ENGINEERING MATHEMATICS
AND COMPUTING LAB

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Using CutFEM for solving the linearized Poisson-Boltzmann equation: Introductory application in HiFlow3

Jonas Roller, Valentin Schmid, Philipp Gerstner, Jakob Niessner, Vincent Heuveline

Preprint No. 2024-01

Preprint Series of the Engineering Mathematics and Computing Lab (EMCL)

www.emcl.iwr.uni-heidelberg.de

Preprint Series of the Engineering Mathematics and Computing Lab (EMCL)
ISSN 2191–0693
Preprint No. 2024-01

Affiliation of the Authors

Jonas Roller[a], Valentin Schmid[a], Philipp Gerstner[a], Jakob Niessner[a,1], Vincent Heuveline[a]

[a] *Engineering Mathematics and Computing Lab (EMCL), Interdisciplinary Center for Scientific Computing (IWR), Heidelberg University, Germany*
[1] *Corresponding Author: Jakob Niessner, jakob.niessner@uni-heidelberg.de*

Impressum

Heidelberg University
Interdisciplinary Center for Scientific Computing (IWR)
Engineering Mathematics and Computing Lab (EMCL)

Im Neuenheimer Feld 205,
69120 Heidelberg
Germany

Using CutFEM for solving the linearized Poisson-Boltzmann equation: Introductory application in HiFlow3

Jonas Roller, Valentin Schmid, Philipp Gerstner, Jakob Niessner, Vincent Heuveline

July 5, 2024

Abstract

The Poisson-Boltzmann equation(PBE) is a fundamental equation for accurate description of electrostatics in ionic solutions or plasma. Hence, it finds application in a wide variety of fields including plasma physics, computational biology, colloidal and interface science and chemistry. In this preprint, we describe a first approach for solving it in a linearized setup using the CutFEM method in the software framework HiFlow^3. The considered methods are easy to use, but not oriented towards extracting optimal computing time. For this purpose,the weak form of the linear PBE is first derived. Then, the CutFEM based numerical method is proposed. Finally, the most important code sections in HiFlow^3 are explained in more details. The electrostatic potential for ubiquitin and the adenovirus virion molecules are calculated and presented in a numerical example at the end of the paper.

1 Introduction

The Poisson-Boltzmann equation(PBE) is a fundamental equation for accurate description of electrostatics in ionic solutions or plasma. Hence, it finds application in a wide variety of fields including plasma physics, computational biology, colloidal and interface science and chemistry. The equation is derived under the assumption that freely mobile ions can be modeled implicitly via a Boltzmann term, which locally describes their concentration in dependence of the local potential. The equation makes use of two external position dependent parameters: The local relative permittivity $\epsilon(\mathbf{x})$ and the ion accessibility $\kappa(\mathbf{x})$. In many numerical applications, these two parameters are modeled via discontinuous functions. Generally, there is a constant dielectric permittivity inside the atoms and molecules and a (different) constant relative permittivity in the surrounding medium. Similarly, the ion accessibility assumes either zero in a region around the molecule, which implicitly modeled ions can not reach and κ_0, the inverse Debye length. The recent development of the CutFEM^2 method provides an ideal framework for the solution of PDE depending on such discontinuous parameters. In this preprint, we therefore describe a first approach for solving the linearized Poisson-Boltzmann equation using the CutFEM technique. The considered method is easy to use in HiFlow^3 but not oriented towards extracting optimal computing time. HiFlow^3 is an in-house developed multi-purpose finite element software providing powerful tools for efficient and accurate solution of a wide range of problems modeled by partial differential equations (PDEs).[4]

2 Mathematical Setup

2.1 Problem

Consider a molecule with N atoms inside the domain Ω. The Poisson-Boltzmann equation is given by

$$-\frac{e_c}{k_B T}\nabla \cdot (\epsilon(\mathbf{x})\nabla\Phi(\mathbf{x})) + \bar{\kappa}^2(\mathbf{x})\sinh\left(\frac{e_c\Phi(\mathbf{x})}{k_B T}\right) = \frac{e_c^2}{k_B T\epsilon_0}\sum_{i=1}^{N} z_i\delta(\mathbf{x}-\mathbf{x}_i) \tag{1}$$

$$\Phi(\infty) = 0, \tag{2}$$

where $\Phi(\mathbf{x})$ is the electrical potential at the point $\mathbf{x}$, T is the temperature, k_B the Boltzmann constant, ϵ_0 is the vacuum permittivity, e_c is the elementary charge, $\mathbf{x}_i$ is the position of the ion i and z_i is the valency of the ion i. Using the non-dimensional potential

$$u(\mathbf{x}) = \frac{e_c\Phi(\mathbf{x})}{k_B T} \tag{3}$$

the equation (1) gets

$$-\nabla \cdot (\epsilon(\mathbf{x})\nabla u(\mathbf{x})) + \bar{\kappa}^2(\mathbf{x})\sinh(u(\mathbf{x})) = \frac{e_c^2}{k_B T\epsilon_0}\sum_{i=1}^{N} z_i\delta(\mathbf{x}-\mathbf{x}_i) \tag{4}$$

$$u(\infty) = 0. \tag{5}$$

Furthermore, if the domain can be divided into the three regions Ω_1: location of the molecule, Ω_2: the ion exclusion layer and Ω_3 the solvent region, we define the dielectric function as

$$\epsilon(\mathbf{x}) = \begin{cases} \epsilon_1, \ \textit{for } \mathbf{x} \in \Omega_1, \\ \epsilon_2 \ \textit{for } \mathbf{x} \in \Omega_2 \cup \Omega_3, \end{cases}$$

and the *modified Debye-Hückel parameter* $\bar{\kappa}(\mathbf{x})$ as

$$\bar{\kappa}(\mathbf{x}) = \begin{cases} 0, \ \textit{for } \mathbf{x} \in \Omega_1 \cup \Omega_2, \\ \sqrt{\epsilon_3}\kappa \ \textit{for } \mathbf{x} \in \Omega_3, \end{cases}$$

where κ is the inverse *Debye* length. By approximating $\sinh(\mathbf{x})$ with the first-order Taylor approximation, the linearized equation

$$-\nabla \cdot (\epsilon(\mathbf{x})\nabla u(\mathbf{x})) + \bar{\kappa}^2(\mathbf{x})u(\mathbf{x}) = C\sum_{i=1}^{N} z_i\delta(\mathbf{x}-\mathbf{x}_i), \tag{6}$$

$$u(\infty) = 0, \tag{7}$$

with $C := \frac{e_c^2}{\epsilon_0 k_B T}$ is obtained, which will be solved in this tutorial.

2.2 Domain and interface conditions

Following the approach proposed in,[6] we consider a computational domain $\Omega \subset \mathbb{R}^3$ of hexahedral shape, that is split into three parts: the molecule domain Ω_1, the ion exclusion layer (IEL) Ω_2 and the solvent domain Ω_3. The interface that separates the molecule domain from the surrounding solute is defined as the Solvent-Excluded-Surface (SES)[9] and denoted by $\Gamma_{12} = \Omega_1 \cap \Omega_2$, see Fig. 1.The SES describes the surface that a probe of a certain radius cannot penetrate around a molecule. Across this interface and across $\Gamma_{23} = \Omega_2 \cap \Omega_3$, it is assumed that the potential and its normal flux are continuous, i.e.

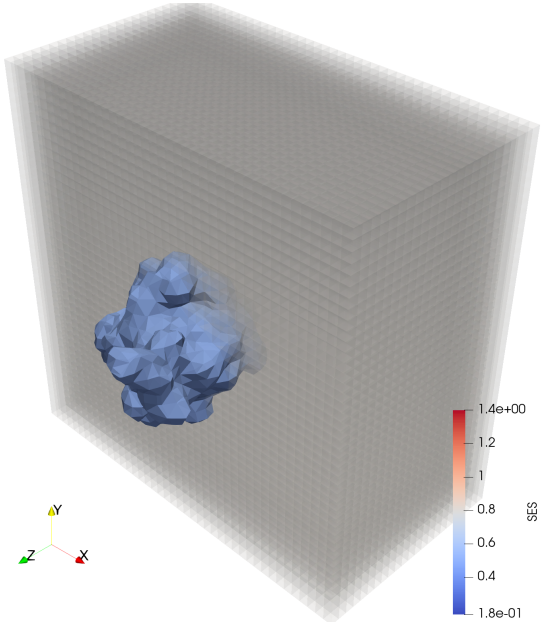


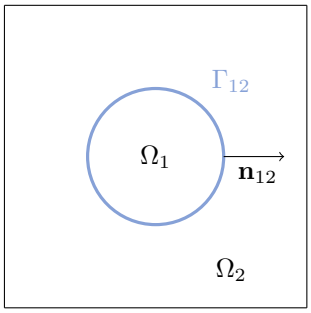
Figure 1: Image of the SES (light blue) calculated in HiFlow[3] for the case of the ubiquitin molecule (PDB ID: 1UBQ). The solvent domain used for this example can be seen in light gray.

$$\phi_1(\mathbf{x}) = \phi_2(\mathbf{x}), \qquad \mathbf{x} \in \Gamma_{12} \tag{8}$$

$$\epsilon_m \nabla\phi_1(\mathbf{x}) \cdot \mathbf{n}_{12}(\mathbf{x}) = \epsilon_s \nabla\phi_2(\mathbf{x}) \cdot \mathbf{n}_{12}(\mathbf{x}), \qquad \mathbf{x} \in \Gamma_{12} \tag{9}$$

$$\phi_2(\mathbf{x}) = \phi_3(\mathbf{x}), \qquad \mathbf{x} \in \Gamma_{23} \tag{10}$$

$$\epsilon_s \nabla\phi_2(\mathbf{x}) \cdot \mathbf{n}_{23}(\mathbf{x}) = \epsilon_s \nabla\phi_3(\mathbf{x}) \cdot \mathbf{n}_{23}(\mathbf{x}), \qquad \mathbf{x} \in \Gamma_{23}. \tag{11}$$



Here, $\phi_i := \phi|_{\Omega_i}$, $\mathbf{n}_{12}$ denotes the unit normal vector for Γ_{12}, directed from Ω_1 into Ω_2; $\mathbf{n}_{23}$ is defined analogously for Γ_{23} and directed from Ω_2 into Ω_3.

While the potential is continuous across Γ_{12} according to (8), condition (9) implies a jump in the normal derivative of ϕ for $\epsilon_m \neq \epsilon_s$. Thus, the shape of the potential exhibits some kind of kink across the SES, which must be taken care of when devising the numerical method. Since there is no change in permittivity across Γ_{23}, one does not have to cope with this issue there. Coming to the boundary conditions, we note that the physically correct conditions is $\phi(\infty) = 0$. Since we are restricted to computational domains of finite extension, we approximate this condition by

$$\phi(\mathbf{x}) = \phi_D(\mathbf{x}), \mathbf{x} \in \Gamma_D = \partial\Omega = \partial\Omega_3, \tag{12}$$

for some suitable Dirichlet boundary condition ϕ_D, as defined below.

2.3 Regularization and potential decomposition

The form of the right-hand-side term in (6) involves a sum over Dirac distributions and is thus singular at the atom center positions $\mathbf{x}_i \in \Omega_1$. To avoid resulting problems in the numerical method, this term has to be regularized. A commonly used technique for this purpose, see[7] for instance, is based on the observation that the function

$$G(\mathbf{x}) = \frac{C}{4\pi\epsilon_m} \sum_{i=1}^{N} \frac{z_i}{|\mathbf{x} - \mathbf{x}_i|} \tag{13}$$

is the exact solution of the Poisson equation

$$-\nabla \cdot (\epsilon_m \nabla u(\mathbf{x})) = C \sum_{i=1}^{N} z_i \delta(\mathbf{x} - \mathbf{x}_i), \mathbf{x} \in \mathbb{R}^3 \text{ with } u(\infty) = 0. \tag{14}$$

Note that G is C^∞ everywhere on $\mathbb{R}^3$, except at the finite number of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \Omega_1$. In particular, the term $\kappa^2(\mathbf{x})G(\mathbf{x})$ is well-defined everywhere, since $\kappa^2(\mathbf{x}) = 0$ for $\mathbf{x} \in \Omega_1$. We now follow the idea of[6] and decompose the potential according to

$$\phi = \begin{cases} G + w + u, & \text{in } \Omega_1, \\ u, & \text{in } \Omega_2 \cup \Omega_3. \end{cases} \tag{15}$$

Here, w denotes the solution of the Laplace equation

$$-\Delta w(\mathbf{x}) = 0, \qquad \mathbf{x} \in \Omega_1, \tag{16}$$

$$w(\mathbf{x}) = -G(\mathbf{x}), \quad \mathbf{x} \in \Gamma_{12}, \tag{17}$$

and can thus be considered as a harmonic extension of the values of G at Γ_{12} into Ω_1. Plugging the full potential ϕ as stated in (15) into the linear Poisson-Boltzmann equation (6) and taking into account the interface and boundary conditions of the section above, we obtain the following system for the regular potential u:

$$-\nabla \cdot (\epsilon(\mathbf{x})\nabla u(\mathbf{x})) + \kappa^2 u(\mathbf{x}) = 0, \qquad \mathbf{x} \in \Omega, \tag{18}$$

$$u_1(\mathbf{x}) = u_2(\mathbf{x}), \qquad \mathbf{x} \in \Gamma_{12}, \tag{19}$$

$$u_2(\mathbf{x}) = u_3(\mathbf{x}), \qquad \mathbf{x} \in \Gamma_{23}, \tag{20}$$

$$(\epsilon_m \nabla u_1(\mathbf{x}) - \epsilon_s \nabla u_2(\mathbf{x})) \cdot \mathbf{n}_{12}(\mathbf{x}) = -\epsilon_m \nabla (G(\mathbf{x}) + w(\mathbf{x})) \cdot \mathbf{n}_{12}(\mathbf{x}), \mathbf{x} \in \Gamma_{12}, \tag{21}$$

$$\nabla u_2(\mathbf{x}) \cdot \mathbf{n}_{23}(\mathbf{x}) = \nabla u_3(\mathbf{x}) \cdot \mathbf{n}_{23}(\mathbf{x}), \qquad \mathbf{x} \in \Gamma_{23}, \tag{22}$$

$$u(\mathbf{x}) = \phi_D(\mathbf{x}), \qquad \mathbf{x} \in \Gamma_D, \tag{23}$$

where, as before, $u_i := u|_{\Omega_i}$. As approximate Dirichlet boundary conditions, we use[7]

$$\phi_D(\mathbf{x}) := \frac{C}{4\pi\epsilon_s} \sum_{i=1}^{N} z_i \frac{e^{-\kappa(|\mathbf{x}-\mathbf{x}_i| - r_i)}}{(1 + \kappa r_i)|\mathbf{x} - \mathbf{x}_i|}, \tag{24}$$

with atom radii r_i. This choice corresponds to the superposition of the analytical solutions of the linear PBE on N non-interacting spheres with point charges. Summing up the previous steps, the solution of the linear PBE (6) is split into two parts: solution of the harmonic equation (HE) (16), (17) and solution of the regularized PBE (rPBE) (18) - (23).

2.4 Weak formulation

To solve a problem using finite element methods, a variational formulation of the problem must be given. It can be derived by multiplying the equation with some test functions, integrating over the domain, and applying integration by parts and Gauss' theorem. Therefore the domain Ω has to be a Lipschitz domain [10, p.89-96]. As to the weak formulation of (rPBE), let $u\colon \Omega \to \mathbb{R}$ denote a smooth solution of (18) - (23) and let $v\colon \Omega \to \mathbb{R}$

be smooth as well with $v(\mathbf{x}) = 0$ for $\mathbf{x} \in \Gamma_D$. Multiplying (18) by v, integrating over Ω, applying integration by parts and using the interface conditions (21), (22) (skipping the argument $\mathbf{x}$) yields

$$\begin{aligned}
0 &= -\int_\Omega \nabla \cdot (\epsilon \nabla u) v + \kappa^2 uv \\
&= -\int_{\Omega_1} \nabla \cdot (\epsilon_m \nabla u) v - \int_{\Omega_2} \nabla \cdot (\epsilon_s \nabla u) v - \int_{\Omega_3} \nabla \cdot (\epsilon_s \nabla u) v + \int_{\Omega_3} \kappa^2 uv \\
&= \int_{\Omega_1} \epsilon_m \nabla u \cdot \nabla v + \int_{\Omega_2} \epsilon_s \nabla u \cdot \nabla v + \int_{\Omega_3} \epsilon_s \nabla u \cdot \nabla v + \int_{\Omega_3} \kappa^2 uv \\
&\quad + \int_{\Gamma_{12}} -\epsilon_m (\nabla u_1 \cdot \mathbf{n}_{12}) v - \epsilon_s (\nabla u_2 \cdot (-\mathbf{n}_{12})) v + \int_{\Gamma_{23}} -\epsilon_s (\nabla u_2 \cdot \mathbf{n}_{23}) v - \epsilon_s (\nabla u_3 \cdot (-\mathbf{n}_{23})) v \\
&= \int_\Omega \epsilon \nabla u \cdot \nabla v + \int_\Omega \kappa^2 uv + \int_{\Gamma_{12}} \epsilon_m \nabla (G + w) \cdot \mathbf{n}_{12} v \\
&=: a(u, v) + c(u, v) - l(w; v),
\end{aligned}$$

with bilinear forms a, c and linear form $l(w; \cdot)$. Let $H^1(\Omega)$ denote the usual Sobolev space of $L^2(\Omega)$ functions whose distributional derivative is in $L^2(\Omega)$ as well and $H^1_0(\Omega) := \{v \in H^1(\Omega) \colon\ u|_{\Gamma_D} = 0\}$. Then, the weak formulation that corresponds to (rPBE) is defined by:

Find $u \in H^1(\Omega)$ with $u|_{\Gamma_D} = \phi_D$ such that

$$a(u, v) + c(u, v) = l(w; v) \text{ for all } v \in H^1_0(\Omega). \tag{25}$$

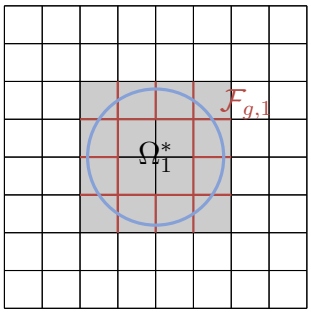
On the other hand, the weak formulation for (HE) is derived analogously:

Find $w \in H^1(\Omega_1)$ with $w|_{\Gamma_{12}} = -G$ such that

$$0 = b(w, v) := \int_{\Omega_1} \nabla w \cdot \nabla v \text{ for all } v \in H^1_0(\Omega_1). \tag{26}$$

2.5 Numerical method

The weak formulation of rPBE (25) and of HE (26) are both discretized through the CutFEM approach proposed in.[2] This method is based on the idea of imposing the interface conditions in a weak sense by using Nitsche's principle. In doing so, it is possible to maintain the full convergence potential provided by the underlying polynomial spaces, without the need to fit the computational mesh to the interface Γ_{12}. This is made possible by additional terms in the discrete variational formulation. In this way, the task of dealing with interface conditions is shifted from the mesh generation towards the computation of the underlying algebraic linear system.



To simplify the presentation, we ignore the solvent domain Ω_3 at this point and only consider the split $\Omega = \Omega_1 \cup \Omega_2$, i.e. $\Omega_3 = \emptyset$, $\kappa = 0$ and $c(u, v) = 0$. We further assume that Γ_{12} is known and sufficiently regular, such that both Ω_1 and Ω_2 are Lipschitz domains. Let $\mathcal{T}$ denote a triangulation (in the following called mesh) of the complete domain Ω, i.e. $\bar{\Omega} = \bigcup_{K \in \mathcal{T}} K$, consisting of hexahedral cells $K \subset \mathbb{R}^3$. In an upcoming section, $\mathcal{T}$ will be described in more detail. For $K \in \mathcal{T}$ we define $h_K := \text{diam}(K)$, $K_i := K \cap \Omega_i$, $i = 1, 2$ and $\Gamma_K := K \cap \Gamma_{12}$. We denote by $\mathcal{T}_i := \{K \in \mathcal{T} \colon\ K_i \neq \emptyset\}$ an overlapping triangulation of Ω_i, by $\mathcal{T}_\Gamma := \{K \in \mathcal{T} \colon\ \Gamma_K \neq \emptyset\}$ a covering of Γ_{12} and for some $\delta > 0$ we define $\mathcal{T}_g := \{K \in \mathcal{T} \colon\ \text{dist}(\Gamma_{12}, K) < \delta\}$ as collection of cells in the vicinity of Γ_{12}. Here, $\text{dist}(\Gamma_{12}, K) := \inf_{\mathbf{x} \in \Gamma_{12}} |\mathbf{x} - \mathbf{x}_K|$ denotes the distance between Γ_{12} and the midpoint $\mathbf{x}_K$ of K. For a given facet e, let $\mathbf{n}_e$ denote an arbitrary, but fixed unit normal vector. If e is a boundary facet, $\mathbf{n}_e$ should coincide with the unit outward normal vector of $\partial\Omega$. If e is an interior facet, let $e = K_e^+ \cap K_e^-$ with $K_e^\pm \in \mathcal{T}$ such that the unit outward normal of K_e^+ coincides with $\mathbf{n}_e$.

Moreover, we denote by $\mathcal{F}_g := \{e \subset \partial K \text{ is facet of some cell } K \in \mathcal{T}_g\}$ the collection of all facets associated with cells in $\mathcal{T}_g$. This set is further divided into $\mathcal{F}_{g,1} := \{e = K^+ \cap K^- \in \mathcal{F}_g\colon\ K^+, K^- \notin \mathcal{T}_2 \setminus \mathcal{T}_g\}$ and $\mathcal{F}_{g,2} := \{e = K^+ \cap K^- \in \mathcal{F}_g\colon\ K^+, K^- \notin \mathcal{T}_1 \setminus \mathcal{T}_g\}$. We define by

$$\Omega_i^* := \bigcup_{K \in \mathcal{T}_i \cup \mathcal{T}_g} K \tag{27}$$

a domain that overlaps Ω_i, i.e. $\Omega_i \subset \Omega_i^*$. On this overlapping domain, we define the finite element spaces of cell-wise linear polynomials:

$$V_i := \{v \in H^1(\Omega_i^*)\colon\ v|_K \in \mathbb{Q}_1\}, \tag{28}$$

where $\mathbb{Q}_k := \{v(\mathbf{x}) = \sum_{i,j,l=0}^{k} \alpha_{ijl} x_1^i x_2^j x_3^l,\ \alpha_{ijl} \in \mathbb{R}\}$. Functions $v \in V_i$ are extended by zero on $\Omega \setminus \Omega_i^*$. The joint finite element space is then given by

$$V_h := V_1 \times V_2, \quad V_{h,0} := \{v \in V_h\colon\ v|_{\Gamma_D} = 0\}, \tag{29}$$

with elements $v = (v_1, v_2)$. Note that v may have two nonzero components on $\Omega_1^* \cap \Omega_2^*$. Occasionally, we will simply consider $v(\mathbf{x})$ has single-valued function by following the convention that $v(\mathbf{x}) = v_i(\mathbf{x})$ for $\mathbf{x} \in \Omega_i \setminus \Gamma_{12}$. For $\omega^+, \omega^- \in [0,1]$ with $\omega^+ + \omega^- = 1$, $\mathbf{x} \in \Gamma_{12}$ and a function $f\colon \Omega \to \mathbb{R}$ let $f^{\pm}(\mathbf{x}) := \lim_{h \to 0^+} f(x \pm h\mathbf{n}_{12}(\mathbf{x}))$, $\{\{f\}\}_\omega(\mathbf{x}) := (\omega^+ f^+(\mathbf{x}) + \omega^- f^-(\mathbf{x}))$ denote its weighted average, $\{\{f\}\}_{\bar{\omega}}(\mathbf{x}) := (\omega^- f^+(\mathbf{x}) + \omega^+ f^-(\mathbf{x}))$ its skew weighted average and $[\![f]\!](\mathbf{x}) := f^-(\mathbf{x}) - f^+(\mathbf{x})$ its jump across Γ_{12}. For a finite element function $v_h \in V_h$ we set $v_h^+(\mathbf{x}) := v_2(\mathbf{x})$ and $v_h^-(\mathbf{x}) := v_1(\mathbf{x})$. For the sake of clarity we outline the ideas presented in[2] for a Poisson interface problem and derive a suitable discrete formulation of (25) in the CutFEM framework. Our goal is the definition of a discrete problem of the form: find $u_h \in V_h$ with $u_h|_{\Gamma_D} = \phi_D$ such that

$$a_h(u_h, v_h) = l_h(w; v_h) \text{ for all } v_h \in V_h. \tag{30}$$

The discrete bilinear form a_h and linear form l_h should be defined such that

1. the discrete problem is consistent, i.e. a sufficiently regular, exact solution u of (18) - (23) also satisfies (30)
2. the discrete problem is well-posed, i.e. it exhibits a unique solution whose norm can be bounded in terms of the input data.

We start with slightly modified terms $\tilde{a}$, $\tilde{l}$ that are close to the ones used in (25) and which take into account that the test functions are now double-valued. We further assume that u denotes a sufficiently regular exact solution of (18) - (23) and plug this function into

$$\sum_{i=1}^{2} \int_{\Omega_i} \epsilon \nabla u \cdot \nabla v_{h,i} =: \tilde{a}(u, v_h) = \tilde{l}(w; v_h) := -\int_{\Gamma_{12}} \epsilon_m \nabla(G + w) \cdot \mathbf{n}_{12} \{\{v_h\}\}_{\bar{\omega}}, \tag{31}$$

for all $v_h \in V_{h,0}$.
Applying once again integration by parts we obtain:

$$\begin{aligned}
-\int_{\Gamma_{12}} \epsilon_m \nabla(G + w) \cdot \mathbf{n}_{12} \{\{v_h\}\}_{\bar{\omega}} &= \sum_{i=1}^{2} \int_{\Omega_i} \epsilon \nabla u \cdot \nabla v_{h,i} \\
&= -\sum_{i=1}^{2} \int_{\Omega_i} \nabla \cdot (\epsilon \nabla u) v_{h,i} \\
&\quad + \int_{\Gamma_{12}} \epsilon_m (\nabla u \cdot \mathbf{n}_{12}) v_{h,1} - \int_{\Gamma_{12}} \epsilon_s (\nabla u \cdot \mathbf{n}_{12}) v_{h,2} \\
&= \int_{\Gamma_{12}} [\![\epsilon (\nabla u \cdot \mathbf{n}_{12}) v_h]\!] \\
&= \int_{\Gamma_{12}} \{\{\epsilon (\nabla u \cdot \mathbf{n}_{12})\}\}_\omega [\![v_h]\!] + \int_{\Gamma_{12}} [\![\epsilon (\nabla u \cdot \mathbf{n}_{12})]\!] \{\{v_h\}\}_{\bar{\omega}}.
\end{aligned}$$

Using (21), the above identity only holds for $\{\{\epsilon(\nabla u \cdot \mathbf{n}_{12})\}\}_\omega = 0$, which however, can not be inferred from the definition of u. Thus, (31) is not consistent. A straightforward modification of $\tilde{a}$ that yields a consistent formulation is given by simply subtracting the problematic term:

$$a_c(u, v_h) := \tilde{a}(u, v_h) - \int_{\Gamma_{12}} \{\{\epsilon(\nabla u \cdot \mathbf{n}_{12})\}\}_\omega [\![v_h]\!]. \tag{32}$$

Repeating the same steps as above shows that

$$a_c(u, v_h) = l(w; v_h) \text{ for all } v_h \in V_{h,0}. \tag{33}$$

The bilinear form a_c is not symmetric, unlike a. For our (potentially discontinuous) discrete solution u_h, we, therefore, subtract an additional term to define a symmetric form a_s by

$$a_s(u_h, v_h) := a_c(u_h, v_h) - \int_{\Gamma_{12}} \{\{\epsilon(\nabla v_h \cdot \mathbf{n}_{12})\}\}_\omega [\![u_h]\!]. \tag{34}$$

To fulfill the continuity condition (19) on Γ_{12} of we penalize jumps with another term:

$$a_p(u_h, v_h) := a_s(u_h, v_h) + \gamma_{IP} \sum_{K \in \mathcal{T}_\Gamma} \frac{2\epsilon_m \epsilon_s}{\epsilon_m + \epsilon_s} h_K^{-1} \int_{\Gamma_K} [\![u_h]\!][\![v_h]\!]. \tag{35}$$

Finally, to ensure coercivity of the bilinear form throughout the computational domain, we add the so-called *ghost penalty term* j given by

$$j(u_h, v_h) := \gamma_g \sum_{i=1}^{2} \sum_{e \in \mathcal{F}_{g,i}} h_{K_e^+} (\epsilon_m \delta_{i1} + \epsilon_s \delta_{i2}) \int_e ([\![\nabla u_{h,i}]\!] \cdot \mathbf{n}_e)([\![\nabla v_{h,i}]\!] \cdot \mathbf{n}_e) \tag{36}$$

to arrive at

$$a_g(u_h, v_h) := a_p(u_h, v_h) + j(u_h, v_h). \tag{37}$$

Now assuming $\Omega_3 \neq \emptyset$, $\kappa \neq 0$ and defining $\tilde{V}_2 := V_2 \cap H^1(\Omega_2 \cup \Omega_3)$ and $\tilde{V}_h := V_1 \times \tilde{V}_2$, the CutFEM weak formulation reads:

Find $u_h \in \tilde{V}_h$ with $u_h|_{\Gamma_D} = \phi_D$ such that

$$a_h(u_h, v_h) = l_h(w; v_h) := l(w; v_h) \text{ for all } v_h \in \tilde{V}_{h,0}, \tag{38}$$

with

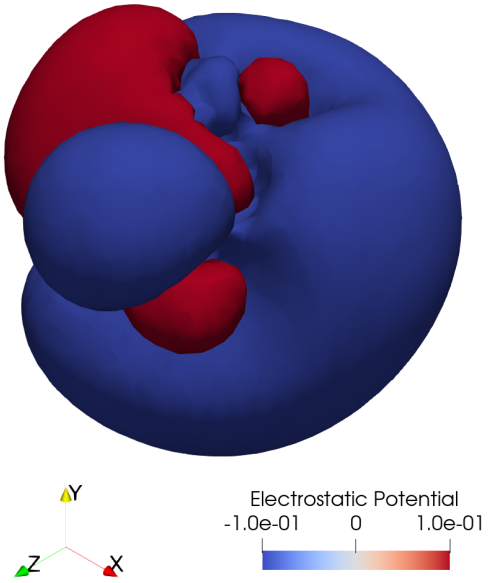
$$a_h(u_h, v_h) := a_g(u_h, v_h) + \int_{\Omega_3} \epsilon \nabla u_h \cdot \nabla v_{h,2} + c(u_h, v_h). \tag{39}$$

3 Numerical Example

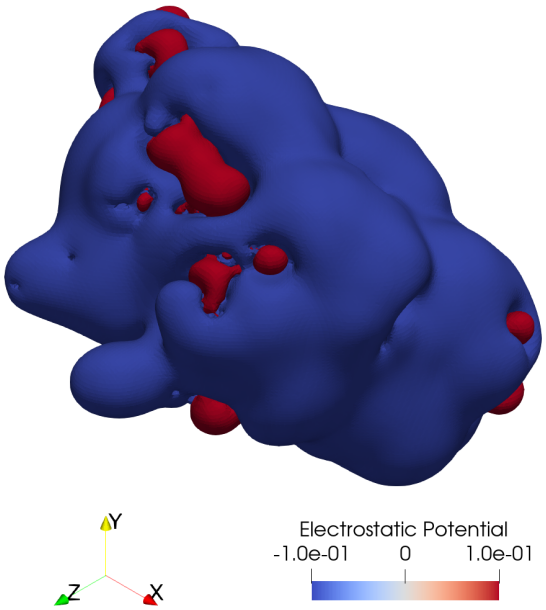
As a numerical example, the electrostatic potentials of the ubiquitin (PDB ID: 1UBQ) and the adenovirus virion (PDB ID: 4CWU) molecules are calculated. The parameters used for this can be seen in the parameter file shown in 4.3 for 1UBQ and in the file `pbe_tutorial_4CWU.xml` for 4CWU. The SI (international standard) units are used in this simulations[1] and in the parameter files. The main physical constants in our problem are given as

Abbr	Value in SI	Physical Constant
e_c	$1.6021476634 \times 10^{-19}\mathrm{C}$	Elementary charge
k_B	$1.380649 \times 10^{-23}\ \mathrm{J/K}$	Boltzmann constant
ε_0	$8.854187812813 \times 10^{-12}\ \mathrm{F/m}$	Vacuum permittivity
N_A	$6.02214076 \times 10^{23} 1/\mathrm{mol}$	Avogadro's number

To solve the linear system GMRES with the ILU++ preconditioner[8] is used. For the structured mesh, the performance of this combination is not optimal. However, an easily accessible preconditioner is chosen in HiFlow[3] to keep this tutorial simple.
The computational effort varies depending on the mesh refinement, the size of the molecule and the number of atoms. For the smaller `1UBQ` molecule a coarser mesh is used resulting in 173806 dofs and for the larger `4CWU` a mesh resulting in 11273130 dofs was used to obtain the results shown in Fig. 2.



(a) Ubiquitin



(b) Adenovirus virion

Figure 2: Electrostatic potential map of u defined in (3) of ubiquitin and adenovirus virion computed with this tutorial and visualized with ParaView[5] with contour lines at ± 0.1.

4 Implementation in Hiflow^3

4.1 How to run the code?

You find the example code (pbe_tutorial.cc, pbe_tutorial.h), a parameter file for the first numerical example (pbe_tutorial.xml) and the molecule data (*.pqr) in the folder
`/hiflow/examples/pbe_tutorial`.

4.1.1 Using HiFlow^3 as a Developer

First, build and compile HiFlow^3 (an installation guide can be found in the `readme.md` file in the `pbe_tutorial` folder). Go to the directory `/build/examples/pbe_tutorial`, where the binary **pbe_tutorial** is stored. Type **./pbe_tutorial pbe_tutorial_1UBQ.xml**, to execute the program in sequential mode for the ubiquitin molecule. To execute in parallel mode with four processes, type **mpirun -np 4 ./pbe_tutorial pbe_tutorial_1UBQ.xml**. In both cases, you need to make sure that the parameter file `.xml` and the molecule data `.pqr` are stored in the same directory as the binary. You can specify the path of your own XML-file with the name of your XML-file and the path, i.e.
./pbe_tutorial `/"path_to_parameterfile"/"name_of_parameterfile".xml`.

4.2 Preliminaries

HiFlow^3 is designed for high-performance computing on massively parallel machines. So it applies the Message Passing Interface (MPI) library specification for message-passing. For the illustrative purpose of this tutorial, not all functionalities of HiFlow^3 are used. Adaptive mesh refinement and parallel partitioning are therefore not used. The example needs to be compiled with p4est.[3]
The following two input files are needed for the PBE tutorial:

- A parameter file: The parameter file is an XML file, which contains all parameters needed to execute the program. It is read in by the program, for example, parameters that define the molecule as well as parameters needed for the linear solver and linear algebra. It is not necessary to recompile the program when parameters in the XML file are changed.
- Molecule information: The file containing the information of the molecule specified in the parameter file. A few molecule files can already be found in the pbe_tutorial example folder.

4.3 Parameter File

The needed parameters are initialized in the parameter file (`.xml`). The name of the molecule is set by the parameter `<Molecule><Name>`. Further molecular parameters listed at the beginning of the `xml` file are the temperature in Kelvin, the ionic strength I_s defined by `<IoStrength>`, the solvent dielectric of the solvent material (for example dielectric of water) defined by `<SolventDiel>` and the solute dielectric dependent on the input molecule and temperature given by `<SoluteDiel>`. The dielectric constants are dimensionless numbers while the ionic strength is given in molar.
It is important that the parameters `<Mesh><XMin>`, `<Mesh><XMax>`, `<Mesh><YMin>`, `<Mesh><YMax>`, `<Mesh><ZMin>` and `<Mesh><ZMax>` indicating the bounding box for the solvent region are chosen large enough depending on the molecule for the linear solver to converge. The following shows the parameter file for the ubiquitin molecule.

```
  '
<Param>
  <Molecule>
    <Name>1UBQ</Name>
    <IoStrength>0.1</IoStrength>
    <Temperature>298.15</Temperature>
    <SolventDiel>78.54</SolventDiel>
    <SoluteDiel>2.0</SoluteDiel>
    <MaxNumAtoms>-1</MaxNumAtoms>
  </Molecule>
  <General>
    <FeDegree>1</FeDegree>
    <LSFeDegree>1</LSFeDegree>
    <QuadratureOrder>4</QuadratureOrder>
  </General>
  <SingularPotential>
    <QuadPointEvalType>1</QuadPointEvalType>
```

```
  <PostProcessingEvalType>0</PostProcessingEvalType>
  <DirichletEvalType>0</DirichletEvalType>
  <CropDistance>10</CropDistance>
</SingularPotential>
<Mesh>
  <SESGrid>
    <RefLevel>3</RefLevel>
    <RelAcceptThresh>2.</RelAcceptThresh>
    <PolynomialDegree>1</PolynomialDegree>
    <RemoveCavities>0</RemoveCavities>
  </SESGrid>
  <CellWidth>4.0</CellWidth>
  <RefLevel>0</RefLevel>
  <XMin>-12</XMin>
  <XMax>72</XMax>
  <YMin>-12</YMin>
  <YMax>72</YMax>
  <ZMin>-28</ZMin>
  <ZMax>64</ZMax>
  <RefLevel>1</RefLevel>
</Mesh>
<Logging>
  <Visu_SubtractSingular>0</Visu_SubtractSingular>
  <VisuIntervals>1</VisuIntervals>
</Logging>
<CutFEM>
  <Extension>
    <SIPG_Sym>1.</SIPG_Sym>
    <SIPG_Pen>1e2</SIPG_Pen>
    <Ghost_Pen>1e-0</Ghost_Pen>
    <Ghost_Type>2</Ghost_Type>
  </Extension>
  <PB>
    <SIPG_Sym>1.</SIPG_Sym>
    <SIPG_Pen>1e1</SIPG_Pen>
    <Ghost_Pen>1e-1</Ghost_Pen>
    <Ghost_Type>2</Ghost_Type>
  </PB>
  <Overlap>
    <Type>2</Type>
    <AbsWidth>1.</AbsWidth>
    <RelWidth>1.</RelWidth>
    <UseRelWidth>1</UseRelWidth>
    <UseOverlapOnFineMeshOnly>0</UseOverlapOnFineMeshOnly>
  </Overlap>
  <Quadrature>
    <IfaceOrder>3</IfaceOrder>
    <SubcellOrder>3</SubcellOrder>
  </Quadrature>
</CutFEM>
<LinearAlgebra>
  <NameMatrix>CoupledMatrix</NameMatrix>
  <NameVector>CoupledVector</NameVector>
  <Platform>CPU</Platform>
```

```
    <MatrixFormat>CSR</MatrixFormat>
    <SparseCompression>1</SparseCompression>
    <Implementation>Naive</Implementation>
 </LinearAlgebra>
  <LinearSolver>
    <Name>LINSOLVER</Name>
    <Type>GMRES</Type>
    <MaxIt>2000</MaxIt>
    <AbsTol>1.e-14</AbsTol>
    <RelTol>1.e-8</RelTol>
    <DivLimit>1.e6</DivLimit>
    <KrylovSize>200</KrylovSize>
    <UsePrecond>0</UsePrecond>
    <PrintLevel>1</PrintLevel>
  </LinearSolver>
</Param>
```

4.4 Important member functions

The main function starts the simulation of the PBE problem (`pbe_tutorial.cc`).

4.4.1 run()

The member function run() executes the PBE tutorial. The function is defined in the class PoissonBoltzmann (`pbe_tutorial.cc`). The first operation `read_parameters()` reads the simulation parameters from the `xml` file. The `build_initial_mesh()` and `prepare_system()` functions construct the computational mesh and initialize the global finite element space. Afterwards, the linear algebra and the level set system are prepared in the `prepare_LA()` and `prepare_LS_system()` functions. Then, the local assemblers for the (HE) and (rPBE) problem are prepared in `prepare_local_assembler()`. The interface mesh is prepared and boundary conditions are assembled in the following functions `initialize_SES_mesh()` and `prepare_interface_and_boundary_conditions()`. Finally the two systems are assembled in `assemble_extension` and `assemble_linear_cut()` and solved in `solve_system` to be visualized with `visualize_LS()` and `visualize()`.

```
   ,
/**
 * Runs the Poisson-Boltzmann simulation.
 */
void PoissonBoltzmann::run()
{

    Timer timer_complete;
    timer_complete.start();

    LOG_INFO("XML file", SOLVER_PARAM_FILENAME);
    parcom_.barrier();

    // Read parameters from XML file
    this->read_parameters();
    // Read molecule configuration
    this->read_molecule_configuration();
    // Build initial mesh
    this->build_initial_mesh();
    // Prepare spaces
```

```
    this->prepare_system();
    // Prepare linear algebra
    this->prepare_LA();
    // Prepare level set system
    this->prepare_LS_system();
    // Prepare local assemblers
    this->prepare_local_assemblers();
    // Initialize SES mesh
    this->initialize_SES_mesh();
    // Prepare interface and boundary conditions
    this->prepare_interface_and_boundary_conditions();
    // Visualize level set functions
    this->visualize_LS();

    // compute harmonic extension
    this->assemble_extension();
    this->solve_system(ext_solver_, ext_rhs_, ext_sol_,
                               this->ext_matrices_,
                               this->fixed_ext_dofs_, this->fixed_ext_values_);

    // Compute CutFEM solution
    this->assemble_linear_cut();
    this->solve_system(cut_solver_,
                               *rhs_, *sol_,
                               this->matrices_,
                               this->fixed_cut_dofs_, this->fixed_cut_values_);

    parcom_.barrier();
    // Visualize solution
    this->visualize();

    MPI_Barrier(this->comm_);
    timer_complete.stop();
    LOG_INFO("", "=========================================");
    LOG_INFO("", "=========================================");
    LOG_INFO("Time", "COMPLETE : " << timer_complete.get_duration() << " sec");
    LOG_INFO("", "=========================================");
    LOG_INFO("", "=========================================");
}
```

4.4.2 solve_system

The member function solve_system is called twice for solving the (HE) and the (rPBE) problem. Therefore, the respective linear problem is passed to the solve_system function.

```
    '
void PoissonBoltzmann::solve_system(std::shared_ptr<LinearSolver<LAD>> &outer_solver,
                                    CVector &rhs,
                                    CVector &sol,
                                    typename LAD::MatrixType &op,
                                    const std::vector<int> &fixed_dofs,
                                    const std::vector<DataType> &fixed_vals)
{
    sol.Zeros();
```

```
    if (!fixed_dofs.empty())
    {
        sol.SetValues(vec2ptr(fixed_dofs),
                      fixed_dofs.size(),
                      vec2ptr(fixed_vals));
    }
    sol.Update();

    Timer setup_timer;
    setup_timer.start();
    outer_solver->SetupOperator(op);

    PreconditionerBlockJacobiExt<LAD> precond_;
    precond_.Init_ILU_pp();
    precond_.SetupOperator(op);
    precond_.Build();
    outer_solver->SetupPreconditioner(precond_);

    setup_timer.stop();

    Timer solve_timer;
    solve_timer.start();

    rhs.Update();
    outer_solver->Solve(rhs, &sol);
    sol.Update();

    solve_timer.stop();

    DataType norm_rhs = rhs.Norm2();
    std::string solver_name = solver_params_["LinearSolver"]["Type"].get<std::string>();
    LOG_INFO(solver_name, "#Iter␣␣␣␣␣␣␣␣␣␣␣␣␣␣" << outer_solver->iter());
    LOG_INFO(solver_name, "Abs␣Residual␣␣␣␣␣␣␣" << outer_solver->res());
    LOG_INFO(solver_name, "Rel␣Residual␣␣␣␣␣␣␣" << outer_solver->res() / norm_rhs);
    LOG_INFO(solver_name, "Time␣setup␣␣␣␣␣␣␣␣" << setup_timer.get_duration());
    LOG_INFO(solver_name, "Time␣solution␣␣␣␣␣␣" << solve_timer.get_duration());
}
```

5 Conclusion

We have demonstrated the feasibility of solving the linear Poisson-Boltzmann equation numerically on a three-dimensional, uniform, Cartesian mesh with the CutFEM approach. Since CutFEM allows us to work with simple, non-matching grids, we could easily compute the electrostatic potential of various molecules. We also showed how the HiFlow^3 library could be practically utilized to discretize, solve and post-process the results of the PBE. Future work will concentrate on benchmarking our implementation under various metrics. Thereafter, we intend use the benchmarks to indicate areas of potential improvement in terms of both solution accuracy and computational efficiency.

References

[1] Bureau International des Poids et Mesures. The international system of units. https://www.bipm.org/utils/common/pdf/si-brochure/SI-Brochure-9.pdf, 2019. [Online; accessed 05-June-2024].

[2] Erik Burman, Susanne Claus, Peter Hansbo, Mats G. Larson, and André Massing. Cutfem: Discretizing geometry and partial differential equations. *International Journal for Numerical Methods in Engineering*, 104(7):472–501, 2015.

[3] Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas. p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.

[4] Simon Gawlok, Philipp Gerstner, Saskia Haupt, Vincent Heuveline, Jonas Kratzke, Philipp Lösel, Katrin Mang, Mareike Schmidtobreick, Nicolai Schoch, Nils Schween, Jonathan Schwegler, Chen Song, and Martin Wlotzka. HiFlow3 – Technical Report on Release 2.0. *EMCL-Preprints.*, (06), November 2017.

[5] Amy Henderson Squillacote. *The ParaView guide: a parallel visualization application*. Kitware, 2007.

[6] Michael Holst. The Poisson-Boltzmann Equation. https://ccom.ucsd.edu/~mholst/pubs/dist/Hols94d.pdf, 1994. [Online; accessed 05-June-2024].

[7] Cleophas Muganda Kweyu. *Fast solution of the Poisson-Boltzmann equation by the reduced basis method and range-separated canonical tensor format*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, 2019.

[8] Jan Mayer. ILU++: A new software package for solving sparse linear systems with iterative methods. *Proc. Appl. Math. Mech.*, 7(1):2020123–2020124, December 2007.

[9] Michel F. Sanner, Arthur J. Olson, and Jean-Claude Spehner. Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers*, 38(3):305–320, 1996.

[10] W.McLean. *Strongly Elliptic Systems and Boundary Integral Equations*. Cambridge University Press, 2000.

A Appendix

A.1 Program Output

HiFlow^3 can be executed in a parallel or sequential mode which influences the generated output data. Executing the program in parallel, for example with four processes by **mpirun -np 4 ./pbe_tutorial** generates the following output data in the `/hiflow/examples/pbe_tutorial/visu/` folder.

- Mesh data:
  - **pbe_tutorial_mesh.pvtu** Uniformly refined mesh in the parallel vtk-format. It combines the local mesh data owned by the different processes to a global mesh.
  - **pbe_tutorial_mesh_X.vtu** Local mesh (vtk-format).
- Level set function data:
  - **pbe_tutorial_LS_MoleculeName.pvtu** Level set function *Xi* for representing the interface computed for the CutFEM approach.
  - **pbe_tutorial_LS_MoleculeName_X.vtu** Local solution of the level set function which belongs to cells owned by process X, for X=$0, \ldots,$ *processes* (vtk-format).
- Solution data:
  - **pbe_tutorial_sol_MoleculeName_.pvtu** Solution of electrostatic potential u. Furthermore, solution data of the harmonic extension u_{ext}, solution data inside the and the regular part of the solution u_{reg} is contained in this file. It combines the local solutions owned by the different processes to a global solution.
  - **pbe_tutorial_sol_MoleculeName_X.vtu** Local solution of the electric potential of the degrees of freedoms which belong to cells owned by process X, for X=0, 1, 2, and 3 (vtk-format).
- Log files:
  - **pbe_tutorial_debug.log** Log file listing errors helping to simplify the debugging process. This file is empty if the program runs without errors.
  - **pbe_tutorial_info.log** Log file listing parameters and some helpful information to control the program such as about the residual of the linear and non-linear solver used.

A.2 Visualizing the Solution

HiFlow^3 only generates output data but does not visualize. The mesh data as well as the solution data can be visualized by any external program that can handle the vtk data format e.g. the program ParaView[5] by opening the `.pvtu` files. A good way to visualize the electrical potential in this problem is with a contour plot showing potential isosurfaces.

Preprint Series of the Engineering Mathematics and Computing Lab

recent issues

No. 2023-02 Ayse Erozan, Philipp Lösel, Vincent Heuveline, Venera Weinhardt : ACSeg: Automated 3D Cytoplasm Segmentation in Soft X-Ray Tomography

No. 2023-01 Marco Schröder, Stefan Machmeier, Vincent Heuveline: Vtable hijacking: Object Type Integrity for run-time type information

No. 2021-02 Alejandra Jayme, Philipp D. Lösel, Joachim Fischer, Vincent Heuveline: Comparison of Machine Learning Methods for Predicting Employee Absences

No. 2021-01 Chen Song, Jonas Roller, Ana Victoria Ponce-Bobadilla, Nicolas Palacio-Escat,Julio Saez-Rodriguez, Vincent Heuveline: Spatial Effect on Separatrix of Two-Cell Systemand Parameter Sensitivity Analysis

No. 2020-01 Saskia Haupt, Nassim Fard-Rutherford, Philipp D. Lösel, Lars Grenacher, Arianeb Mehrabi, Vincent Heuveline: Mathematical Clustering Based on CrossSections in Medicine: Application to the Pancreatic Neck

No. 2019-02 Nils Schween, Nico Meyer-H übner, Philipp Gerstner, Vincent Heuveline: A time stepreduction method for Multi-Period Optimal Power Flow problems

No. 2019-01 Philipp Gerstner, Martin Baumann, Vincent Heuveline: Analysis of the StationaryThermal-Electro Hydrodynamic Boussinesq Equations

No. 2018-02 Simon Gawlok, Vincent Heuveline: Nested Schur-Complement Solver for a Low-MachNumber Model: Application to a Cyclone-Cyclone Interaction

No. 2018-01 David John, Michael Schick, Vincent Heuveline: Learning model discrepancy of anelectric motor with Bayesian inference

No. 2017-06 Simon Gawlok, Philipp Gerstner, Saskia Haupt, Vincent Heuveline, Jonas Kratzke, Philipp Lösel, Katrin Mang, Maraike Schmidtobreick, Nicolai Schoch, Nils Schween, Jonathan Schwegler, Chen Song, Marin Wlotzka: HiFlow3 Technical Report on Release 2.0

No. 2017-05 Nicolai Schoch, Vincent Heuveline: Towards an Intelligent Framework for Personalized Simulation-enhanced Surgery Assistance: Linking a Simulation Ontology to a Reinforcement Learning Algorithm for Calibration of Numerical Simulations

No. 2017-04 Martin Wlotzka, Thierry Morel, Andrea Piacentini, Vincent Heuveline: New features for advanced dynamic parallel communication routines in OpenPALM: Algorithms and documentation

No. 2017-03 Martin Wlotzka, Vincent Heuveline: An energy-efficient parallel multigrid method for multi-core CPU platforms and HPC clusters

No. 2017-02 Thomas Loderer, Vincent Heuveline: New sparsing approach for real-time simulations of stiff models on electronic control units

No. 2017-01 Chen Song, Markus Stoll, Kristina Giske, Rolf Bendl, Vincent Heuveline: Sparse Gridsfor quantifying motion uncertainties in biomechanical models of radiotherapy patients

Preprint Series of the Engineering Mathematics and Computing Lab (EMCL)

www.emcl.iwr.uni-heidelberg.de