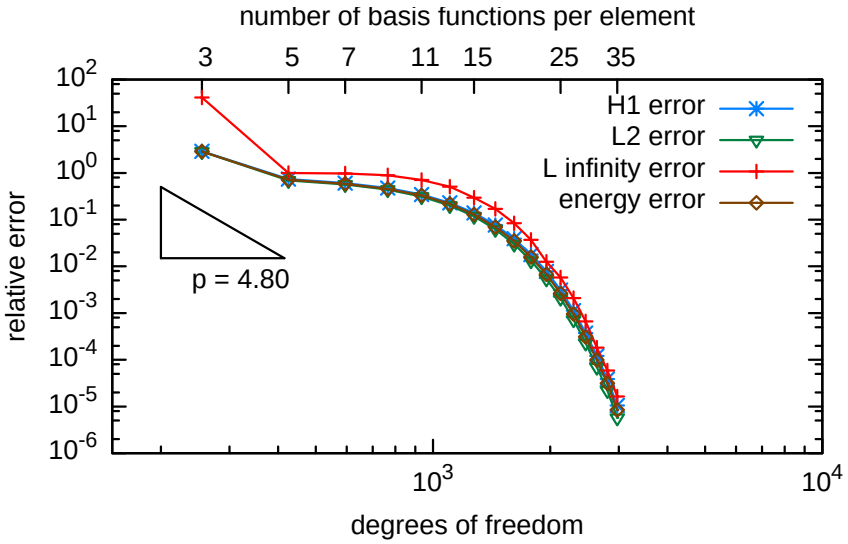


Figure 4.8: Experiment 3: Convergence of DG with exponential basis functions with $\sigma = 10^4$. Each element has the same set of basis functions. Number of basis functions n is varied on a fixed mesh of mesh width $h = 0.25$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

4.3 DG with Bessel Basis Functions

Experiment 4: h Convergence

Next we look at the Bessel basis functions, introduced in section 3.3. Bessel basis functions have the advantage of not suffering from ill-conditioning. But the drawback is we can not choose the direction of the skin layer. The first two examples, figure 4.9 with $\sigma = 1$ and figure 4.10 with $\sigma = 100$, are quite similar to the exponential case. The convergence rate does not decrease from the lower to the higher σ, but it is at a slightly lower level.

The interesting case of $\sigma = 10^4$ in figure 4.11 shows again a peak of huge error at about 260 degrees of freedom. It can only be attributed to an unfavourable mesh combined with quite few basis functions to resolve the narrow skin layer. Indeed it looks much better if we choose 11 basis functions per element, shown in figure 4.12. It does not suffer from ill-conditioning as with exponential basis functions, although the condition number increases with the number of basis functions, and decreases with the mesh width. But all this at a lower level.

Experiment 5: Convergence for Number of Basis Functions per Element

Let's look at the convergence on a fixed mesh with increasing number of basis functions per element n shown in figure 4.13. Again we observe a quicker convergence in terms of degrees of freedom compared to mesh refinement. A minimum number of basis functions per element is needed for the method to start converging, but then it converges quickly.

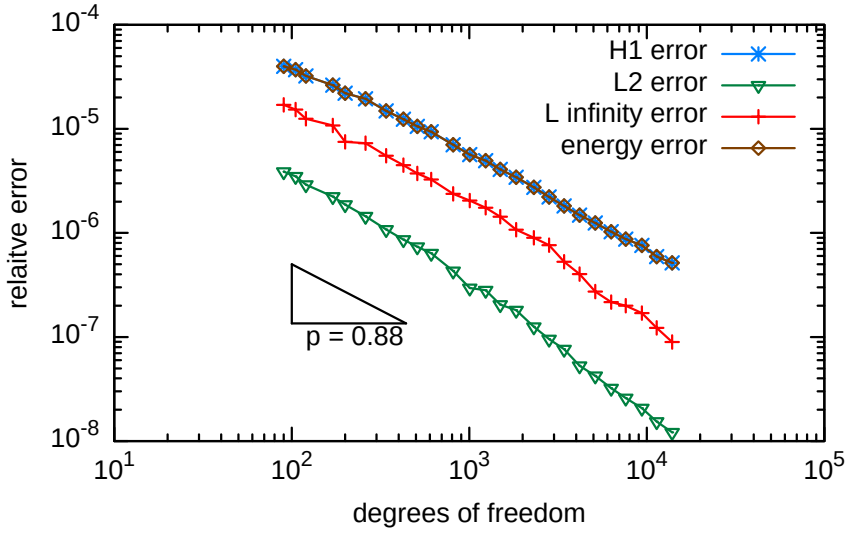


Figure 4.9: Experiment 4: Convergence of DG with Bessel basis functions with $\sigma = 1$. Each element has the same set of basis functions. Number of basis functions $n = 5$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

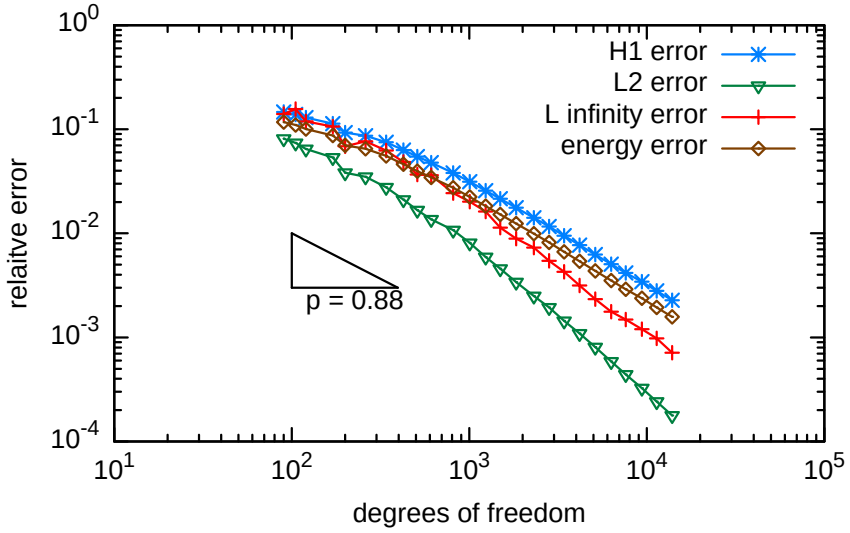


Figure 4.10: Experiment 4: Convergence of DG with Bessel basis functions with $\sigma = 100$. Each element has the same set of basis functions. Number of basis functions $n = 5$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

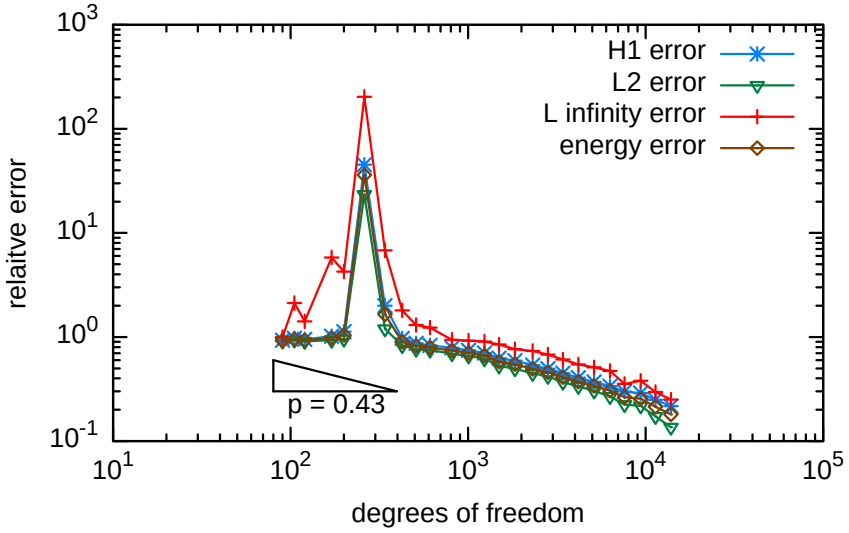


Figure 4.11: Experiment 4: Convergence of DG with Bessel basis functions with $\sigma = 10^4$. Each element has the same set of basis functions. Number of basis functions $n = 5$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

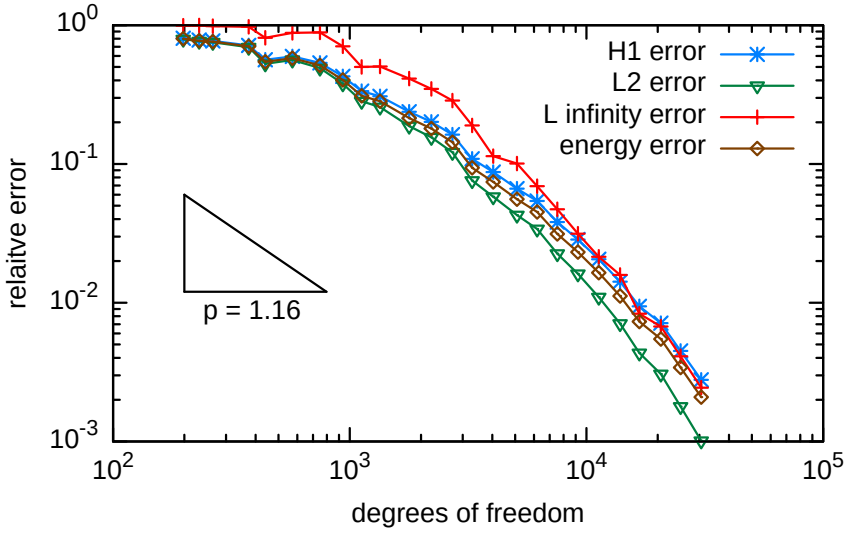


Figure 4.12: Experiment 4: Convergence of DG with Bessel basis functions with $\sigma = 10^4$. Each element has the same set of basis functions. Number of basis functions $n = 11$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

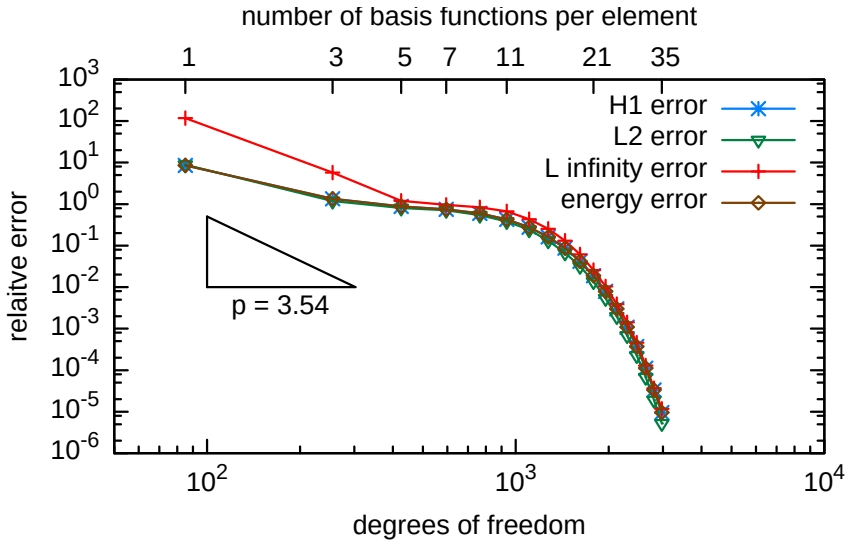


Figure 4.13: Experiment 5: Convergence of DG with Bessel basis functions with $\sigma = 10^4$. Each element has the same set of basis functions. Number of basis functions n is varied on a fixed mesh of mesh width $h = 0.25$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

4.4 DG with Element Specific Exponential Basis Functions

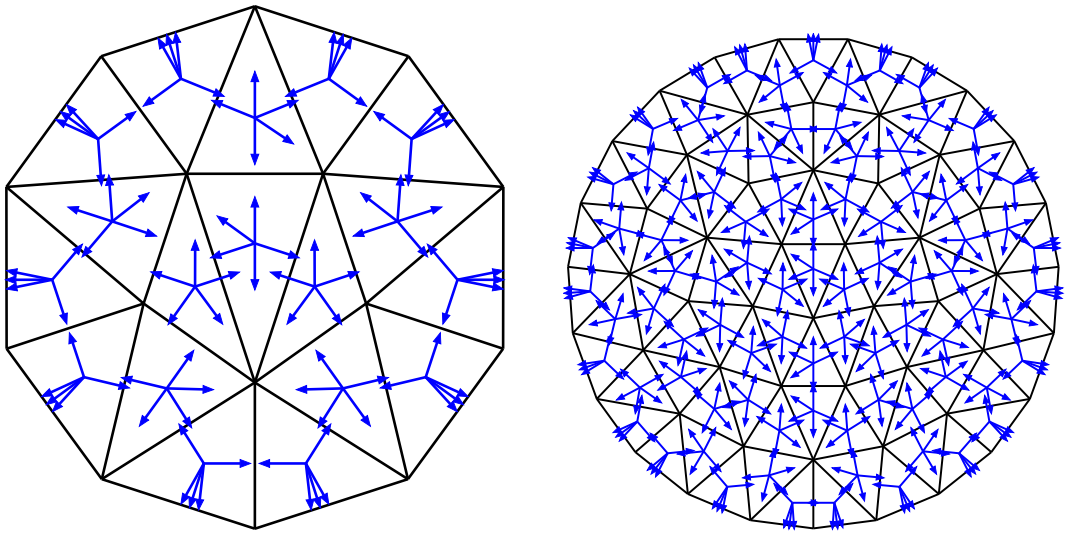


Figure 4.14: Two examples of element wise chosen directions for exponential basis functions. Each blue arrow resembles one basis function. Section 3.6.1 explains how the basis functions are chosen.

The exponential basis functions suffer from bad conditioning when we have too many of them in directions where we do not have a skin layer. But they are better suited to approximate a smooth skin layer. We can take advantage of the knowledge that the skin layer will follow the boundary and choose the directions for the exponential basis functions accordingly. In section 4.2 all elements had the same set of basis functions. Now let's choose the directions according to section 3.6.1. As an example see the figure 4.14.

Experiment 6: h Convergence

For a low σ like in figure 4.15, the convergence is actually a bit worse than with homogeneous directions on all elements. But this is not that astounding as the choice of directions is designed for an apparent skin layer.

In figure 4.16 we have a skin layer, although not a too narrow one. The element wise choice of basis functions has an effect especially on the error for coarse meshes. For finer meshes, where the element size is below the skin depth, the error is close to the one for a homogeneous choice of directions. This is because we have a quite optimal basis in the first row of elements at the boundary. If this first row is thicker than the skin depth, we successfully resolved the most important part of the problem with a good set of basis functions. Beyond that point, the arbitrary directions introduce a less optimal basis.

For $\sigma = 10^4$, figure 4.17, the same effect is observable. The relative errors start at a

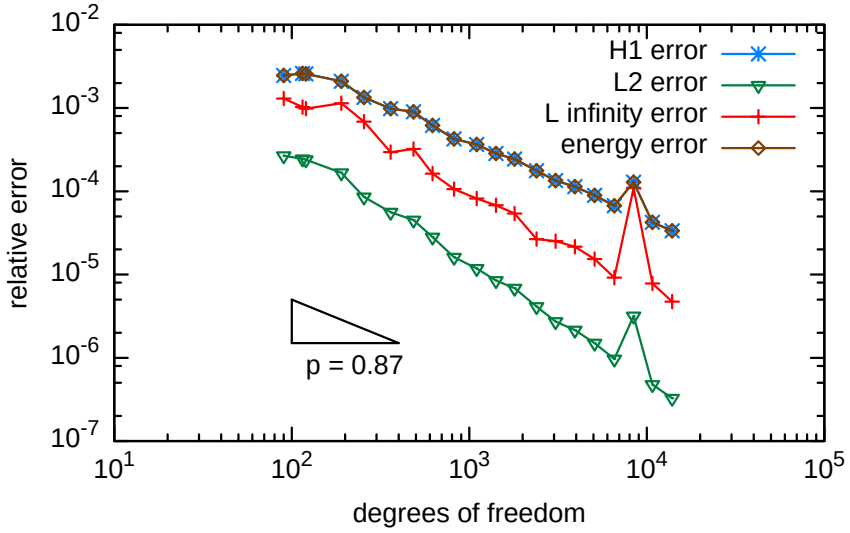


Figure 4.15: Experiment 6: Convergence of DG with exponential basis functions with $\sigma = 1$. Basis Functions are chosen according to element shape. Number of basis functions $n = 5$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

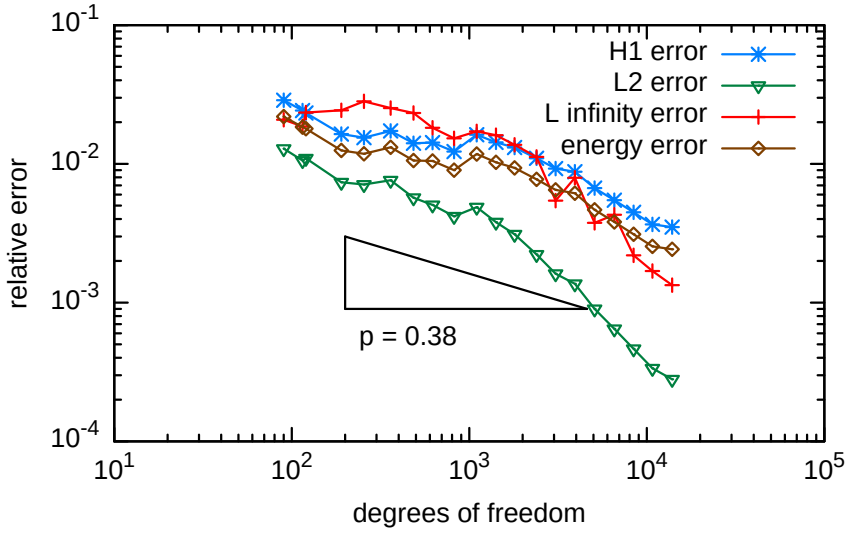


Figure 4.16: Experiment 6: Convergence of DG with exponential basis functions with $\sigma = 100$. Basis Functions are chosen according to element shape. Number of basis functions $n = 5$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

high level, but quickly drop below 10%.

Experiment 7: Convergence for Number of Basis Functions per Element

We can also study what happens when we add more basis functions in step 2 and 4. The additional basis functions on the boundary edges are restricted to six. More such basis functions would be too far off the normal to have an impact on the boundary layer resolution. The result is shown in figure 4.18. The error increases for high numbers of basis functions, which is probably caused by the ill-conditioning of the matrix.

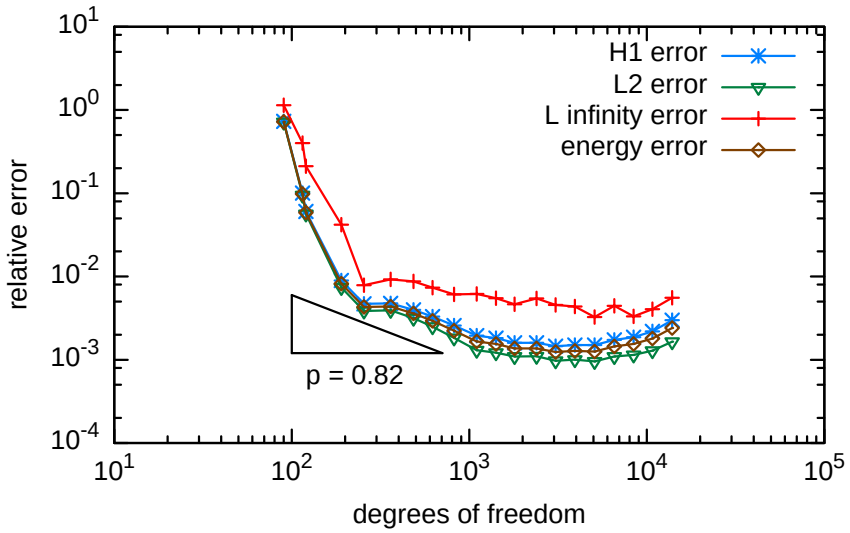


Figure 4.17: Experiment 6: Convergence of DG with exponential basis functions with $\sigma = 10^4$. Basis Functions are chosen according to element shape. Number of basis functions $n = 5$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

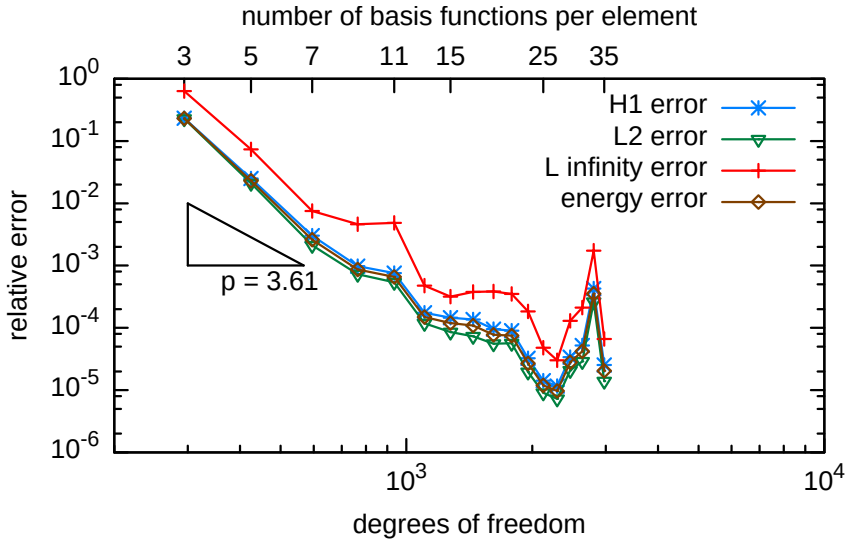


Figure 4.18: Experiment 7: Convergence of DG with exponential basis functions with $\sigma = 10^4$. Basis Functions are chosen according to element shape. Number of basis functions n is varied on a fixed mesh of mesh width $h = 0.25$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

4.5 DG with Exponential and Bessel Basis Functions

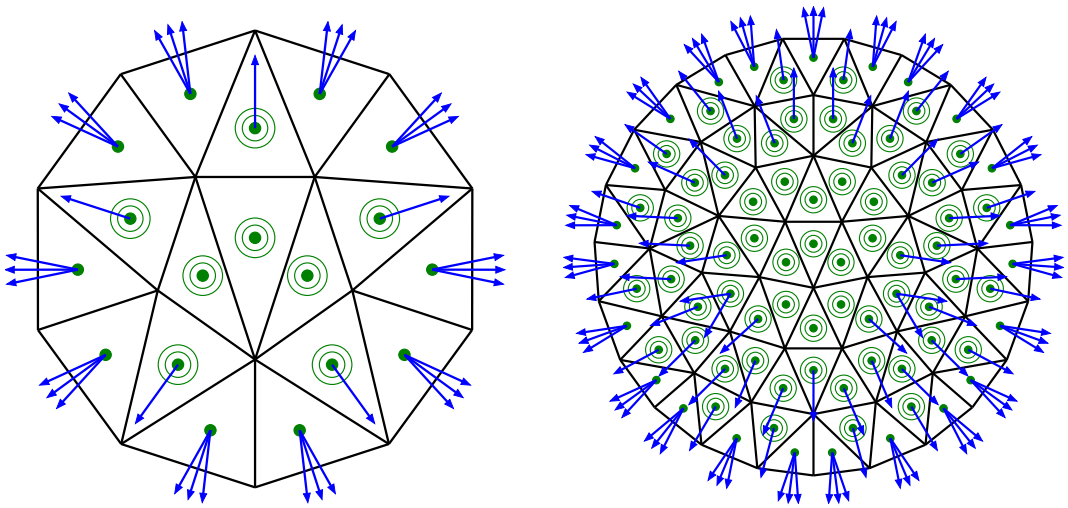


Figure 4.19: Two examples of element wise chosen set of exponential and Bessel basis functions according to section 3.6.2. Each blue arrow resembles one exponential basis function. The green circles represent the Bessel basis functions. The green dot stands for the single Bessel basis function of order zero, each green circle around it stands for a pair of Bessel basis functions of increasing order.

To amend the problem of ill condition of the exponential basis functions, but in order to still have a good basis in boundary proximity, we can mix Bessel and exponential basis functions. The set of basis functions is chosen according to section 3.6.2

Experiment 8: h Convergence

In the case of $\sigma = 1$, figure 4.20, the combination of exponential and Bessel basis functions is not favourable for coarse meshes. Relative errors are high compared to pure Bessel basis functions. But this case without a skin layer is not the focus of the method, and it still works reasonably well.

Figure 4.21, $\sigma = 100$, shows the same errors for coarse meshes as with exponential basis functions only. The combination with Bessel basis functions lead to clearly better conditioning of the matrix, which leads to a more continuous and faster decrease of the error.

In the extreme case of $\sigma = 10^4$ in figure 4.22, the error does not decrease that smoothly any more. But no significant increase is observed compared to pure exponential adaptive basis functions. And the relative errors drop quickly for coarse meshes to a quite low level.

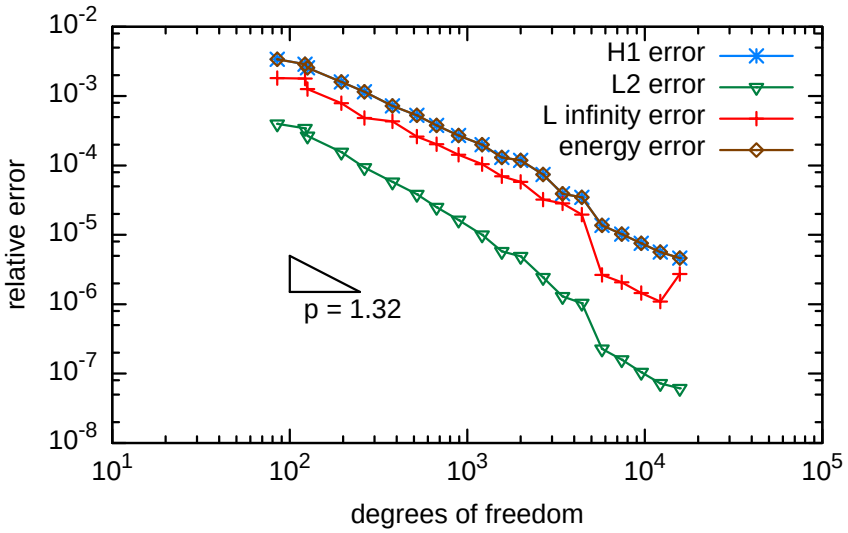


Figure 4.20: Experiment 8: Convergence of DG with exponential and Bessel basis functions with $\sigma = 1$. Basis Functions are chosen in relation to boundary edges. Number of basis functions per element n around 5. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

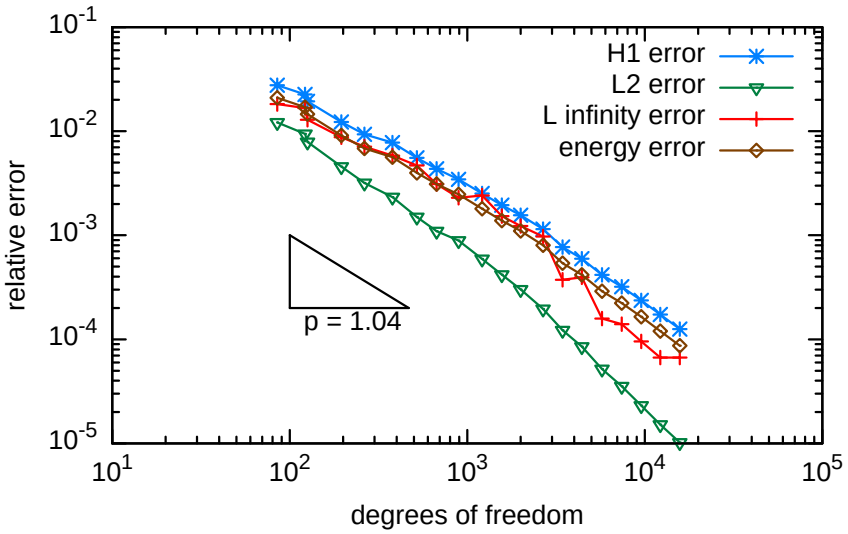


Figure 4.21: Experiment 8: Convergence of DG with exponential and Bessel basis functions with $\sigma = 100$. Basis Functions are chosen in relation to boundary edges. Number of basis functions per element n around 5. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

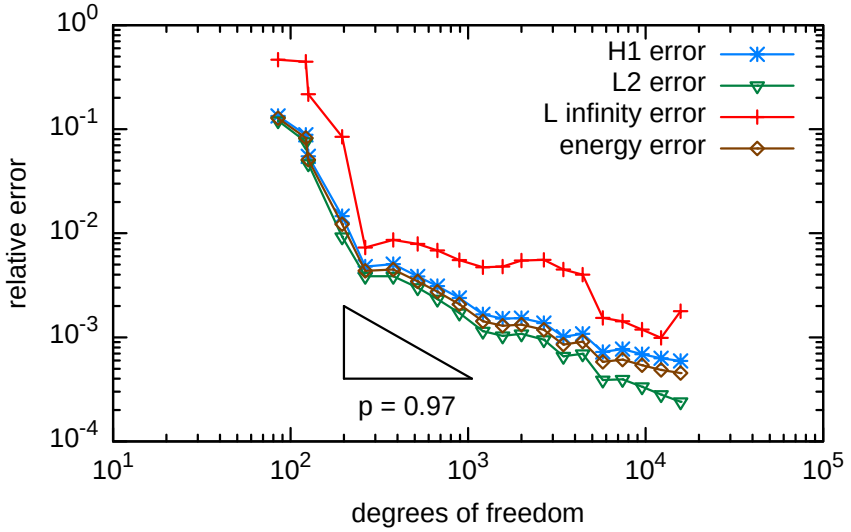


Figure 4.22: Experiment 8: Convergence of DG with exponential and Bessel basis functions with $\sigma = 10^4$. Basis Functions are chosen in relation to boundary edges. Number of basis functions per element n around 5. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

Experiment 9: Convergence for Number of Basis Functions per Element

Convergence for increasing number of basis functions per element is not straight forward, as enough Bessel basis functions can approximate an exponential basis function quite well. Although the aim was to circumvent ill conditioning by mixing both basis functions, this can go wrong if too many basis functions are taken. It depends on the skin depth, but for $\sigma = 10^4$ (skin depth 1%) about 17 Bessel basis functions is the maximum to use combined with exponential basis functions, as can be seen in figure 4.23.

In that figure, if n is the number of basis functions per element, $m = \min(n, 7)$ is the number of exponential basis functions used for each boundary edge. n Bessel basis functions are chosen for interior elements and $n - m + 1$ for boundary elements. Obviously the method fails for large numbers of basis functions. Non the less it reaches faster relative errors of the order of 10^{-6} than any other method presented here. The true strength though is the low relative error for few basis functions.

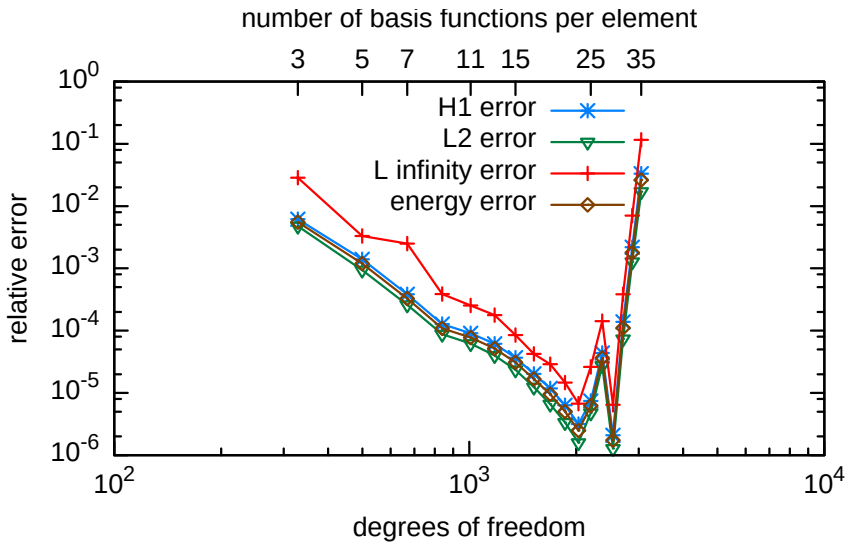


Figure 4.23: Experiment 9: Convergence of DG with exponential and Bessel basis functions with $\sigma = 10^4$. Basis Functions are chosen in relation to boundary edges. Number of basis functions n is varied on a fixed mesh of mesh width $h = 0.25$. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

4.6 Other Geometries

Experiment 10: L-Shape

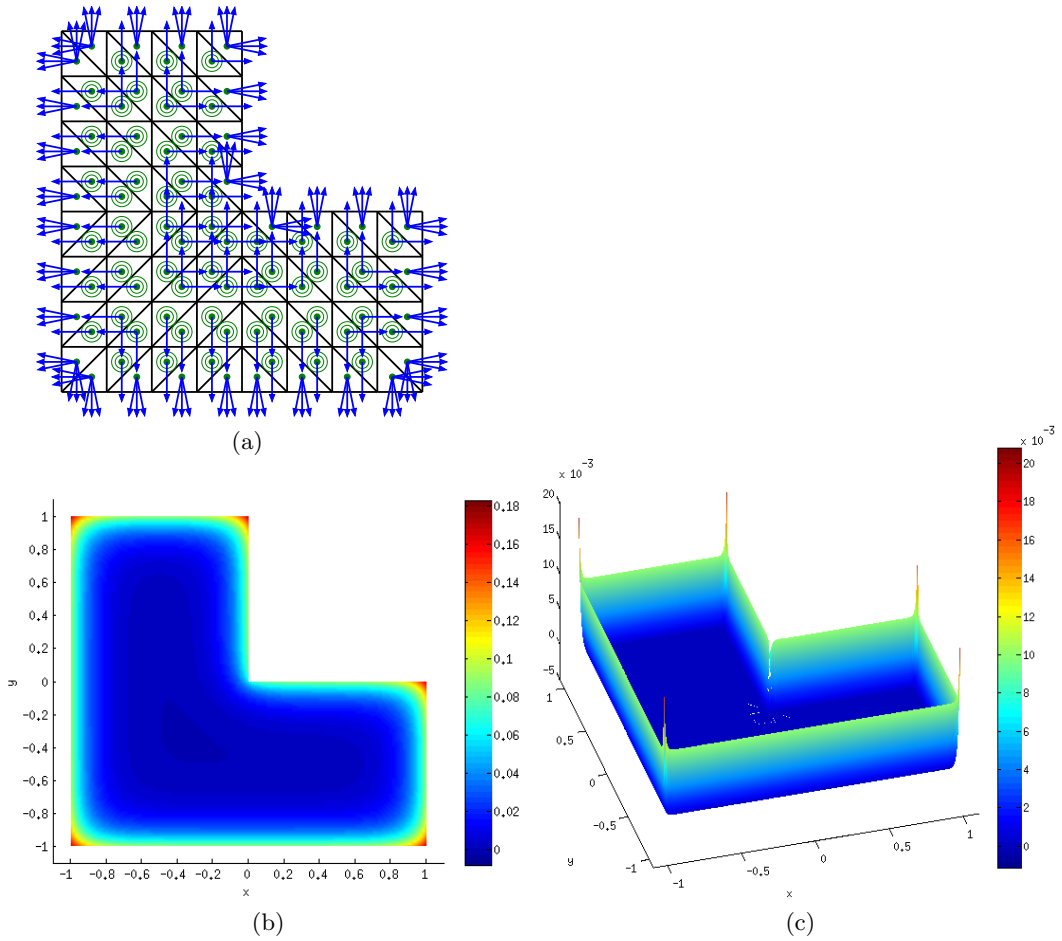
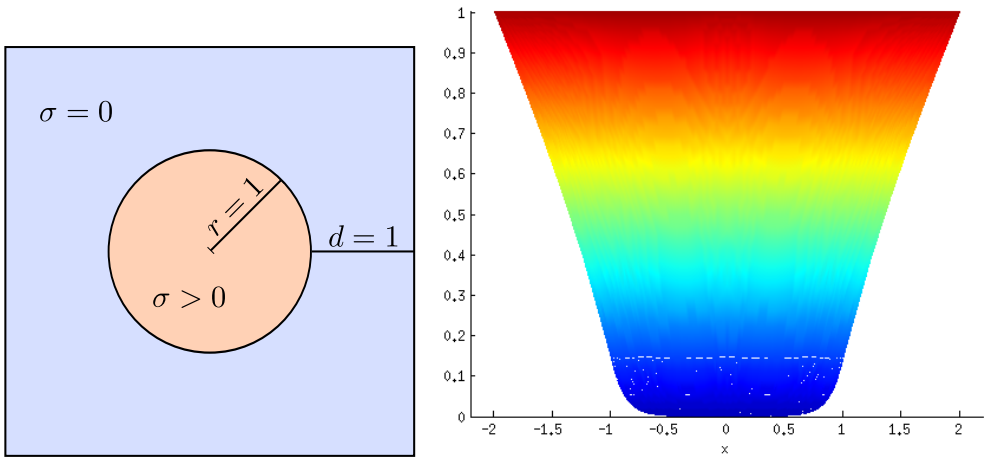


Figure 4.24: Experiment 10: two examples of an L-shaped domain with boundary condition $g(x, y) = 1$. (b) shows the solution for $\sigma = 100$, (c) shows a 3D plot of the thin skin layer for $\sigma = 10^4$. In both cases meshwidth $h = 0.25$, number of basis functions per element n around 5. See (a) for the chosen set of basis functions. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

The L-shape is a simple geometry without curved boundaries, but with corners. Figure 4.24 shows two examples calculated on the L-shape. Both examples are calculated on the same mesh with the mixed set of basis functions. The simulation with the DG method agrees well with the finite element solution. Note the tiny gap for the inwards pointing corner in 4.24c.

Experiment 11: Dirichlet Problem with Two Different σ Values



(a) (b)

Figure 4.25: Experiment 11: Domain with two different σ values and Dirichlet boundary condition $g = 1$. (b) shows the side view of the solution. $\sigma = 100$ for the inner circle. Meshwidth $h = 0.25$, number of basis functions per element n around 5. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

In all previous experiments, σ was constant over the whole domain. Here we split the domain into two regions (see figure 4.25a): The domain is bounded by a square of side length four. In the center of it is a circle with radius one where $\sigma > 0$, and outside this circle is $\sigma = 0$. There is the restriction that σ is constant on each element, therefore the mesh is chosen such that the affected edges approximate the circle well. This experiment uses a different boundary condition: $u = g$ on $\partial\Omega$.

The set of basis functions is a bit different for the special case $\sigma = 0$, because exponential or Bessel basis functions would only give 1 as a basis function and nothing else. Instead simple linear, continuous basis functions are used on elements where $\sigma = 0$, but with the DG formulation. The solution inside the circle is related to the normal solution on a circle. See figure 4.25b, in the interval between -1 and 1. On the circle boundary, the value of the solution u is already quite low. It goes to zero for increasing σ. Figure 4.27 shows this behaviour. The value of a point on the circle is plotted against σ. As mentioned in the introduction, heating due to ohmic losses depend on this diminishing part of the solution. The ohmic losses are calculated as $\int_\Omega \sigma(\vec{x})|u(\vec{x})|^2 \mathrm{d}V$. Figure 4.26 shows the ohmic losses calculated in this example.

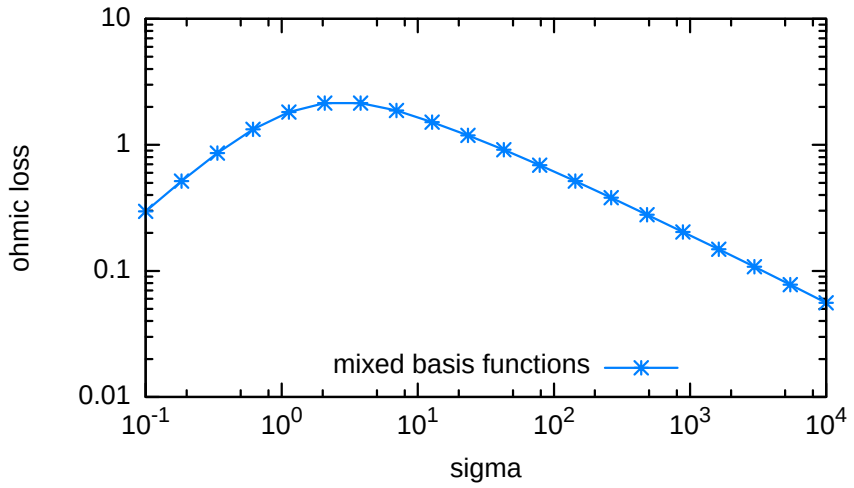


Figure 4.26: Experiment 11: Ohmic losses for varying σ. Dirichlet boundary condition $g = 1$, Meshwidth $h = 0.25$, number of basis functions per element n around 5 inside the circle. Outside the circle continuous linear basis functions are used. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

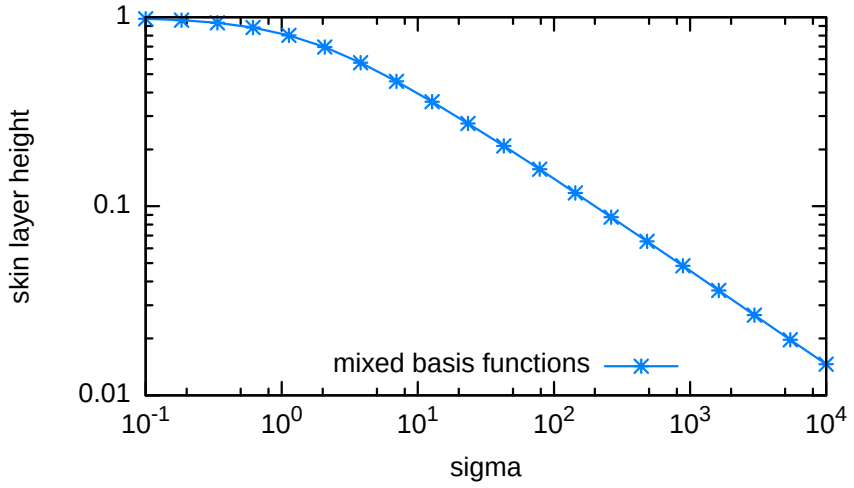


Figure 4.27: Experiment 11: height of the skin layer for varying σ. Dirichlet boundary condition $g = 1$, Meshwidth $h = 0.25$, number of basis functions per element n around 5 inside the circle. Outside the circle continuous linear basis functions are used. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

4.7 Error for Varying σ

Experiment 12: Comparison FEM and DG for Varying σ

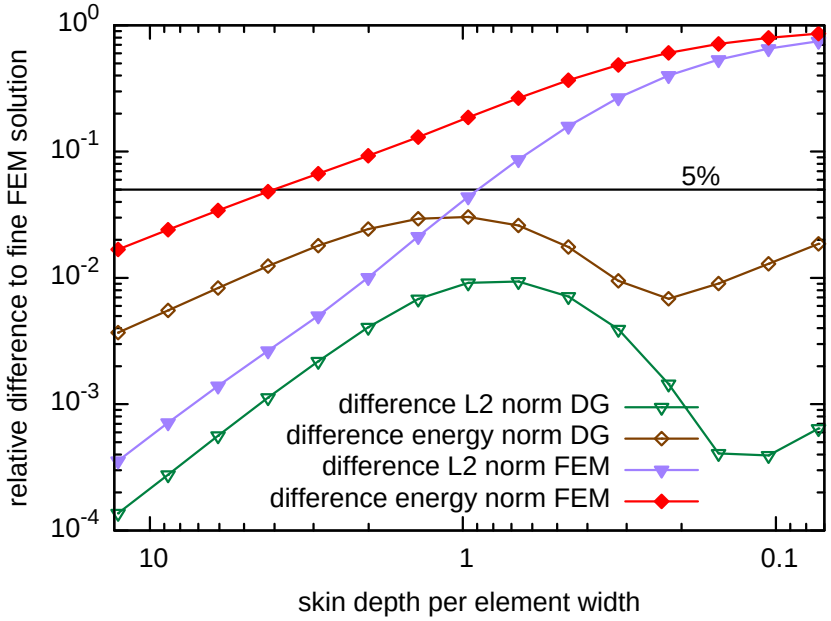


Figure 4.28: Experiment 12: Comparison of linear finite elements and the DG method with element adaptive mixed basis functions for varying σ on a square domain. Number of basis functions per element n around 5 for DG method. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

As mentioned in the introduction, standard finite elements can not resolve skin depths below an element width, which is the motivation for this work. Therefore, in this experiment linear finite elements and the DG method are compared for decreasing skin depth. The simulation is carried out on a square domain with boundary condition $g = 1$, with a fixed mesh. In figure 4.28 σ is varied. The x axis shows the skin depth per element width, which is $(\sqrt{\sigma}h)^{-1}$. As there is no analytical solution on the square, the solution is compared to a very fine linear finite element mesh. The y axis shows the relative error to this reference solution. As expected, for $(\sqrt{\sigma}h)^{-1} < 4$, which means less than four elements to resolve the skin depth, linear finite elements error get above the 5% mark, which is usually targeted in industrial applications.

Although both methods use the same mesh, the DG method uses 704 degrees of freedom in contrast to 81 for linear finite elements. This is the disadvantage of the discontinuous basis functions. Obviously the DG method behaves a lot better for high σ values, it stays at an acceptable error range. For small skin depths one can see the error of the DG method going up. One factor for this is surely the error in the reference

solution. At $(\sqrt{\sigma}h)^{-1} = 0.1$ the reference solution has about ten elements per skin depth, which has an estimated L_2 error of order of magnitude 10^{-4}. This is the same order of magnitude we see in the L_2 error for the DG method.

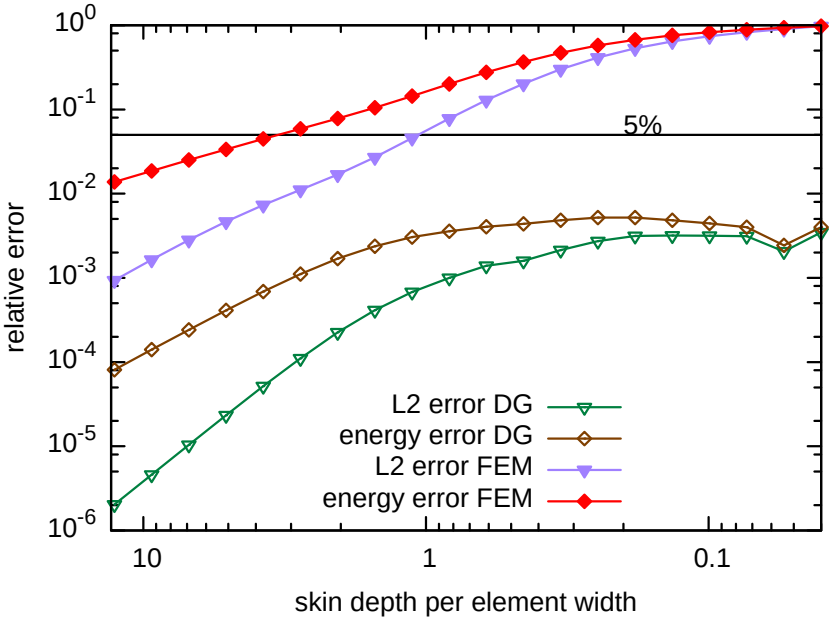


Figure 4.29: Experiment 12: Comparison of linear finite elements and the DG method with element adaptive mixed basis functions for varying σ on the circular domain. Number of basis functions per element n around 5 for DG method. Parameters $\alpha = n^2/(h\sqrt{\sigma})$, $\beta = 0.5$, $\delta = 0$ and $\vec{\zeta} = \vec{0}$

Another example is figure 4.29, where the same is calculated on the circular domain. This comparison is slightly less fair, as no curved boundaries have been implemented for linear finite elements. But that should not have a large impact on the error. On the other hand we are able to calculate the exact error, because the analytic solution is known. This makes it more reliable for small skin depths. It can be seen that the error stays quite stable for up to $\sigma = 10^4; (\sqrt{\sigma}h)^{-1} = 0.04$. Although there is a small kink visible, which has not been further investigated for $\sigma > 10^4$. Note that in contrast to the square, the circle geometry does not have corners, which might influence the behaviour of the error with increasing skin depth.

5 Conclusions and Outlook

5.1 Conclusions

The goal was to have a method which resolves skin layers accurately, even if the skin depth is smaller than an element width. The DG method presented here certainly meets this requirement. If the mesh resolves the geometry accurately enough, then it is able to solve the problem for a wide range of skin depths, up to a skin depth of at least 0.04 of the element size.

Several sets of basis functions have been presented, and two of them work particularly well. The best set, if you want to have a good solution with a moderate amount of basis functions per element, would be the mixed set presented in section 3.6.2. It uses the knowledge of the boundary and thus can approximate the skin layer quite well with only a few basis functions. It has some problems when a lot of basis functions per element are used, because a certain amount of Bessel basis functions can approximate an exponential basis function quite well. Then the system of equations get degenerate. In that case, using only Bessel basis functions is the right choice. The results for a few basis functions per element are worse, but it is possible to add more of them than with a mixed set of basis functions.

One of the drawbacks of the DG method is certainly the much larger number of degrees of freedom for the same mesh. But being able to resolve skin layers which are smaller than an element width should more than compensate for that, as one can choose coarser meshes, if the geometry permits it. Another disadvantage is that the DG method can produce relative errors much larger than 1, which does not occur for linear finite elements.

5.2 Future Work and Outlook

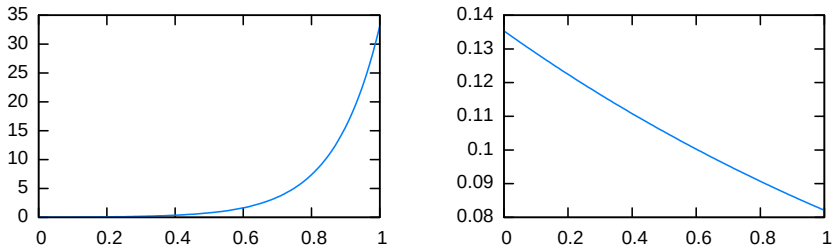
The method presented here was implemented only for 2D, for a differential equation which was reduced to the relevant behaviour. The next steps would be towards integrating this method in an electromagnetic simulation, in 3D with the actual differential equations. Combining linear finite elements and the Trefftz basis functions should also be interesting. That would allow to use the Trefftz basis functions only in regions where a skin layer exists.

The integrals should also be further investigated. The exponentials could be integrated analytically. However the Bessel basis functions don't seem to be analytically integrable. Therefore a numerical integration is needed. The problem is that the functions can get really steep towards the integration boundary with increasing σ. To be sure to resolve

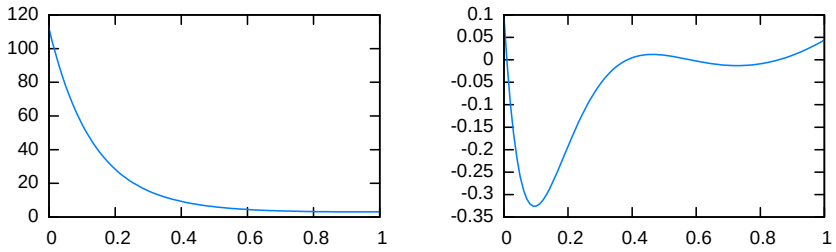
this properly, a large number of quadrature points were used in the implementation. A better approach would be to use an adaptive integration scheme. Potential maxima of the integral are known. They are at the integral boundaries, and, if Bessel basis functions are involved, at the maxima of sin or cos which can be mapped onto the integration domain. For some examples of these Integrals see figure 5.1. The examples are for a medium σ of 100, in order to see the behaviour without zooming in. For thinner skin layers, the integrals get also thinner, especially at the integration boundaries. The smallest features of the integrals scale with $(2\sqrt{\sigma}h)^{-1}$.

Acknowledgements

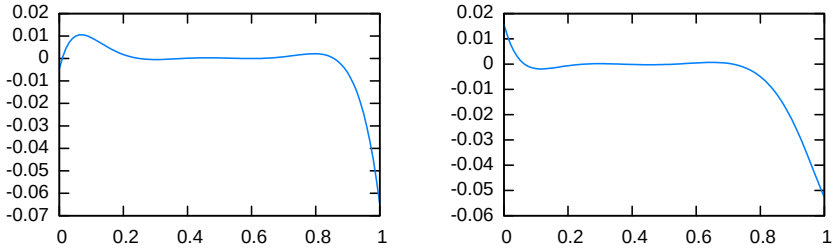
I would like to thank Florian Krämer for helping me through my master thesis, Prof. Hiptmair as my advisor, and Jörg Ostrowski and the ABB for the cooperation on this thesis.



(a) exponential: $\int_0^1 \exp\left(\sqrt{\sigma}\vec{\lambda}(t)\vec{\theta}_1\right) \exp\left(\sqrt{\sigma}\vec{\lambda}(t)\vec{\theta}_2\right) \mathrm{d}t$



(b) exponential and Bessel: $\int_0^1 \exp\left(\vec{\lambda}(t)\vec{\theta}_1\right) I_k\left(\sqrt{\sigma}r(t)\right) \mathsf{trig}\left(k\phi(t)\right) \mathrm{d}t$



(c) Bessel: $\int_0^1 I_k\left(\sqrt{\sigma}r(t)\right) \mathsf{trig}\left(k\phi(t)\right) I_\nu\left(\sqrt{\sigma}r(t)\right) \mathsf{trig}\left(\nu\phi(t)\right) \mathrm{d}t$

Figure 5.1: Examples of the integrals solved to calculate the matrix elements. $\vec{\lambda}(t) = P_{\text{start}} + t(P_{\text{end}} - P_{\text{start}})$ returns the coordinates along the edge from P_{start} to P_{end}. $\vec{\theta}_1$ and $\vec{\theta}_2$ are the vectors which represent the directions of the exponential basis function, $\vec{\theta}_1 = [\cos(\alpha_1), \sin(\alpha_1)]$. I_k is the modified Bessel function of the first kind of order k. $r(t)$ and $\phi(t)$ return the coordinates along the edge in polar coordinates. trig represents one of sin or cos.

Bibliography

[1] J. D. Jackson, *Classical Electrodynamics* (John Wiley & Sons, 1975).

[2] D. Braess, *Finite Elemente* (Springer, 2007).

[3] S. C. Brenner and R. L. Scott, *The Mathematical Theory of Finite Element Methods (Texts in Applied Mathematics)*, 3rd ed. (Springer, 2007).

[4] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini, SIAM J. Numer. Anal. **vol. 39 No. 5**, 1749 (2002).

[5] P. Castillo, B. Cockburn, I. Perugia, and D. Schotzau, SIAM Journal on Numerical Analysis **38**, 1676 (2001).

[6] C. J. Gittelson, R. Hiptmair, and I. Perugia, ESAIM **M2AN 43**, 297 (2009).

[7] O. Cessenat and B. Despres, SIAM Journal on Numerical Analysis **35**, 255 (1998).