ELLIPTIC: THE EXTENSIBLE LIBRARY FOR PHYSICAL SIMULATIONS

Guilherme Praciano Karst Caminha

Ricardo Jorge Morais de Lira Filho

Ramiro Brito Willmersdorf

Darlan Karlo Elisiário de Carvalho

Paulo Roberto Maciel Lyra

gpkc@cin.ufpe.br

r.liraf@gmail.com

ramiro@willmersdorf.net

dkarlo@uol.com.br

prmlyra@padmec.org

CTG - UFPE

Av. Prof. Moraes Rego, 1235 – Cidade Universitária, 50740-560, Recife, Pernambuco, Brasil

Abstract. *ELLIPTIc (The Extensible Library for Physical simulaTIons) aims to be an enabling technology for the development of numerical methods based on compact and non-compact stencils, with structured, non-structured, conformal and non-conformal meshes, in one, two or three dimensions. Developed with the Python programming language, ELLIPTIc uses the high-performance libraries MOAB and Trilinos, built in C++ and Fortran, which offer data structures and algorithms for the storage and representation of meshes, sparse matrices, preconditioners and linear and non-linear solvers. Therefore, ELLIPTIc offers the ease of programming and advanced Software Engineering techniques that can be applied to the Python programming language, while using high-performance libraries, compensating potential bottlenecks that can arise while using a high-level and interpreted language such as Python. This library is, therefore, not only a tool for prototyping and testing of numerical methods but also usable in real and large-scale contexts. To provide qualitative results regarding the usability of the library, we present two simple problems and their solution using ELLIPTIc.*

Keywords: *Numerical Methods, High-Performance, Python, Enabling Technology*

Proceedings of the XXXVIII Iberian Latin-American Congress on Computational Methods in Engineering P.O. Faria, R.H. Lopez, L.F.F. Miguel, W.J.S. Gomes, M. Noronha (Editores), ABMEC, Florianópolis, SC, Brazil, November 5-8, 2017.

1 INTRODUCTION

In the development of new technology - and its application in many industries, computational simulations is of paramount importance for cost reduction, performance evaluation and some other advantages. Most of the commercial software for applications such as in product design, reservoir simulation, etc., rely on the discretization of physical phenomena that are studied through mathematical models using Partial Differential Equations (PDEs). PDEs are obtained in general from conservative laws of physics applied to some system which is being studied. Solving these equations, however, becomes a very complex task. In most cases, analytic solutions might not even be possible and therefore computational methods are the key for dealing with these problems.

Many existing PDE solver packages (LibMesh, FeNICs, DOLFIN, …) focus on the important task of numerically solving the linearized set of algebraic equations that results from discretizing a set of PDEs. Of the PDE solver packages that focus at an appropriately high level, many are commercial, expensive, and difficult to customize (Guyer, et al., 2009). One fit example is the MatLab® tool. Moreover, the complexity of many phenomena can imply in high computational costs due to the enormous amount of operations that a computer must process during simulation, demanding more and more high-performance computing (HPC) capabilities

The impact of HPC on the field of Computational Fluid Dynamics (CFD) has been felt in several ways. It has become possible to carry out much more complex simulations such as problems involving unsteadiness and multiphysics coupled with complex geometries.

For software improvement of performance, traditional object-oriented finite element libraries provide basic tools such as computational meshes, linear algebra interfaces, and finite element basis functions. This greatly simplifies the implementation of numerical methods, but the user must typically implement the assembly algorithm, or at least part of it (Logg & Wells, 2010).

The ELLIPTIc project was created to facilitate the prototyping, testing and development of modern numerical methods. It works by abstracting most of the functionalities that are desired when developing high performance physical simulation software, such as access to sparse matrices, iterative solvers, and mesh data structures.

A large and increasing number of other software for high-performance computing can be found in the literature, many of them first designed for some specific physical applications, but based on some kind of numerical method. FeniCS/DOLFIN, LibMesh and FiPy are among the projects that strongly inspired the development of ELLIPTic. We briefly describe these libraries below.

1.1 FEniCS/DOLFIN

The FEniCS software consists of a collection of interoperable components. FeniCS is composed of libraries that are part of the original problem as well as other parts that integrate the tool for completeness. Among these libraries, DOLFIN – a library based on finite elements – relies on a form compiler to generate the inner-most loops of the algorithm. This allows DOLFIN to implement a general and efficient assembly algorithm. Among its main features, DOLFIN may assemble arbitrary rank tensors, varying from scalars, vectors and matrices to

even higher-rank tensors, on simplex meshes in one, two, and three-dimensional spaces for a wide range of user-defined variational forms and for a wide range of finite elements. Furthermore, tensors may be assembled into any user-defined data structure, or any of the data structures implemented by one of the built-in linear algebra backends.

Initially, DOLFIN was a monolithic, stand-alone C++ library including implementations of linear algebra, computational meshes, finite element basis functions, variational forms, and finite element assembly. Since then, it has undergone a number of design iterations and some functionality has now been "outsourced" to other FEniCS components and third-party software. The design encompasses coexistence with other libraries, and permits a user to select particular components (classes) rather than to commit to a rigid framework or an entire package. DOLFIN supports a wider range of finite elements than any of the above-mentioned libraries since it may assemble any finite element variational form on any finite element space supported by the form compiler and finite element backend. For more on the FeniCS/DOLFIN project, read (Alnaes, et al., 2015) and (Logg & Wells, 2010).

1.2 libMesh

The libMesh - a standard cell-based discretization library - focuses on parallel, adaptive, multiscale, multiphysics finite element simulations. Mainly two key factors motivated the creation of libMesh: the existence of a relatively robust and rapidly-evolving parallel hardware–software infrastructure that included affordable parallel distributed clusters running Linux and high-performance implementations and the rapid evolution of adaptive mesh methodology, and algorithms for domain decomposition and efficient repartitioning. libMesh aids in providing a research platform for parallel adaptive algorithms which has been growing rapidly (Kirk, et al., 2006).

The main idea behind libMesh lies in centralizing physics-independent technology as well as leveraging existing libraries whenever possible, which makes the software development effort required to support parallel and adaptive unstructured mesh-based simulations less arduous. This enables users to only focus on the specifics of a given application (i.e. the physics itself) without considering the additional complexities of parallelizing the problem, distributing workload, etc. This is one of the main reasons why libMesh has proved to be a valuable test bed for many numerical problems. Originally it intended to provide resourceful data structure supporting many features, such as adaptive mesh refinement for arbitrary unstructured meshes, particularly focused on finite element and finite volume approaches. By breaking the adaptive meshing technology from the application script, reusing the code is evident, enabling, for instance, a large number of diverse applications which can leverage from the library.

The adaptive technology behind LibMesh utilizes element subdivision to locally refine the mesh and thereby resolve different scales such as boundary layers and interior shock layers (Carey, et al., 2004). Among the finite element formulations that one can find in the library are Galerkin, Petrov–Galerkin, and discontinuous Galerkin methods. Parallelism is achieved using domain decomposition through mesh partitioning, in which each processor contains the global mesh but in general computes only on a particular subset. Parallel implicit linear systems are supported via an interface with the PETSc library. Since AMR/C (Adaptive Mesh Refinement) is a dynamic process, efficient load balancing requires repartitioning in both steady-state and evolution problems. In evolution problems, in particular, coarsening is desirable and sometimes

essential for obtaining efficient solutions. For more information on libMesh and its functionalities refer to (Kirk, et al., 2006).

1.3 FiPy

Most of the described software are based on a Finite Element method formulation. However, some problems use the Finite Volume (FV) approach. FiPy is an object oriented, PDE solver, written in Python. The framework has been developed in the Materials Science and Engineering Division (MSED) and Center for Theoretical and Computational Materials Science (CTCMS), in the Material Measurement Laboratory (MML) at the National Institute of Standards and Technology (NIST) (Guyer, et al., 2009). Combining the FV method and Python provides a tool that is extensible, powerful and freely available. A significant advantage to Python is the existing suite of tools for array calculations, sparse matrices and data rendering, such as the libraries SciPy, Numpy, etc. FiPy framework supports convection-diffusion-reaction problems, or any combinations resulting in elliptic, hyperbolic and parabolic PDEs. FiPy has been utilized for treatments of polycrystalline, dendritic, and electrochemical phase transformations as well as a level set treatment of the electrodeposition process (Josell, et al., 2001).

1.4 Motivation

The decision to build a new package was made from the fact that there are currently no generic Python packages that are capable of running large-scale, distributed problems. The FEniCS software can be run in parallel, however, it is specific for Finite Elements problems. ELLIPTIc intendeds to be generic, and could be used to implement any mesh-based numerical methods.

Also, most current numerical packages and libraries can't perform computation on non-conformal and polygonal meshes. ELLIPTIc supports retrieving adjacency information on general meshes, which can be unstructured, polygonal and non-conforming.

2 PACKAGE OVERVIEW

ELLIPTIc is implemented as a Python package. Its source code can be found on GitHub (https://github.com/padmec-reservoir/ELLIPTIc).

The main focus of the project is to provide access to high-level, abstracted functionalities to allow the prototyping, testing and development of numerical schemes. It uses the low-level, high performance libraries MOAB (Mesh-Oriented datABase) (Tautges, et al., 2004) and Trillions (Heroux, et al., 2003), therefore allowing large-scale simulations.

2.1 Python and Scientific Computing

As can be stated from the previous section, there is a tendency towards using high-level languages such as Python for the development of scientific packages. There are several reasons for that. Below we list a few of those reasons, which were considered valuable for the decision-making process of which software stack should be used for ELLIPTIc:

- Python is free for use and has a remarkably thriving community;

- Python has a plethora of scientific packages that are widely supported, with cutting-edge algorithms and data structures for many applications, such as Artificial Intelligence and Data Science;
- Python allows for rapid prototyping of algorithms, while also allowing for production-level code;
- Python is very easy to maintain.

However, it is widely known that Python is a relatively slow language. Being an interpreted and high-level language, despite allowing for many of the above points to exist, makes it harder for general optimizations in its core to be done.

Nevertheless, Python is used in many resource demanding tasks, such as physical simulations. This is due to the fact that in modern days, computational power became extremely cheap when compared to the cost of hiring more engineers, since computers can easily be rented or purchased. Therefore, a language whose code is easier to use and maintain can potentially compensate possibly higher computing expenses, by reducing engineering costs. Also, Python can be used for the outer loops and the so-called "glue code", while a lower level language such as C++ can be used for the more critical parts of the code.

2.2 Dependencies

To achieve high-performance while also possessing a high-level API (Application Programming Interface) for the implementation of numerical schemes, ELLIPTIc makes use of two low-level libraries: MOAB and Trilinos. Below, we briefly discuss each library.

2.2.1 MOAB

MOAB (Mesh-Oriented datABase) is a library to represent and store mesh data (Tautges, et al., 2004). It allows for parallelism through MPI (Message Passing Interface), making it possible to distribute partitioned meshes on clusters with several computing nodes.

In ELLIPTIc, the MOAB library is used as a mesh processing and representation toolbox. Its main features being used include:

- Storing and retrieving meshes along with information for boundary conditions and physical properties through a native HDF5-based file format (The HDF Group, 1997-2017).
- Partitioning and mesh distribution.
- Easy retrieval of topological data, such as second order adjacencies.

MOAB is implemented in C++, C and Fortran, and is designed with large-scale parallelism and memory efficient traits. It also features a Python interface (PyMOAB), which is used on ELLIPTIc.

2.2.2 Trilinos

Trilinos is a group of packages for the solution of large-scale engineering and scientific problems mainly written in C++ (Heroux, et al., 2003). For the current purpose of ELLIPTIc,

two main classes of features of the Trilinos project are being used, namely its solvers and matrix / vector data structures.

Matrices and vectors are among the most basic data structures used on most physical simulations. Trilinos defines abstract interfaces for interacting with matrices and vectors. One of the most important concrete implementations of these interfaces is given through the Trilinos package Epetra, which is extensively used by ELLIPTIc. The Epetra package is a powerful collection of classes that supports vectors, dense matrices and sparse matrices.

For the solver part, Trilinos provides the Aztec package, with its object-oriented follow-on AztecOO, which provides an elegant interface and many functionality extensions. AztecOO offers preconditioned iterative solvers. As most Trilinos packages, AztecOO accepts as input Epetra vectors and matrices.

Trilinos also has a Python interface through the PyTrilinos package.

Despite of not yet being applicable on ELLIPTIc, it is worth mentioning the following Trilinos packages:

- Ifpack: Algebraic preconditioners.
- ML: Multi-level / multigrid preconditioners.

2.3 Scope

ELLIPTIc is originally intended to be an abstract framework that allows the development of numerical methods on arbitrary unstructured meshes, which might be non-conformal. Its main focus is on finite volume simulations. By adopting an abstract object-oriented and modular approach, ELLIPTIc allows for code reuse and extensibility.

The major goal of the package is to allow for stencil-oriented algorithms to be developed. This means that stencils of any order, both in space and time, explicit or implicit, can be implemented using ELLIPTIc.

2.4 Open Source and Development

ELLIPTIc is distributed under the MIT License (Open Source Initiative, 2017), which is a relatively short and permissive license. It allows for commercial use, modification, distribution and private use, as long as the original copyright and license are included in every copy of the software.

The popular website GitHub was chosen to host the project, along with the git version control system. ELLIPTIc also includes an automated testing suite, allowing for regression tests and continuous integration, so that on every commit the entire test suit is run and code coverage is verified.

Installing and configuring an environment for running ELLIPTIc can be an involved process, including the installation of MOAB and Trilinos with the correct options. For that matter, ELLIPTIc also includes Docker support (Merkel, 2014), which is a software that provides lightweight and isolated virtualization through containers.

3 SOFTWARE ARCHITECTURE

The ELLIPTIc project is developed with many software engineering practices being applied, such as the Single Responsibility principle and the Don't Repeat Yourself (DRY) principle, among others. Also, the Python language has a culture that encourages the writing of high quality code, with readability and maintainability in mind. This can be seen at the Zen of Python (Peters, 2017).

With that in mind, Fig. 1 represents the data pipelines, logical connections and modules that have been adopted for the development of the package. It depicts a high-level overview of the software architecture. In the following sections, we briefly discuss each module and its role on the development of numerical methods through ELLIPTIc.

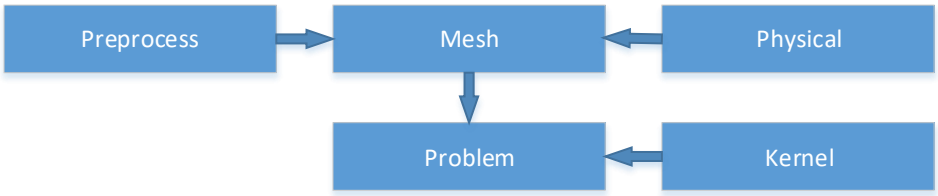

Figure 1. High-level software architecture of ELLIPTIc.

3.1 Preprocess

The Preprocess module leverages the MOAB support of the HDF5 format for storing data. It can be used to import and preprocess a mesh file from any format supported by MOAB, or to create and preprocess a new mesh using MOAB primitives.

By employing a modular approach, similar to a middleware architecture, it is possible to define which preprocessing modules should be used through a configuration file.

Fig. 2 shows an example of a preprocessor pipeline, whose steps are thereafter explained.

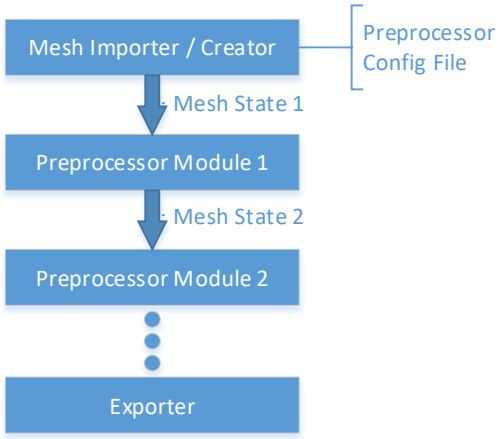

Figure 2. Example of a preprocessor pipeline.

Proceedings of the XXXVIII Iberian Latin-American Congress on Computational Methods in Engineering
P.O. Faria, R.H. Lopez, L.F.F. Miguel, W.J.S. Gomes, M. Noronha (Editores), ABMEC, Florianópolis, SC, Brazil, November 5-8, 2017.

- The first module, the Mesh Importer / Creator, reads the preprocessor configuration file, which is then analyzed and the subsequent preprocessor modules are found and instantiated. Then, from the configuration file, the Mesh Importer / Creator reads basic mesh information, such as which mesh file is to be imported, or in the case of mesh creation, what is the mesh size, etc.
- After the Mesh Importer / Creator finishes defining the starting mesh, it forwards the mesh, which is in the initial state (empty) to the first preprocessor module. After its preprocessing is done, it forwards the new mesh to the next preprocessor module, and so on.
- Each preprocessor module is responsible for storing important data or modifying the mesh. Examples include adjacencies, volumes and areas.
- After the preprocessing pipeline is finished, the final mesh is then forwarded to a mesh exporter, which will save the mesh in the HDF5 format, used by ELLIPTIc.

3.2 Mesh

The Mesh module is perhaps the most intuitive module present in the package. It basically interfaces with MOAB and provides an API for the other modules to use and interact indirectly with MOAB. It also provides the mesh iteration logic for the Kernel modules.

This approach allows for another mesh engine to be used instead of MOAB, given that the interface is correctly used and abstracted, following the interface adopted on ELLIPTIc.

3.3 Physical

When dealing with physical modeling and simulation, it is desired to have a concise and easy way to represent and manage physical properties and boundary conditions.

On ELLIPTIc, this is done through the Physical module, which defines an interface for registering and accessing physical properties and boundary conditions.

The physical properties and boundary conditions are stored as fields through MOAB tags, which can be associated with an arbitrary number of entities of any dimension. These fields can also be modified during the simulation, which suggests that, in practice, physical properties and boundary conditions are initial conditions, and it's the engineer's responsibility to interpret those as needed and make the necessary distinctions.

3.4 Kernel

The Kernel modules are the main part of ELLIPTIc. Each Kernel is responsible for explicitly or implicitly computing a field.

In the case of an explicit field, the Kernel simply fills a MOAB tag with information computed from the mesh.

If a field is being implicitly calculated, the Kernel works as a matrix assembly unit. In this case, a Trilinos sparse matrix is filled, which will be the left-hand side matrix of the linear system $Ax = b$. The right-hand side vector b is given from a previously computed field.

ELLIPTIc provides base classes, or mixins, that can be plugged into user-defined Kernels to provide extra functionality and define behavior. For example, a Kernel that might need to access entity adjacencies should include the mixin responsible for supplying that functionality. As another example, a Kernel that iterates through a given set of entities should inherit from the mixin responsible of defining entities to be iterated on.

3.5 Problem

The Problem modules are responsible for setting up and running a class of physical problem or simulation. A Problem associates a Kernel with the given mesh, and interoperates with an implicit solver, if applicable.

Examples of concrete implementations of Problem would be, for instance, a class responsible for solving implicitly a linear system, or a class responsible for solving explicitly a system. These classes would use a Kernel as matrix assembler or field computer, and retrieve the calculated solution.

4 RESULTS

In this section, we present two problems and one solution for each using ELLIPTIc: a steady state diffusion type problem, and a transient two-phase flow problem. The code can be accessed through the Digital Object Identifier (DOI) found at (Caminha, 2017).

These are preliminary results, and simply demonstrate the feasibility of using ELLIPTIc as a tool for developing numerical methods. In the future, specific works will focus on more robust and advanced methods.

4.1 Diffusion Problem

The steady state, single-phase, diffusion problem presented here is the solution of the elliptic pressure equation (Carvalho, 2005):

$$\nabla \cdot (-\underline{\underline{K}} \nabla p) = Q \,. \tag{1}$$

where $\underline{\underline{K}}$ is a second-order tensor, representing the medium permeability, p is the single-phase fluid pressure, and Q is a volumetric source flow.

When solving Eq. (1), we adopted the finite volume method with a two-point stencil (Souza, 2015), which can only yield correct results on K-orthogonal meshes. Here, we suppose $\underline{\underline{K}}$ as being a scalar, isotropic medium, that may vary on space. This method works by approximating the fluxes through the i-th control surface with the following expression:

$$\vec{v}_i \cdot \vec{N}_i = -\frac{2K_{\bar{R}}K_{\bar{L}}}{K_{\bar{R}}\Delta x_{\bar{L}} + K_{\bar{L}}\Delta x_{\bar{R}}}(p_{\bar{R}} - p_{\bar{L}}) \,. \tag{1}$$

where $\vec{v}$ is the flow velocity, $\overrightarrow{N_i}$ is the area normal vector of the i-th control surface, $\overline{R}$ and $\overline{L}$ are the right-hand and left-hand side volumes, respectively, in relation with the control surface, and Δx represents given volume's length.

The problem was solved on meshes of sizes 512, 4.096 and 32.758 degrees of freedom. The problem geometry is a cuboid of size 3 on the x direction, and 1 on the y and z directions. For this problem, the permeability is set as 1. There is a Dirichlet condition of magnitude 1 in the *yz* plane at $x = 0$, and another of magnitude 0 in the *yz* plane at $x = 3$. A Neumann condition of magnitude 0 is set at the rest of the boundary.

This is, by symmetry, a 1-D problem, which analytical trivial solution is linear, varying through the *x* direction. This behavior was reproduced correctly. The results are shown in Fig. 3 for each mesh size.

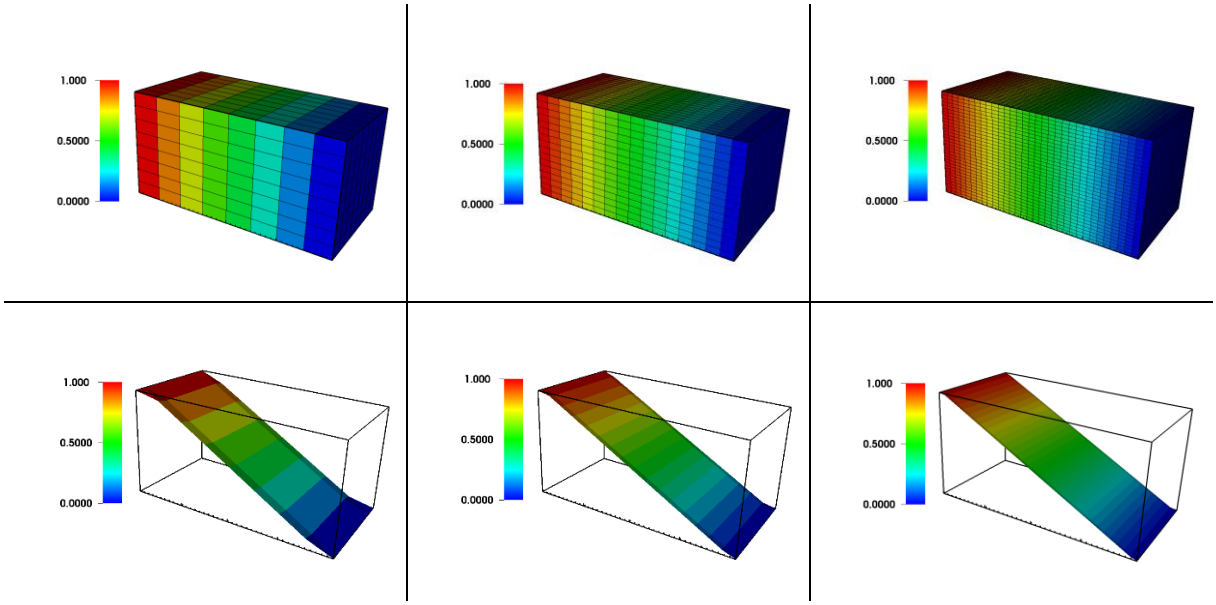

Figure 3. Solutions for the pressure on three different mesh sizes. On top, the results for the full mesh are shown. Below the elevated results for a slice are shown.

4.2 Two-Phase Flow Problem

We have chosen a problem that models two-phase flow in porous media (Carvalho, et al., 2006) to represent the transient capabilities of ELLIPTIc. Supposing that the porous medium is saturated with oil and water, the formulation is based on the Buckley-Leverett equation (Carvalho, 2005):

$$\phi \frac{\partial S_w}{\partial t} + \vec{v} \nabla f_w = Q_w . \tag{2}$$

where ϕ is the porosity of the medium, S_w is the saturation of the water phase, t is the time variable, $\vec{v}$ is the flow velocity, f_w is the water fractional flux, and Q_w is the volumetric flow rate of water.

The water fractional flux is a non-linear function of the water saturation. The relative mobilities of each phase characterize the water fractional flux as:

$$f_w = \frac{\lambda_w}{\lambda_w + \lambda_o} = \frac{\frac{k_{rw}}{\mu_w}}{\frac{k_{rw}}{\mu_w} + \frac{k_{ro}}{\mu_o}}. \tag{3}$$

where λ_w and λ_o are the water and oil mobilities, respectively, μ_w and μ_o are the water and oil viscosities, and k_{rw} and k_{ro} are the relative permeability functions of water and oil, respectively.

There are several models for the relative permeabilities that can be found in the literature. Here, we adopt the following quadratic model (Brooks & Corey, 1964):

$$k_{rw} = \left(\frac{S_w - S_{wr}}{1 - S_{wr} - S_{or}} \right)^2. \tag{4}$$

$$k_{ro} = \left(\frac{1 - S_w - S_{wr}}{1 - S_{wr} - S_{or}} \right)^2. \tag{5}$$

where S_{wr} and S_{or} are the residual saturation of water and oil, respectively, and are considered negligible, so that $S_{wr} = 0$ and $S_{or} = 0$.

Here, we adopt a finite volume explicit formulation for the transient term of the saturation equations, disregarding the volumetric flow source term, leading to the discrete form (Carvalho, 2005):

$$S_{w(k)}^{n+1} = S_{w(k)}^{n} + \frac{\Delta t}{\phi \forall_k} \sum_{i=1}^{NF} f_w \vec{v}_{k,i} \cdot \vec{N}_{k,i}. \tag{6}$$

where *n* is the current time level, $\forall_k$ is the volume of the k-th finite volume, *NF* is the number of control surfaces forming the k-th finite volume, and $\vec{N}_{k,i}$ is the outward normal, with magnitude equal to the surface area, of the i-th control surface in relation with the k-th finite volume.

The flow balance terms are approximated using a first-order upwind scheme (Hirsch, 2007). So, for each i-th control surface of each k-th finite volume:

$$f_w = \begin{cases} f_{w,L}, & if \quad \vec{v}_{k,i} \cdot \vec{N}_{k,i} < 0 \\ f_{w,R}, & if \quad \vec{v}_{k,i} \cdot \vec{N}_{k,i} \geq 0 \end{cases}. \tag{7}$$

where $f_{w,L}$ and $f_{w,R}$ represent the fractional flux of the left-hand and right-hand sides of the i-th control surface.

The problem geometry consists of a three-dimentional cuboid of size 5 on the x direction, and 1 on the y and z direction. An initial, static velocity field $\vec{v} = (1,\ 0,\ 0)$, i.e. a velocity of 1 on the x direction, and 0 otherwise. The initial water saturation field is zero everywhere, except on the *yz* plane at $x = 0$, where the water saturation is 1. Also, there is a prescribed water saturation of 1 at the same plane.

The results for that problem are shown in Fig. 4 at three different timesteps, which gives a qualitative solution, being noticeably diffusive as a first-order upwind scheme is adopted.

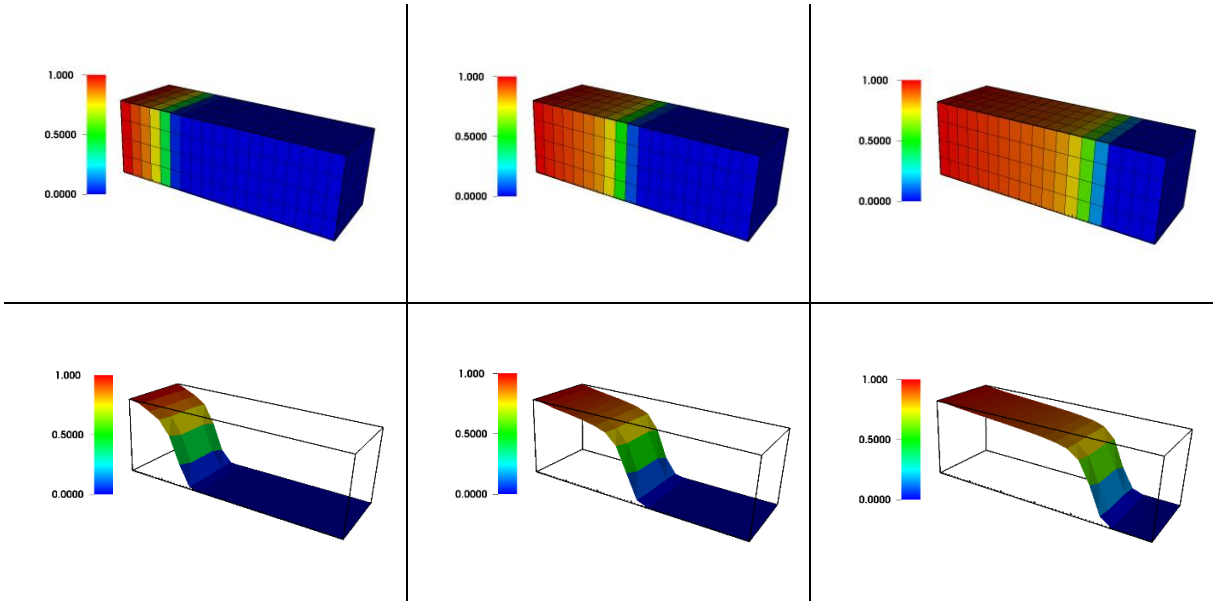

Figure 4. Solutions for the saturation on three different timesteps. On top, the results for the full mesh are shown. Below the elevated results for a slice are shown.

5 CONCLUDING REMARKS AND FUTURE WORKS

The ELLIPTIc package has proven to be a software engineering challenge, given its very high-level nature, while also keeping high performance under concern. This allows engineers to build and run simulations while also having the possibility to apply code reuse and other good programming practices.

In the future, the ELLIPTIc team aims to leverage both Trilinos and MOAB parallelism. Also, a Just-in-Time compiler for ELLIPTIc Kernels is being studied. With this, ELLIPTIc could achieve much faster computational times, while still keeping a very declarative and intuitive interface for Kernel development.

Finally, an Adaptive Mesh Refinement (AMR) module is being explored. This will require the Preprocessor module to be decoupled, so that it can be run for each new entity in the case of local refinement, or in the entire mesh, if global remeshing is applied.

ACKNOWLEDGEMENTS

The authors would like to thank CAPES and CNPq for the financial aid, used for the development of this research.

REFERENCES

Alnaes, M. S. et al., 2015. The FEniCS project version 1.5. *Archive of Numerical Software.*

Brooks, R. & Corey, A., 1964. Hydraulic Properties of Porous Media. *Hydrology Papers.*

Caminha, G. P. K., 2017. ELLIPTIc Release v0.2-alpha. *doi: 10.5281/zenodo.853401*.

Carey, G. F., Anderson, M., Carnes, B. & B., K., 2004. Some aspects of adaptive grid technology related to boundary and interior layers. *Journal of Computational and Applied Mathematics,* p. 55–86.

Carvalho, D. K. E., 2005. Uma Formulação do Método dos Volumes Finitos com Estrutura de Dados por Aresta para a Simulação de Escoamento em Meios Porosos. *Doctoral dissertation on Mechanical Engineering - Universidade Federal de Pernambuco.*

Carvalho, D. K. E., Willmersdorf, R. B. & M., L. P. R., 2006. A Node-Centered Finite Volume Formulation for the Solution of Two-Phase Flows in Non-Homogeneous Porous Media. *International Journal for Numerical Methods in Fluids,* 53(8), p. 1197–1219.

Guyer, J. E., Wheeler, D. & Warren, J. A., 2009. FiPy: Partial Differential Equations with Python. *Computing in Science & Engineering,* 11(3), pp. 6-15.

Heroux, M. et al., 2003. An Overview of Trilinos. *Sandia National Laboratories.*

Hirsch, C., 2007. *Numerical Computation of Internal and External Flows.* 2 ed. : Butterworth-Heinemann.

Josell, D., Wheeler, D., Huber, W. H. & Moffat, T. P., 2001. Superconformal electrodeposition in submicron features. *Physical Review Letters,* 87(1).

Kirk, B. S., Peterson, J. W., Stogner, R. H. & Carey, G. F., 2006. libMesh: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. *Engineering With Computers,* 22(3-4), p. 237–254.

Logg, A. & Wells, G. N., 2010. DOLFIN: Automated finite element computing. *ACM Transactions on Mathematical Software,* 37(2).

Merkel, D., 2014. Docker: lightweight Linux containers for consistent development and deployment. *Linux Journal,* 2014(239).

Open Source Initiative, 2017. *The MIT License.* [Online] Available at: https://opensource.org/licenses/MIT

Peters, T., 2017. *The Zen of Python.* [Online] Available at: https://www.python.org/dev/peps/pep-0020/

Souza, M. R. A., 2015. Simulação Numérica de Escoamento Bifásico em Reservatórios de Petróleo Heterogêneos e Anisotrópicos Utilizando um Método de Volumes Finitos

"Verdadeiramente" Multidimensional com Aproximação de Alta Ordem. *Doctoral dissertation on Civil Engineering - Universidade Federal de Pernambuco.*

Tautges, T. J. et al., 2004. MOAB: A Mesh-Oriented Database. *Technical Report.*

The HDF Group, 1997-2017. *Hierarchical Data Format, version 5.* [Online] Available at: http://www.hdfgroup.org/HDF5/