SCHWARZ SPLITTING FOR THE STEADY STATE PROBLEM OF SALTWATER INTRUSION IN COASTAL AQUIFERS

E. Maroudas[♮], N. Vilanakis[‡], Ch. Antonopoulos[♮], E. Mathioudakis[†], Y. Saridakis[‡] and M. Vavalis[♮]

Abstract—In this study we investigate the effectiveness of a meta-computing method on the modeling and the implementation of a truly multi-domain, multi-physics numerical scheme for an important practical problem in the area of environmental engineering. A case study of the steady state of saltwater intrusion in costal aquifers is considered and a meta-computing scheme is developed and implemented on the basis of modern highly efficient software tools and practices. Numerical experiments exhibit the several desired characteristics of the proposed methodology and the associated implementation. They also justify the necessity for further research and development for an emerging new numerical computing paradigm that although known has not yet prove its practical value in particular for realistic engineering problems.

Keywords—Multi-Physics - Multi-Domain problems, FEniCS software, Schwarz method.

I. Introduction

OVERLAPPING domain decomposition methods [1] are efficient and flexible. They are also inherently suitable for high level parallel computing the numerical solution of partial differential equations (PDEs), where the methods of concern are based on a physical decomposition of a global solution domain. The global solution of the PDE problem is then sought by solving the smaller subdomain problems collaboratively and then combining their individual solutions. Their collaboration is realized through an iteration scheme that allows information flowing among the subproblems. These numerical methods are therefore termed as domain decomposition (DD) methods.

It is important to point out that our methodology is solely at continuous level. Linear algebra based DD methods are of obvious importance but unable to serve the high level meta-computing paradigm which we believe has great potential for multi-domain multi-physics (MDMP) problems. In this paradigm the basic components are PDE problems and numerical discretization and linear algebra computations take place only within each subdomain and not globally.

The rest of this paper is organized as follows. In the next session we briefly present our efforts to develop a framework for the numerical solution of MDMP problems. The mathematical description of the well known Schwarz method that consists one of the basis of our framework is given in Section III. Section V includes material related to our implementation. Selective numerical results are presented in section VII. Our conclusions are given in section VIII.

Manuscript received November 8, 2015.

[‡]School of Production Engineering and Management, Applied Math & Computers Laboratory, Technical University of Crete, 73100 Chania, Greece

[†]School of Mineral Resources Engineering, Applied Math & Computers Laboratory, Technical University of Crete, 73100 Chania, Greece

[♮]Department of Electrical and Computer Engineering, University of Thessaly, Gklavani 37, 38221 Volos, Greece

Email of the corresponding author : y.saridakis@amcl.tuc.gr

II. A platform for multi-domain, multi-physics PDE problems.

There exist a plethora of software platforms for solving composite PDE problems in various way and utilizing various programming languages and techniques. Our platform utilizes and extends the Python user interface of the FEniCS Dolfin library. The reason behind our preference in Python is clearly practical. Its syntax is closer to UFL syntax and is less time consuming to experiment with due to its scripting nature. our platform is based on FEniCS 1.3 and is briefly described below. More information about our efforts to design and implement this platform can be found in [2].

As mentioned we focus on multi-domain multi-physics (MDMP) problems modeled with PDEs. As such we propose that every new solving module should be implemented on top of the existing ones, either as a new Python module using the available data structures and classes, or as an external dynamically shared C++ library, wrapped as a Python module using SWIG [3].

Our goal is to design and offer an enhanced meta-computing environment based on simple scripting languages and their practices that facilitate the numerical solution of PDEs associated with existing MDMP mathematical models. To accomplish that we utilize state of the art numerical solvers as offered by the supported FEniCS numerical discretization schemes and the state of the art linear algebra backends.

The platform aims to cover a wide range of problems, following a generic design that can support arbitrary shapes (rectangular or curvilinear) for domains and interfaces, for both 2D and 3D geometries.

One of our critical development decisions during our development phase was to keep compatibility with existing user codebases. Therefore to eliminate the possibility of breaking any existing functionality we kept the official release of FEniCS unmodified, putting all the new functionality on external Python modules.

Problems with great interest are problems with different elliptic differential operators on different subdomains as well as problems with different PDE discretization and solving

modules on different subdomains. FEniCS already supports independent subdomain definitions; the platforms honors the existing infrastructure and builds upon it.

There are two emerging methodologies integrated to our platform, that can be used directly with any existing type of MDMP problem, as far as it conforms with the mathematical model behind them. One is a hybrid stochastic/deterministic Monte Carlo-based approach [4] and the other is an overlapping domain decomposition method known as the classical alternating Schwarz method [1, chapter 2.1] considered below.

III. Schwarz Splitting Methodology

THE classical alternating Schwarz method demonstrates the basic mathematical idea of overlapping domain decomposition methods.

For presentation reasons, and without loss of generality, we consider the following very simple boundary-value problem where the domain Ω is defined as the union of a circle Ω_1 and a rectangle Ω_2, as it is depicted in Fig. 1.

$$-\nabla^2 u = f \text{ in } \Omega = \Omega_1 \cap \Omega_2,$$
$$u = g \text{ on } \vartheta\Omega.$$

The part of the subdomain boundary $\vartheta\Omega_i \backslash \vartheta\Omega$ is referred to as the artificial internal boundary of the subdomain Ω_i for $i = 1, 2$.

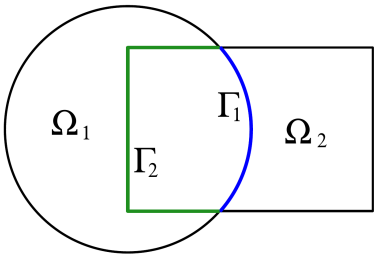

Fig. 1: A composite PDE model problem

In order to utilize analytical solution methods for solving the Poisson equation on a circle and a rectangle separately, Schwarz proposed the following iterative procedure for the solution in the entire composite domain Ω. Let u_i^n denote an approximate solution in subdomain Ω_i, and f_i the restriction of f to Ω_i. Starting with an initial guess u_0, we iterate by computing successive approximate local solutions u^i, $i = 1, 2, \ldots$, until they converge to the overall solution. During each iteration, we first solve the Poisson equation restricted to the circle Ω_1, using the previous iterations solution from Ω_2 on the artificial internal boundary Γ_1 :

$$-\nabla^2 u_1^i = f_1 \text{ in } \Omega_1,$$
$$u_1^i = g \text{ on } \vartheta\Omega_1 \backslash \Gamma_1,$$
$$u_1^i = u_2^{n-1} \mid \Gamma_1 \text{ on } \Gamma_1.$$

Then, we solve the Poisson equation on the rectangle Ω_2, using the latest solution u_1^i on the artificial internal boundary Γ_2:

$$-\nabla^2 u_2^i = f_2 \text{ in } \Omega_2,$$
$$u_2^i = g \text{ on } \vartheta\Omega_2 \backslash \Gamma_2,$$
$$u_2^i = u_1^i | \Gamma_2 \text{ on } \Gamma_2.$$

Note that the classical alternating Schwarz method is sequential by nature, meaning that the two Poisson solves within each iteration must be carried out in a predetermined sequence, first in Ω_1 then in Ω_2. For more than two subdomains though, it exhibits pipeline type of parallelism in a Gauss-Seidel manner.

A variant of the above method, which inherently promotes parallel computing, is called additive Schwarz method and is a Jacobi type variation of the above described scheme. This variant converges slower for almost all problems.

IV. An Environmental Engineering Application

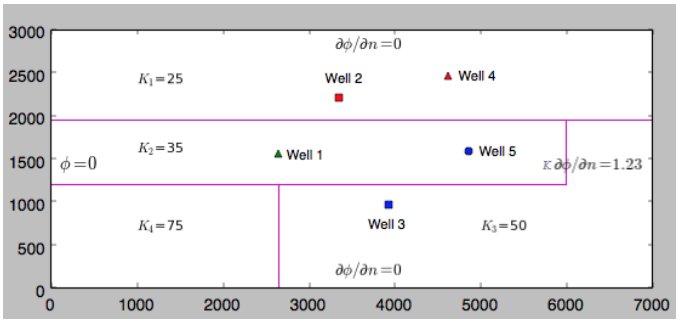

Fig. 2: Kalimnos aquifer.

To demonstrate the potential of the PDE solving procedure mentioned above and of the associated implementation framework to deal with large application problems, we consider solving a PDE that describes the steady state problem of saltwater intrusion in coastal aquifers. The PDE needs to be solved in every iteration step of a stochastic optimization algorithm (cf. [5]–[7]), used to optimally control pumping from all active pumping sources (wells) of a coastal aquifer and protect them from salinization. Its solution (flow potential) is being used to locate/determine the interface between salt and fresh water. The aquifer considered here and depicted in Fig. 2 simulates of a real coastal aquifer located at Bathi area in the Greek island of Kalymnos (cf. [8]). The computer realization of the problem is achieved by using FEniCS [9], [10] computing platform, while the whole implementation has been developed in Python.

Specificaly, let us consider the elliptic PDE

$$\frac{\partial}{\partial x}(K\frac{\partial \phi}{\partial x}) + \frac{\partial}{\partial y}(K\frac{\partial \phi}{\partial y}) + N - Q = 0 \ , \quad (x, y) \in \mathcal{R}, \quad (1)$$

where ϕ (m^2) denotes the Struck's flow potential, N (m/day) denotes the total aquifer recharge uniformly distributed over the surface of the aquifer, K (m/day) denotes the hydraulic conductivity and Q (m/day) denotes the total aquifer discharge. Furthermore, let us assume that the rectangular-shaped aquifer $\mathcal{R}$ extends over an area of 7×3 Km, is *heterogeneous* with respect to the hydraulic conductivity, and contains M wells w_i $(i = 1, \ldots, M)$ pumping at Q_i (m^3/day) rates. In our test problem the above physical parameters are assumed the values of $N = 0.03$ $m/year$, $M = 5$ and $Q_1 = 252$ m^3/day, $Q_2 = 450$ m^3/day, $Q_3 = 749$ m^3/day,

$Q_4 = 1045\ m^3/day$ and $Q_5 = 1270\ m^3/day$, while the hydraulic conductivity K assumes the values $K_1 = 25\, m/day$, $K_2 = 35\ m/day$, $K_3 = 50\ m/day$ and $K_4 = 75\ m/day$ associated with the four sub-regions of $\mathcal{R}$ (see Fig. 2). Moreover, the total discharge rate Q assumes the value

$$Q = \sum_{i=1}^{5} \tilde{Q}_i \delta(x - x_i, y - y_i)$$

where $\tilde{Q}_i$ denotes the pumping rate Q_i normalized over some elemental area and $\delta(x-x_i, y-y_i)$ denotes the Delta function. Finally, Dirichlet boundary condition $(\phi(0, y) = 0)$ is assumed on the left (coastline) edge, while, on all other edges, Neumann boundary conditions are imposed, as shown in Fig. 2.

The above problem has been solved using the FEniCS FE solver and the computed flow potential is depicted in Fig. 3.

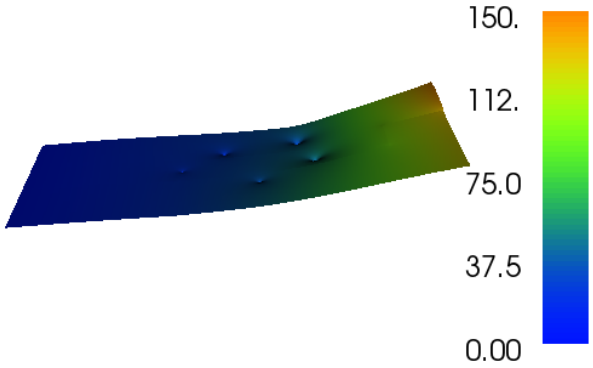

Fig. 3: Kalimnos aquifer flow potential

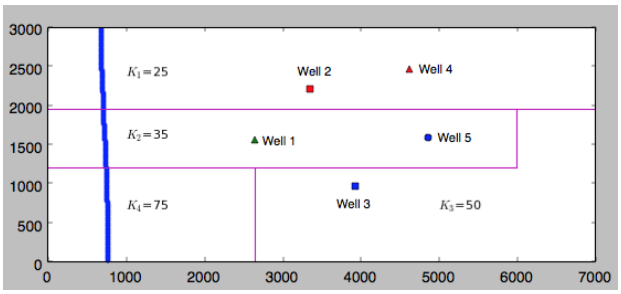

Fig. 4: Interface location between salt and fresh water.

The interface between salt and fresh water is being algorithmically determined in the sequel, and shown graphically in Fig. 4.

V. Implementation

We implement the additive Schwarz method and use it as a high level solver to our MPDMP problems. The full description of our implementation is beyond the scope of this paper. Below we discuss few implementation issues that characterize our implementation.

A. Split problem into files

For the best utilization of our platform, we propose and encourage the use of an organization scheme based on defining each subdomain into a separate file.

This organization scheme highlights the dependence between domains as distinct programming units. It also allows easy collaboration and sharing between different researchers or research groups and hides problem definition details from the not interested parties. Another more technical reason is that this subdomain separation to files, greatly simplifies the implementation for supporting remote solvers and methods as web services, a direction we follow which is quite attractive due to its performance benefits.

B. Python module

All underlying datatypes of the base classes are either pure Python or FEniCS objects. There is no dependence from third party software libraries at this level. The Python module consists of two files:

solverconfig.py provides the base classes with sanity checks and the API for the solver to function properly.

solver.py implements the solving routine among a handful of helper functions that simplify the whole process and further sanity checks to ensure the proper setup of the user's problem.

Inside solverconfig.py the module defines the following base classes:

class LogInfo Its purpose is to keep track of the progress in a particular subdomain. The available information may be written to a user defined logfile.

class ConfigCommon Holds separately the configuration of the whole solver. Some of the attributes which the user may set are the number of dimensions of the problem, the maximum number of iterations for the solver, a tolerance value that is used to check for convergence, the filenames of the subdomains which will take part in the solving process, whether the user wants the creation of logfiles and whether they want visual plots of the solutions in each iteration. The class provides some predefined default values for all attributes.

class Config2D (as well as class Config3D) Derives from ConfigCommon, with predefined number of dimensions set to 2 (or 3 respectively). Everything else is the same as the parent class.

class ConfigCommonProblem This is the base class the user extends to define each subdomain. There are three methods that need to be overridden. We discuss them analytically below.

1) Domain API: Each subdomain object that inherits from the ConfigCommonProblem base class must override the following methods. The solver object calls these methods, before the actual solving phase begins, in order to gather the appropriate useful information and setup the appropriate data structures for each subdomain.

init() This method holds the UFL [11] definition of the subdomain and sets as class attributes the subdomain's function space, linear and bilinear form of the PDE.

neighbors() It provides information to the solver about the other subdomains this subdomain overlaps with, in order for the solver to automatically update the boundary values after each iteration. It returns a Python dictionary with keys the filename of the neighbor subdomain and as value a method that returns the Boolean value True only for the nodes on the common boundary of this subdomain and the neighbor subdomain.

boundaries() It informs the solver about the fixed external boundaries of the subdomain. It returns a Python list of all the subdomain's external boundaries, each element being a DirichletBC object.

2) Iterative solver: The entry point of the iterative solver is the solve() method as defined inside solver.py. It takes as arguments a ConfigCommon object with the configuration of the solving environment (max iterations, tolerance, etc) and a Python list of user defined problem objects, all derived from ConfigCommonProblem base class. After certain initial steps (create logfiles, initialize solution vectors, etc), the main solving routine, named __solve(subdomains, config), is called.

The main points of interest of the iteration algorithm and two of the helper methods are shown in the associated listing in the Appendix:

After each iteration, for each subdomain solution, the algorithm checks a set of halting criteria in the following order that may terminate the solving process:

1) If the exact solution is known, check for convergence w.r.t. the user defined tolerance value.
2) If the errornorm of the current and previous approximations (thereafter called iterants) is below a user defined tolerance value.
3) If the max iterations limit as defined by the user is reached.

Note that in order for the stop_criterion() method to terminate the algorithm, all subdomains must converge for either of the two first criteria. The third criterion is common for all subdomains.

The solver keeps logfiles for each boundary interface between all overlapping subdomains. They keep track of the progress per iteration in a column based format which is suitable to use as input to graphics modules for example `https://plot.ly` and Gnuplot.

VI. Example

For example a skeleton file (circle2D_1.py) with the definition (in Python) for the circle subdomain in Fig. 1 can be the following as shown in the associated listing in the Appendix.

The skeleton definition is abstract to the geometry and number of dimensions of the subdomain. That means that the same skeleton code from the listing can be used to define the rectangle subdomain as well.

Given two defined subdomains in files circle2D_1.py and rectangle2D_1.py, the driver code that solves them is depicted in the listing 1 below.

Listing 1: Code for solving two overlapping subdomains

```
from dolfin import *
import solverconfig
import solver

import circle2D_1 as circle
import rectangle2D_1 as rectangle

cp = circle.Problem()
rp = rectangle.Problem()
subdomains=[ cp, rp ]

config = solverconfig.Config2D()
solver.solve(subdomains=subdomains,config=config)

# keep plots on screen
interactive()
```

VII. Numerical Experiments

The above descripted implementation allow us to perform a set of experiments that validate primarily the convergence and secondly our assumptions concerning the desired characteristics of the method.

We therefore consider the problem described in section IV above.

This problem is naturally split into the four problems that are defined due to the different PDE operator (different Qs and Ks in PDE equation 1) and clearly depicted in Fig. 2. We further split the right bottom domain into two subdomains for the simple domain geometry reason so we only consider rectangular subdomains. This results into a total of five subproblems.

In multidomain implementation for Schwarz method an overlapping area for the second subdomain with hydraulic conductivity K_2 is considered. This area extends horizontally for 400 m inside the other subdomains. Also an overlapping region for the fourth subdomain in the third subdomain for 400 m is assumed. The region for 1400 m including well3 inside the second subdomain is chosen for the overlapping region for the third subdomain. The above overlapping configuration results into a total of 14 interfaces among the five subdomains. Unfortunately it is difficult to visualize them easily.

Due to space limitations

- No details on the FE modules utilized withing each subproblem are given (they could be any state-of-the-art FE module available within FEniCS and beyond).
- We present below a only a series of figures that reveal the convergence characteristics of the method. A complete experimental study of the proposed method is beyond the scope of this paper and will be presented elsewhere.

As clearly seen from Fig. 6 and 5 the method regardless the involvement of five subproblems with a total of fourteen interfaces converges rather rapidly.

The nature of the convergence is depicted also in Fig. 7 where it is seen that high frequency error terms are first eliminated (during the first few iterations) and the solution is further being smooth in subsequent iterations.

Finaly, comparing Fig. 7 and 3 observe that our DD iteration scheme converges to the solution computed by means of the conventional FE method on the whole domain.

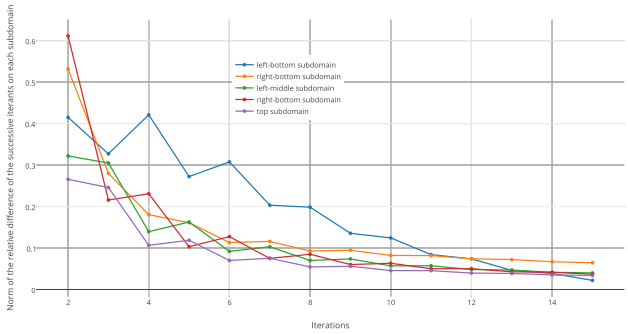

Fig. 5: The history of convergence with repect to the norm of the relative differences in the successive iterants on each subproblem.

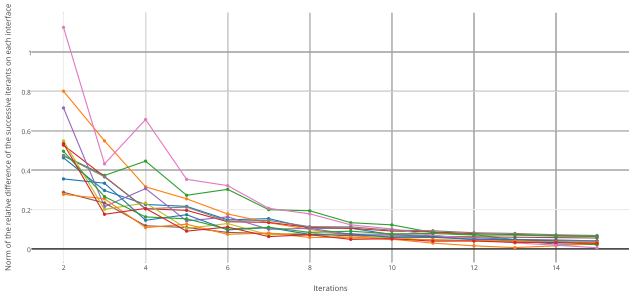

Fig. 6: The history of convergence with respect to the norm of the relative differences in the successive iterants on each of the 14 interfaces associated with the subproblems in Fig. 5.

VIII. Conclusion

We have developed a software platform with convenient Application Programming Interfaces and utilize it for the effective numerical solution of a practical problem in environmental engineering. Our scheme shows that the metacomputing paradigm for solving composite MDMP problems on state-of-the-art platforms consists a very promising approach. It allows us to relate the multi- nature of the problem to associated programming components and solving modules. Surely, more research and development is needed on both theoretical, algorithmic and programming matters, and we plan to work in this direction.

Acknowledgment

The present research work has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program Education and Lifelong Learning of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALES (Grant number: MIS-379416). Investing in knowledge society through the European Social Fund.

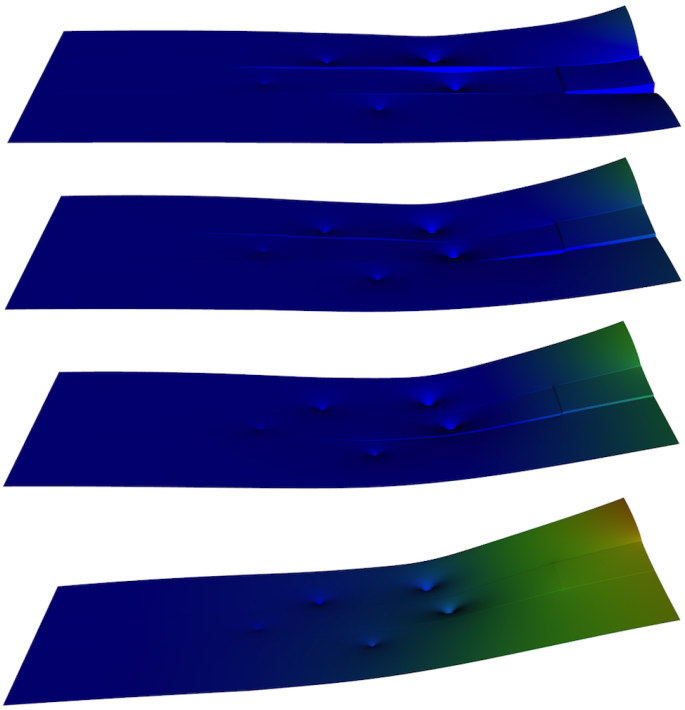

Fig. 7: Plots of the computed global solution at iterations 1, 2, 15 and 50 from top to bottom respectively.

References

[1] X. Cai, "Overlapping domain decomposition methods," in *Advanced Topics in Computational Partial Differential Equations*, ser. Lecture Notes in Computational Science and Engineering, H. Langtangen and A. Tveito, Eds. Springer Berlin Heidelberg, 2003, vol. 33, pp. 57–95.

[2] C. Antonopoulos, M. Maroudas, and M. Vavalis, *Journal of Mathematics and Statistical Science*, vol. to appear, p. No. JMSS15090801, On PDE problem solving environments for multidomain multiphysics problems.

[3] D. M. Beazley, "Swig: An easy to use tool for integrating scripting languages with c and c++," in *Proceedings of the 4th Conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*, ser. TCLTK'96. Berkeley, CA, USA: USENIX Association, 1996, pp. 15–15.

[4] M. Vavalis, "Implementing hybrid pde solvers," in *Proceedings of the Eleventh International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, 2014.

[5] P. Stratis, Y. Saridakis, M. Zakynthinaki, and E. Papadopoulou, "Alopex stochastic optimization for pumping management in fresh water coastal aquifers," in *Journal of Physics: Conference Series*, 2014, vol. 490.

[6] P. Stratis, G. Karatzas, E. Papadopoulou, M. Zakynthinaki, and Y. Saridakis, "Stochastic optimization for an analytical method of saltwater intrusion on coastal aquifers," 2015, submitted to PLOSone.

[7] P. Stratis, Z. Dokou, G. Karatzas, E. Papadopoulou, and Y. Saridakis, "Stochastic optimization and numerical simulation for pumping management of the hersonissos freshwater coastal aquifer in crete," in *Procs of 2015 Int. Conf. on Water Resources, Hydraulics and Hydrology, Zakynthos, Greece*, pp. 329–334.

[8] A. Mantoglou, M. Papantoniou, and P. Giannoulopoulos, "Management of coastal aquifers based on nonlinear optimization and evolutionary algorithms," *Journal of Hydrology*, vol. 297, pp. 209–228, 2004.

[9] *http://fenicsproject.org*.

[10] A. Logg, Anders, K.-A. Mardal, and G. Wells, *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2004, vol. 84.

[11] M. S. Alnæs, *UFL: a Finite Element Form Language*. Springer, 2012, ch. 17.

Listing 2: Common skeleton code example for subdomain definitions

```
# user defined methods
def OveralappingWithOther():              pass
def getOrCreateMesh():                    pass
def userDefinedUFL():                     pass
def userDefinedBoundaryCondition():       pass

# skeleton example
def ExtBC(x,on_boundary):
    return on_boundary and not OveralappingWithOther()

def ExtIface(x,on_boundary):
    return on_boundary and OveralappingWithOther()

class Problem(ConfigCommonProblem):
    def init(self,*args,**kwargs):
        mesh = getOrCreateMesh(*args,**kwargs)
        self.V = FunctionSpace(mesh,'Lagrange',1)
        self.a, self.L = userDefinedUFL(V)

    def neighbors(self):
        interface = {}
        interface['rectangle'] = ExtIface
        return interface

    def boundaries(self):
        bc = DirichletBC(self.V, userDefinedBoundaryCondition(), ExtBC)
        return [ bc ]
```

Listing 3: Core code of the iterative algorithm routine

```
def __interpolate_interfaces(subdomains):
    for subdomain in subdomains:
        for iface in subdomain.interfaces.itervalues():
            interpolant = interpolate(iface['solution'],subdomain.trial_space())
            iface['interpolant'].vector()[:] = interpolant.vector()

def __solve_iteration(subdomains):
    for subdomain in subdomains:
        subdomain.solve()

def __update_interfaces(subdomains):
    for subdomain in subdomains:
        for iface in subdomain.interfaces.itervalues():
            iface['previous'].vector()[:] = iface['current'].vector()
            iface['bc'].apply(iface['current'].vector())

def __solve(subdomains,config):
    iteration = 0
    iterate = True
    while iterate:
        iteration += 1

        __interpolate_interfaces(subdomains)
        __solve_iteration(subdomains)
        __update_interfaces(subdomains)
```

```
26
27          if config.show_solution_plots:
28              for subdomain in subdomains:
29                  plot(subdomain.solution(), title=subdomain.name)
30          for subdomain in subdomains:
31              if stop_criterion(config,subdomain,iteration):
32                  iterate = False
33
34      return [ subdomain.solution() for subdomain in subdomains ]
```