Aachen Institute for Advanced Study in Computational Engineering Science

Preprint: AICES-2008-10

23/December/2008

Software Supporting Optimal Experimental Design: A Case Study of Binary Diffusion Using EFCOSS

A. Rasch, H. M. Bücker and A. Bardow

RWTH AACHEN UNIVERSITY

AICES

Financial support from the Deutsche Forschungsgemeinschaft (German Research Association) through grant GSC 111 is gratefully acknowledged.

List of AICES technical reports: http://www.aices.rwth-aachen.de/preprints

Software Supporting Optimal Experimental Design: A Case Study of Binary Diffusion Using EFCOSS

Arno Rasch [a,*] H. Martin Bücker [a] André Bardow [b]

[a] *Institute for Scientific Computing, RWTH Aachen University, 52056 Aachen, Germany*

[b] *Process & Energy Laboratory, Delft University of Technology, Leeghwaterstraat 44, 2628 CA Delft, The Netherlands*

Abstract

Methods for optimal experimental design aim at minimizing uncertainty in parameter estimation problems. Despite their long tradition in applied mathematics and importance in practical applications, they are currently not widely used in computational science and engineering. To make the techniques of optimal experimental design more accessible to a broader community, we introduce a novel software environment called EFCOSS and demonstrate its ease of use and versatility in two case studies of binary diffusion experiments. Through the use of a component-based software architecture, integration of automatic differentiation technology and facilitated interfacing to optimization algorithms, EFCOSS minimizes the computational overhead for the user who can thus focus on model development and analysis itself. The presented case studies focus on diffusion experiments in liquids since these experiments are typically very demanding. The use of optimal experimental design techniques allows to reduce experimental time and effort significantly.

Key words: optimal experimental design, parameter estimation, EFCOSS, diffusion experiment, distributed computing, automatic differentiation

* Corresponding author. Full address: Institute for Scientific Computing / RWTH Aachen University / Seffenter Weg 23 / D-52074 Aachen / Germany / phone: +49-(0)241-80-24913 / fax: +49-(0)241-80-22241

Email addresses: rasch@sc.rwth-aachen.de (Arno Rasch), buecker@sc.rwth-aachen.de (H. Martin Bücker), a.bardow@tudelft.nl (André Bardow).

1 Introduction

The aim of modeling is to accurately predict complicated phenomena arising from real-world problems by an interdisciplinary approach bringing together physical understanding, mathematics, and computer science in the context of a scientific or engineering application. The underlying mathematical models with their various parameters typically cause controversy. In particular, it is often not obvious how model parameters effect the outcome of a simulation in different classes of application scenarios. Therefore, a systematic approach for assessing the parameter impact is needed, allowing quantitative model analysis by rigorous comparison with experimental data. The theory of optimal experimental design (OED) (Atkinson and Donev, 1992) identifies an experimental setup that delivers experimental data allowing parameter estimation with minimal uncertainty. Thereby, experimental effort can often be reduced by orders of magnitude.

In recent years, the importance of proper optimal experimental design was emphasized in various (bio)chemical applications; for recent overviews see Arellano-Garcia et al. (2007); Franceschini and Macchietto (2008). In particular, OED for dynamic systems has recently been studied and improved problem formulations have been proposed (Baltes et al., 1994; Bardow, 2008; Bauer et al., 2000; Benabbas et al., 2005; Bernaerts et al., 2002; Galvanin et al., 2007; Körkel et al., 2004; Navarro-Laboulais et al., 2008; Sedrati et al., 1999; Vlachos et al., 2006).

Despite their high relevance, OED techniques are not widely used in practice today. One of the reasons is the lack of adequate software tools that support the underlying concepts in a user-friendly way. For nonlinear DAE systems, first dedicated software packages with OED-options have been developed recently, in particular VPLAN (Körkel, 2002) and gPROMS (PSE, 2004). However, the software and interfaces in these packages are tailored towards DAE systems. The goal of this work is to demonstrate the use and versatility of a novel software tool called EFCOSS that provides more general interfaces and is therefore designed to make OED available for a broader audience. The idea behind our approach is to let the user concentrate on model development and remove the burden of mathematical optimization, derivative computation, and the integration of different software packages from the user. The latter aspect is similar to the component-based system for process modeling and design described in (Yang et al., 2008). In (Rasch and Bücker, 2008) the features of EFCOSS are described in detail, and various software engineering aspects are elaborated from the developer's point of view. In the present complementary work, we focus on EFCOSS from a user's perspective when solving a typical OED problem with standard objective functions. We stress that the modular structure of EFCOSS allows easy extension to user-defined objective functions

addressing more advanced OED problems.

In this article, we briefly introduce the OED problem formulation in Section 2 and give an overview of EFCOSS in Section 3. The main contribution is the detailed description of using EFCOSS in two case studies of binary diffusion introduced in Section 4 with constant and concentration-dependent coefficients in Section 5 and Section 6, respectively.

2 Optimal Experimental Design

A common scenario in modeling the behavior of a certain system is the identification of model parameters, based on physical experiments. In this task, also referred to as parameter estimation, the goal is to adjust the model parameters such that the model best mimics the behavior of the physical system under investigation. This goal is typically accomplished by minimizing some cost function that describes the discrepancy between measurements obtained from experiments and the corresponding model prediction.

In the sequel, the model is considered as a mathematical function η, mapping input vectors $\mathbf{x} \in \mathbb{R}^m$ and $\boldsymbol{\theta} \in \mathbb{R}^p$ to some output vector $\eta(\mathbf{x}, \boldsymbol{\theta}) \in \mathbb{R}^\ell$. Here, $\boldsymbol{\theta}$ denotes the set of model parameters, and $\mathbf{x}$ represents the set of so-called explanatory variables describing the physical conditions for the experiment. In general, we consider the model η to be nonlinear in $\mathbf{x}$ and $\boldsymbol{\theta}$.

There are standard methods, e.g., least-squares, to estimate model parameters $\boldsymbol{\theta}$ from given experimental data. Quantitative optimal experimental design aims at minimizing the uncertainty of the parameter estimate by appropriately changing the experimental setup. The covariance matrix of a parameter vector $\boldsymbol{\theta}$, denoted by V_θ, can be used to assess the uncertainty of the result. High variance for a certain parameter indicates that the estimated value for this parameter might not be reliable. Furthermore, the correlation between different parameters can be analyzed, based on the information given in V_θ. If two parameters are strongly correlated, they cannot be easily estimated together. In general, V_θ is difficult to obtain. However, for maximum likelihood estimators, V_θ can be approximated by the inverse of the so-called *Fisher information matrix* (Walter and Pronzato, 1997) given by

$$\mathbf{F}(\boldsymbol{\theta}^*) = E\left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}^T}\right]_{\boldsymbol{\theta} = \boldsymbol{\theta}^*} = -E\left[\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T}\right]_{\boldsymbol{\theta} = \boldsymbol{\theta}^*}, \tag{1}$$

where E is the expectation operator, $\mathcal{L}$ is the logarithm of the likelihood function, and $\boldsymbol{\theta}^*$ represents the (unknown) true values of $\boldsymbol{\theta}$. For a given model η, a set of n experimental conditions $\mathbf{x}_i$ collectively represented by $\mathbf{X} :=$

$(\mathbf{x}_1^T, \ldots, \mathbf{x}_n^T)^T$, and a parameter set $\boldsymbol{\theta}$, the Fisher information matrix can be explicitly computed:

$$\mathbf{F}(\eta, \mathbf{X}, \boldsymbol{\theta}) = \sum_{i=1}^{n} Q_i^T V_i^{-1} Q_i \ \in \ \mathbb{R}^{p \times p}, \tag{2}$$

where V_i is the covariance matrix for the measurement errors of the i-th experiment and Q_i is given by

$$Q_i := \left[\frac{\partial \eta(\mathbf{x}_i, \boldsymbol{\theta})}{\partial \theta_1}, \ldots, \frac{\partial \eta(\mathbf{x}_i, \boldsymbol{\theta})}{\partial \theta_p} \right] \in \mathbb{R}^{\ell \times p} \ . \tag{3}$$

In many practical applications, the covariance matrices V_i are assumed to be diagonal, with constant variance σ_i^2. Then, (2) can be written as

$$\mathbf{F}(\eta, \mathbf{X}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \left(\frac{\partial \eta(\mathbf{x}_i, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right)^T \frac{\partial \eta(\mathbf{x}_i, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \ . \tag{4}$$

The Fisher information matrix $\mathbf{F}$ can now be used to minimize the uncertainty in the model parameters, $\boldsymbol{\theta}$, by systematically improving the experimental setup, $\mathbf{x}$, such that some scalar measure Φ on $\mathbf{F}$ is minimized (Walter and Pronzato, 1990)

$$\mathbf{X}^* = \arg\min_{\mathbf{X}} \Phi(\mathbf{F}(\eta, \mathbf{X}, \boldsymbol{\theta})). \tag{5}$$

Common choices for the scalar measure function Φ are described in the next section. Noteworthy, the objective function Φ in (5) already contains first-order derivatives of the model η. Thus, second-order derivatives of η are always required if gradient-based optimization algorithms are to be employed for optimal experimental design. In addition, problem (5) is known to be very complex in general (Walter and Pronzato, 1990). The design of numerical algorithms for OED problems is therefore by itself an active field of research (Bauer et al., 2000; Petzold et al., 2006).

Due to these reasons, the practical formulation, implementation and solution of an OED problem (5) currently put high requirements on the user of these methods, limiting their practical applicability. The software environment EFCOSS discussed in this work is intended to reduce the human effort needed to solve OED problems in practice.

3 The EFCOSS Environment

The abbreviation EFCOSS stands for *Environment for Combining Optimization and Simulation Software*. It is an interactive environment for supporting the combination of simulation- and mathematical optimization codes in an

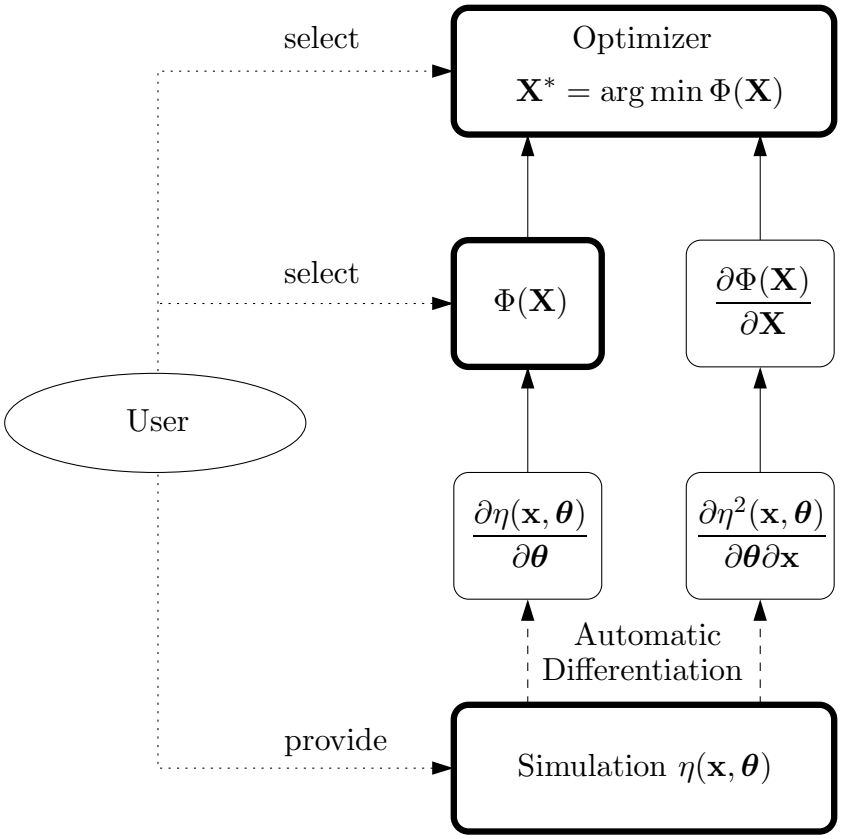

Fig. 1. Building blocks and data flow in an OED problem using EFCOSS

automated fashion. This way, it facilitates the solution of various optimization problems arising in the context of physical modeling and engineering simulation, e.g., design optimization and parameter estimation.

In this article, we describe a scenario, where the user wants to conduct an optimization using standard OED criteria for a given simulation code and some optimization package that is already integrated into EFCOSS. The building blocks for solving a typical OED problem are depicted in Fig. 1 from an abstract point of view. In order to minimize the objective Φ of the problem (5), the optimizer needs to evaluate Φ and its derivative $\partial\Phi/\partial\mathbf{X}$, which in turn requires the computation of the first- and second-order derivatives of the simulation η for each set of explanatory variables, $\mathbf{x}$. The data flow is illustrated by the solid lines. The basic steps requiring user interaction in EFCOSS are indicated by the bold-framed boxes: the user only needs to specify the simulation evaluating η, choose an objective function Φ from a set of predefined OED criteria, and finally invoke the optimization algorithm. EFCOSS takes care of the data flow and the computation of derivatives by means of automatic differentiation, thus eliminating the error-prone manual implementation of first- and second-order derivatives. In essence, the user is responsible for the simulation while the remaining parts are handled by EFCOSS.

EFCOSS provides a component-based distributed software architecture, where the simulation code, the optimization code, and the mathematical objective function are treated as separated components, allowing a seamless exchange of each of these components independently. The use of CORBA technology allows the work in a heterogeneous computing environment where the components are implemented in different languages or running on different platforms. For example, the simulation code may require extensive hardware resources provided only by a supercomputer, while the other software components can be executed locally on the user's desktop PC or laptop. The overall software architecture of a prototype version is described in Bischof et al. (2003). EFCOSS has recently been extended; more details on the current implementation and some examples of commonly-used objective functions for parameter estimation and optimal experimental design are given in Rasch and Bücker (2008). In the following we briefly sketch various features of EFCOSS.

3.1 Rapid Change Between Objective Functions

While EFCOSS also supports parameter estimation problems, we here focus on OED in order to find values for the explanatory variables $\mathbf{x}$ such that the uncertainty in the model parameters $\boldsymbol{\theta}$ is minimal. Three optimality criteria widely-used in OED are as follows:

- The so-called *D-optimality criterion* minimizes the generalized variance of the parameter estimates. This can be achieved by maximizing the determinant of $\mathbf{F}$.
- Another strategy called *A-optimality criterion* is to minimize the sum (or average) of the variances of the parameter estimates. This corresponds to minimizing the trace of the covariance matrix $V_\theta = \mathbf{F}^{-1}$.
- A third example is the so-called *E-optimality* criterion which minimizes the largest eigenvalue of the covariance matrix $V_\theta = \mathbf{F}^{-1}$. This can be viewed as minimizing the longest axis of the confidence ellipsoid of the model parameters.

These three optimality criteria are already available in EFCOSS. In addition, the user can easily implement further criteria in the Python scripting language.

3.2 Rapid Change and Easy Integration of Optimization Software

Different optimization software packages have different strengths and weaknesses. In practical scenarios, the user might want to explore various optimization algorithms in order to assess the robustness of a solution. Therefore, EFCOSS allows to switch the optimization algorithm without touching

the objective function and the underlying simulation η. Several optimization packages are already integrated in EFCOSS; see Table 1. In addition to these optimization packages, the user can integrate further packages with minimal human effort.

Table 1
Optimization packages integrated into EFCOSS (in alphabetic order)

Optimizer	Remarks	Reference
COBYLA	inequality-constraint optimization by linear approximation	Powell (1994)
DONLP2	mixed equality/inequality-constrained SQP-method	Spellucci (1998a,b)
ELSUNC	nonlinear least-squares (unconstrained/ bound-constrained)	Lindström (1982)
ENLSIP	nonlinear least-squares (general constraints)	Lindström (1983)
FFSQP	general constrained nonlinear optimization using two SQP-algorithms	Lawrence and Tits (1996); Zhou et al. (1997)
IPOPT	interior-point filter line-search with equality constraints	Wächter and Biegler (2006)
L-BFGS-B	limited memory BFGS algorithm for large-scale bound-constrained optimization problems	Zhu et al. (1997)
MINPACK-1	nonlinear least-squares (unconstrained) by a modification of the Levenberg-Marquardt algorithm	Moré et al. (1980, 1984)
NAG	several algorithms for constrained and unconstrained optimization problems, including nonlinear least-squares	NAG (2006)
PORT	several algorithms for unconstrained and bound-constrained optimization problems, nonlinear least-squares	Fox et al. (1978); Gay (1990)
TENSOLVE	nonlinear least-squares optimization by tensor methods	Bouaricha and Schnabel (1997)
TENMIN	general unconstrained optimization by tensor methods	Chow et al. (1994)
UNCMIN	various methods for general unconstrained optimization	Schnabel et al. (1985)

3.3 Easy Integration of User's Simulation Software

In computational science and engineering is not uncommon to consider varying mathematical models, η, to analyze some complicated phenomenon of interest. EFCOSS facilitates easy integration of arbitrary models η given in the form of a computer program by simply specifying its input and output variables. Currently, EFCOSS accepts programs written in Fortran, C, and C++. Conceptually, the model can be implemented in any language callable from C/C++. For instance, we recently integrated a model written in MATLAB. However, this integration process is currently not fully automated.

3.4 Seamless Integration of Derivatives

Many optimization algorithms rely on accurate derivative information on the objective function, which in turn requires the computation of derivatives of the model η. As noted above, in order to evaluate the objective functions for OED mentioned in Section 3.1, the Fisher matrix, $\mathbf{F}$, (or its inverse), needs to be computed, requiring the computation of $\partial\eta/\partial\boldsymbol{\theta}$. Hence, in order to evaluate the derivative of the objective function, second-order derivative information on η is needed.

Since η is given in the form of a computer program, EFCOSS employs *automatic differentiation* (Rall, 1981; Griewank, 2000) to compute the desired derivatives. Depending on the implementation language of the model function η, the automatic differentiation tools ADIFOR (Bischof et al., 1996; Fagan and Carle, 2000; Abate et al., 1997) or ADIC (Bischof et al., 1997) are used to transform the source code of η into new code capable of computing first- and second-order derivatives of η with respect to certain input parameters specified by the user.

The automatically generated computer code for the derivative computation is integrated in a way similar to the integration of the model η itself. In particular, the actual execution of the derivative code, including initialization, as well as the computation of the Fisher matrix are completely automated by EFCOSS.

3.5 Flexible Choice of Experimental Conditions and Model Parameters

The set of actual free variables during an optimization process is typically a subset of the model parameters, $\boldsymbol{\theta}$, or the parameters describing the experimental conditions, $\mathbf{x}$. In order to support incremental model-based design

and analysis of experiments, the set of free variables can be changed without touching the model η. A flexible initialization mechanism is able to immediately reflect changes in the currently relevant sets $\mathbf{x}$ and $\boldsymbol{\theta}$, such that only the desired derivatives of η are computed.

3.6 Interactive User Interface via Python

An optimization process carried out in EFCOSS, for instance for optimal experimental design, is controlled via sessions in the scripting language Python. This language has a clear and intuitive syntax and, moreover, its interpreter offers a high degree of interactivity. For the sake of simplicity, we consider a single Python session in this note. In general, such a Python session for EFCOSS has the following overall structure:

(A) Initialize EFCOSS
(B) Setup simulation server (including derivatives)
(C) Setup objective function (and optionally constraints)
(D) Setup optimization
(E) Start optimization

In the sequel, we give actual examples of each of these steps of a Python session using case studies arising from diffusion experiments.

4 A Binary Diffusion Experiment

The efficient measurement of diffusion coefficient in liquids remains one of the major challenges in transport phenomena (Bird, 2004). Current experimental techniques are often laborious and time-consuming. Diffusion experiments therefore present a sound target for optimal experimental design. EFCOSS is demonstrated here in the context of optimal experimental design for the estimation of diffusion coefficients. We consider two case studies taken from Bardow (2004) where a novel approach for the model-based experimental analysis of multicomponent diffusion is introduced.

As explained above, the design of an experiment for reliable parameter estimation can be formulated as an optimization problem involving some scalar measure on the Fisher information matrix. In this section, we briefly sketch the modeling of an experiment for the measurement of diffusion in a binary mixture. In the following two sections, we show how the functionality provided by EFCOSS simplifies the setup and numerical solution of the experimental design. We demonstrate the integration of a new simulation code into EF-

COSS, the flexible problem formulation including the choice of experimental conditions and model parameters using different OED criteria. We also show how to switch between optimization software packages.

The considered diffusion experiment (Bardow et al., 2003; Bardow, 2004; Bardow et al., 2005, 2006) is conducted in a vertical column, as depicted in Fig. 2. At the beginning of the experiment, a stable layering of two mixtures of different composition is generated. The concentration profiles will then equilibrate over time due to diffusion. The one-dimensional diffusion process is observed in a part of the diffusion cell using 1D-Raman spectroscopy. The concentration profiles obtained from the measured Raman spectra are considered as primary experimental data. The parameters describing the experimental setup are:

- the total height of the liquid in the cell, L
- the initial height of the lower phase, z_0
- the molar volume of the two components, denoted by V_α and V_β
- the initial mole fraction of component α in the lower and upper part of the cell, x_L and x_U, respectively
- the position of the sensor, z_{sensor}, defining the lower bound of the measurement zone
- the total height of the sensor, L_{sensor}
- the spatial resolution of the sensor, n_{sensor}, defining the number of data observed in the measurement zone at one time
- a number of fixed points in time, $t_1, \ldots, t_{\max}$, specifying when the measurements are taken.

All these parameters could be regarded as free variables, except for the total height of the sensor L_{sensor}, its resolution n_{sensor}, and the molar volumes V_α and V_β, which are fixed. In order to complete the model for the diffusion experiment, appropriate values of the diffusion coefficients need to be provided, which could be obtained by parameter estimation. In the following studies, the goal is to design an experiment which minimizes the uncertainty in the result of this parameter estimation. Two cases from Bardow (2004) are selected, assuming either constant or concentration-dependent diffusion coefficients.

5 EFCOSS for Constant Diffusivities

Assuming a constant diffusion coefficient D, the concentration c of component α at location z and time t is given by (Crank, 1975)

$$c(z,t) = (c_L - c_U)\left[\frac{z_0}{L} + \frac{2}{\pi}\sum_{k=1}^{\infty}\frac{1}{k}\sin\left(\frac{k\pi z_0}{L}\right)\exp\left(-\frac{Dk^2\pi^2 t}{L^2}\right)\cos\left(\frac{k\pi z}{L}\right)\right] + c_U, \tag{6}$$

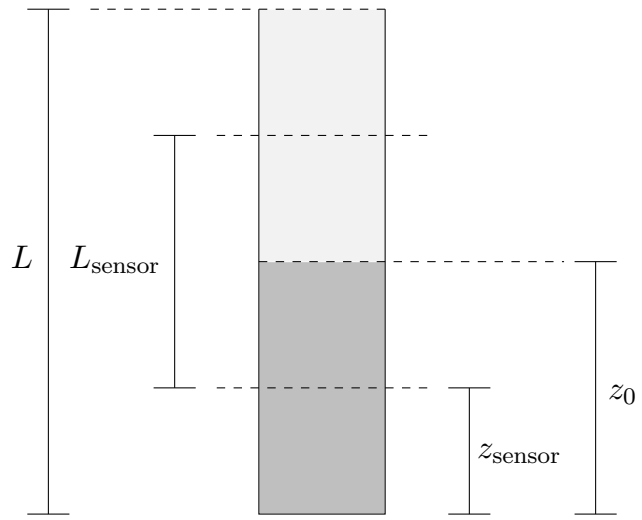

Fig. 2. Schematic of a measuring cell for binary diffusion experiments

where c_L and c_U are the initial concentration in the lower and upper part of the measuring cell, respectively. This means, the initial concentration $c(z,t)$ of component α at time $t = 0$ is c_L for $0 < z \leq z_0$, and c_U for $z_0 < z < L$. Given the molar volume of both components, and the initial mole fraction of component α in the lower part of the cell, the initial concentration, c_L, in the lower part of the measuring cell is calculated by

$$c_L = \frac{x_L}{V_\beta + x_L(V_\alpha - V_\beta)}.$$

The initial concentration of component α in the upper part of the cell, c_U, can be determined in an analogous way.

Using the general notation from Section 2, a model for the diffusion experiment using a constant diffusion coefficient D could be defined via

$$\eta(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{c} \in \mathbb{R}^{n_{\text{sensor}}},$$

where $\mathbf{x} = (L, L_{\text{sensor}}, z_{\text{sensor}}, z_0, t, V_\alpha, V_\beta, x_L, x_U, n_{\text{sensor}})^T$, $\boldsymbol{\theta} = (D)$, and $\mathbf{c} = (c_1, \ldots, c_{n_{\text{sensor}}})^T$ is the concentration of component α at n_{sensor} equidistant locations within the measurement zone at a given time t. More precisely,

$$c_i := c(z_i, t) \quad \text{where} \quad z_i := z_{\text{sensor}} + \frac{(i-1)L_{\text{sensor}}}{n_{\text{sensor}} - 1} \qquad \forall\, i = 1, \ldots, n_{\text{sensor}}.$$

For practical reasons, the model represents only one measurement at a given time t, rather than a series of measurements. In order to simulate a whole experiment, consisting of measurements at times $t_1, \ldots, t_{\max}$, the model η is evaluated multiple times.

[Steps (A) and (B)] Here we would like to design an experiment for estimating the diffusion parameter D with minimal uncertainty, where the measurement times are regarded as experimental degrees of freedom, i.e., we are looking for an optimal sequence of points in time $t_1, \ldots, t_{\max}$. Consider a Fortran subroutine

```
diffusion(D,L,Lsensor,zsensor,z0,t,V,xL,xU,nsensor,c)
```

implementing the model η, where the infinite sum in (6) is approximated using $k = 1, \ldots, 256$. In order to use this code with EFCOSS, the following subtasks of step (B) of the Python session are performed to set up the simulation server:

(1) The user specifies the routine implementing the model, i.e., its name, and the input and output variables. In addition, default values for the input variables are provided.
(2) Since the simulation is an ingredient to an optimization problem to be specified later, the user specifies a subset of the simulation's input variables with respect to which derivatives are required. In a parameter estimation problem, derivatives of the model η w.r.t. $\boldsymbol{\theta}$ or a subset thereof, are needed. In OED, the derivatives of η w.r.t. (a subset of) the variables in $\mathbf{x}$ and $\boldsymbol{\theta}$ are computed. This way, the set of potentially free variables in an optimization problem, denoted by `OptVars`, is specified. In the present case study, we consider the model parameter, D, and the experimental design variable, t.
(3) Based on the above specification, control files for a selected automatic differentiation tool are generated by EFCOSS, using `OptVars` as the set of independent variables and all output variables as dependent variables. Furthermore, EFCOSS generates interface code for evaluating the model and the first- and second-order derivatives of the model with respect to the variables contained in `OptVars`. In the present example, we employ ADIFOR version 3 (Fagan and Carle, 2000) for automatic differentiation of the simulation code.

The initialization step (A) and the above three subtasks of step (B) are performed within the Python shell using the script given in Fig. 3.

After execution of this Python script, the user runs the automatic differentiation tool in order to obtain the required derivative code. Then, the original subroutine and the derivative code are compiled together with the generated interfaces in order to obtain the simulation server executable, named "diffusion". To link the Python session to the simulation server "diffusion" the user enters

```
setSimulationServer("diffusion")
```

in the Python shell which completes step (B).

```
### (A) Initialize EFCOSS
from efcoss_utilities import *
from efcoss_codegen import *
getBrokerRef("broker1")

### (B) Setup simulation server
# specification of input variables
D       = newInputVar("D", 0.01)
L       = newInputVar("L", 25.0)
Lsensor = newInputVar("Lsensor", 10.0)
zsensor = newInputVar("zsensor", 0.0)
z0      = newInputVar("z0", 10.9275)
t       = newInputVar("t", 6.0)
vol     = newInputVar("V", [98.5071, 108.77])
xL      = newInputVar("xL", 1.0)
xU      = newInputVar("xU", 0.0)
nsensor = newInputVar("nsensor", 401)

# specification of output variables
c = newOutputVar("c",[nsensor])

# define the ordering of the variables in
# the calling sequence of the simulation
setSimulationCallingSequence([D,L,Lsensor,zsensor,z0,
                              t,vol,xL,xU,nsensor,c])

# specification variables for which derivatives are required
setOptVars([D,t])

TopLevelRoutine="diffusion"
AD_Tool="adifor3"

# generation of control files for AD tool
generateADControlFile(TopLevelRoutine,AD_Tool)

# generation of CORBA interfaces
generateServerIF(TopLevelRoutine,"diffusionIF.cc",AD_Tool)
```

Fig. 3. Problem setup for the simulation with constant diffusivities

[Step (C)] As a first subtask of step (C), a suitable objective function must be selected. For the case of a single unknown parameter, the Fisher information matrix reduces to a scalar so that the D-, A-, and E-optimality criteria coincide.

In the case study, we choose the D-optimality criterion, where the model parameter θ is the diffusion coefficient, D, and the design parameters $\mathbf{x}$ are the times $t_1, \ldots, t_{\max}$ when the samples are taken. These measurement times are given in hours. We start with an initial design consisting of $n = 24$ samples which are evenly distributed in the interval $[0.5, \ldots, 12.0]$. This corresponds to an experiment where one measurement is performed every half hour. That is, we formally consider $\mathbf{x}_i = (t_i) \in \mathbb{R}^m$ with $m = 1$ and $\mathbf{X} = (t_1, \ldots, t_{24})^T$. The selection of the objective function and the initial design with $N = n \cdot m$ scalar values is given by the following statements:

```
# define objective function
obj = setObjectiveFunction("OED","D_opt")
obj._set_model_parameters([D])
obj._set_design_parameters([t])
# initial design
N = 24
X = (arrayrange(N)*0.5)+0.5
for val in X:
    obj.addDesign([t],[val],1.0)
```

The `arrayrange()` function is part of the Numerical Python package which is implicitly loaded when importing the `efcoss_utilities` module. In the above code this function is used to create an array of 24 integers $[0, 1, \ldots, 23]$ which is scaled with 0.5 before an element-wise addition of the value 0.5 is performed. The methods `addDesign()`, `_set_model_parameters()`, and `_set_design_parameters()` are members of an `OED` objective class.

The experimental setup requires a minimum time of 60 seconds between subsequent measurements. This condition is formulated by linear inequality constraints:

$$t_{i+1} - t_i \geq 1/60 \quad \forall i = 1, \ldots, 23.$$

The constraints are specified in Python via the following code:

```
A = zeros((N-1,N),Float64)
for i in range(N-1):
    A[i][i] = -1
    A[i][i+1] = 1
b = (1.0/60)*ones((N-1),Float64)
setLinearConstraints(A,b,[],[])
```

Here, we make again use of the array type and array-generating functions provided by the Numerical Python package. The function `setLinearConstraints()` is used to specify the constraint relation $A\mathbf{X} \geq \mathbf{b}$. Handling nonlinear constraints is also implemented.

[Steps (D) and (E)] Thereafter, we want to optimize the initial experimental design, i.e., we are seeking 24 measurement times such that the variance of the model parameter, D, is minimized. In this case study, we choose the FFSQP code (Lawrence and Tits, 1996) to perform the optimization, using the above specified objective function. Provided with appropriate control parameters, the FFSQP optimizer, which is integrated into EFCOSS, can be accessed from within the Python shell.

```
### (D) Setup optimization
from ffsqp import *

# setting of specific control parameters and
# work arrays for FFSQP omitted (see Appendix)

# set bound constraints
bl=array([ 0.0]*N,Float64)    # lower bounds
bu=array([24.0]*N,Float64)    # upper bounds

### (E) Start optimization
inform = efcoss_ffsqp(nf,nineqn,nineq,neqn,neq,mode,
                      iprint,miter,bigbnd,eps,epseqn,
                      udelta,bl,bu,X,f,g,iw,w)

print "result vector:",X
print "inform:",inform
```

Fig. 4. Setup and execution of the FFSQP optimization algorithm

The corresponding code representing steps (D) and (E) is given in Fig. 4. First, the control parameters and work arrays specific to FFSQP are initialized, which is omitted here for the sake of brevity. For details we refer to the appendix. Then, upper and lower bound constraints for all `N` unknowns are defined, using Numerical Python arrays. Here, the upper bound constraints restrict the total duration of the overall experiment to 24 hours maximum. Finally, the FFSQP optimizer is invoked, and its results are printed.

With the FFSQP-specific control parameter `eps` set to 10^{-2}, the solution is found by FFSQP after 54 iterations. The resulting vector of 24 measurement times, as well as the corresponding Fisher information is graphically displayed in Fig. 5. Every circle represents a point of time when a measurement is performed. Apparently, the majority of the measurement times, t_1 to t_{16}, tend to be distributed around some value between 1.6 and 1.85, precisely obeying the minimum required difference of $1/60 = 0.01\overline{6}$ between two consecutive measurement points. The values for t_{17} to t_{24} lying between 6.0 and 12.0 might just not been sufficiently converged.

In order to check this, we repeat the optimization process with a tighter termi-

nation criterion. Here, we change the value of the control parameter `eps` from 10^{-2} to 10^{-4}. With this setting, the algorithm terminates after 127 iterations. The corresponding solution, represented by the crosses in Fig. 5, clearly shows the expected behavior. The zoom into the region 1.5 h $< t <$ 1.95 h shows that the solution for `eps` $= 10^{-4}$ is now better converged and that the Fisher information has a maximum at $t_{12} = 1.72$ h. All remaining measurement times cluster around this maximum value.

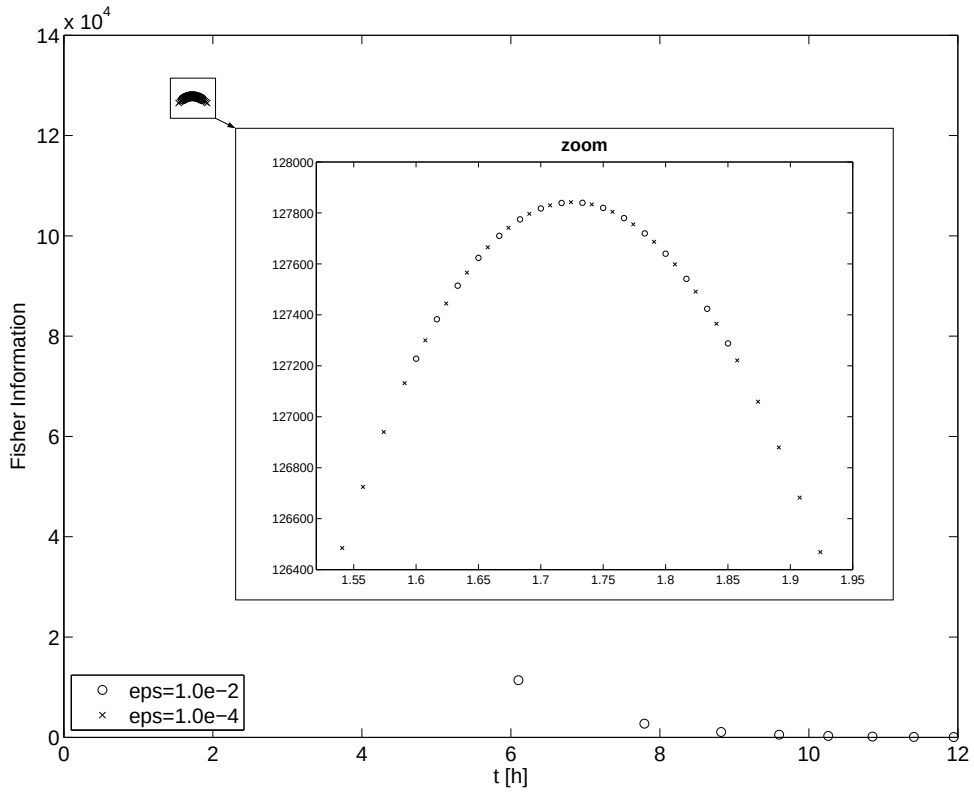

Fig. 5. Distribution of the optimal measurement times $t_1, \ldots, t_{24}$ and corresponding Fisher information, obtained by FFSQP with control parameter `eps`=10^{-2} (circles) and `eps`=10^{-4} (crosses). The picture contains an enlarged view of the area of interest, where 1.5 h $< t <$ 1.95 h.

The optimal design for a nonlinear problem depends on the initial guess for the unknown value of the diffusion coefficient D itself. In order to verify the robustness of the obtained result, we therefore consider only one measurement, but vary the value of the diffusion coefficient, D. As initial design, we consider $t = 0.1$ h:

```
oed.clearDesign()
oed.addDesign(design_parameters,[0.1],1.0)
```

Again, we employ FFSQP, using the same settings for the control parameters

as before, but with the number of free variables, `N`, set to 1. The following code determines the optimal measurement time for three different values of D:

```
diffusion_coefficients = [ 0.1 , 0.01 , 0.001 ]
for dc in diffusion_coefficients:
    setDefaultValue( D , dc)
    X = array(getInitialValues(),Float64)
    efcoss_ffsqp(...)  # invoke FFSQP algorithm
    print "D =",dc , " opt. measurement time =",X
```

The corresponding output from the `print` statement is given below:

```
D = 0.1    opt. measurement time = [ 0.17256821]
D = 0.01   opt. measurement time = [ 1.7255123]
D = 0.001  opt. measurement time = [ 17.25512684]
```

This result shows a close relationship, $D \sim 1/t$, between the assumed value of the diffusion coefficient, D, and the optimal measurement time, t. A more detailed analysis shows that the optimal diffusion experiment is indeed characterized through an optimal Fourier time $\mathrm{Fo} = Dt/L^2$ (Bardow, 2004; Bardow et al., 2006).

The case study demonstrated how foreign simulation code is integrated into EFCOSS and exemplifies the flexibility the environment provides with respect to problem analysis.

6 EFCOSS for Concentration-Dependent Diffusivities

The previous case study focused on constant diffusion coefficients. Recently, it has been shown that the concentration-dependence of a diffusion coefficient can be derived directly from a single experiment (Bardow et al., 2005; Kriesten et al., 2008). In this section, we therefore study the estimation of a concentration-dependent diffusion coefficient. Following Bardow (2004) the concentration c is given by

$$\frac{\partial c}{\partial t} = \frac{\partial}{\partial z}\left(D(x,\boldsymbol{\theta})\frac{\partial c}{\partial z}\right) \tag{7}$$

with

$$x = \frac{cV_\beta}{1-(V_\alpha - V_\beta)c}, \quad x(z,0) = x_0(z),$$

and

$$\frac{\partial c(0,t)}{\partial z} = \frac{\partial c(L,t)}{\partial z} = 0.$$

The diffusion coefficient, $D = D(x, \boldsymbol{\theta})$, is modeled by a quadratic polynomial

$$D(x, \boldsymbol{\theta}) = \theta_1 x + \theta_2(1 - x) + \theta_3 x(1 - x). \tag{8}$$

In a parameter identification problem, one would try to find values for the coefficients of the quadratic polynomial, $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$, such that the simulation output best matches the experimental results at given points of time $t_1, \ldots, t_{\max}$. In an OED problem, we are interested in optimizing the experimental conditions such that the parameter identification problem can be solved with minimal uncertainty, i.e., the variance in the estimated parameters should be minimized. We perform the same steps as in the previous section, where only the problem-specific parts are slighly different.

[Steps (A) and (B)] Similar to Section 5, a computer program implementing the model is considered. Again, the simulation is written in Fortran, now using the PDECOL package (Madsen and Sincovec, 1979) for solving the partial differential equation (7). The scalar input parameters of the top-level subroutine, called `diffusion_con()`, include L, L_{sensor}, z_{sensor}, z_0, x_L, x_U, and n_{sensor}, as in Section 5. Furthermore, the routine `diffusion_con()` takes two double precision arrays, `vol` and `theta`, of length 2 and 3, respectively, as input parameters. Here, `vol(1:2)` contains the molar volume of the two components, as in Section 5, and the entries in `theta(1:3)` represent the three coefficients of the quadratic polynomial (8). In contrast to Section 5, the simulation code computes the concentration for all points in time in a one-shot fashion. More precisely, for given values n_t and $t_{\max}$ specifying the number of measurements in time, and the maximum duration of the experiment, respectively, the subroutine `diffusion_con()` computes an $n_{\text{sensor}} \times n_t$ matrix c, where the entry $c(i, j)$ corresponds to the concentration at sensor location n_i and time t_j, and the t_j are evenly distributed in the interval $[0, t_{\max}]$.

[Steps (C), (D) and (E)] In this case study, we are interested in optimizing the design variables z_{sensor} and $t_{\max}$ representing the position of the sensor and the total duration of the experiment, respectively. Formally, this corresponds to $n = 1$ and $\mathbf{X} = \mathbf{x} = (z_{\text{sensor}}, t_{\max})^T$. The corresponding Python script for the problem specification using the D-optimality criterion with initial design $\mathbf{X} = (10.0, 4.0)^T$ is given in Fig. 6. For the optimization, the general minimization routine with simple bound constraints, `dmngb()`, from the PORT library (Fox et al., 1978) is employed. An excerpt from the Python script driving the optimization is given in Fig. 7. Here, the lower bounds for z_{sensor} and $t_{\max}$ are 10^{-3} mm and 0.5 h, respectively. The upper bounds for z_{sensor} and $t_{\max}$ are 15 mm and 24 h, respectively. For the control parameters of the optimization routine, the default values mentioned in the PORT user's manual (Gay,

1990) are used, except for the relative function-convergence tolerance (parameter #32) and the X-convergence tolerance (parameter #33) which are both set to 10^{-6}. The result obtained after 10 iterations is $\widehat{\mathbf{X}} = [15.0, 4.28468065]$, corresponding to an experimental design with

$$z_{\text{sensor}} = 15.0 \text{ mm} \qquad \text{and} \qquad t_{\max} \approx 4.2847 \text{ h}.$$

```
### (A) Initialize EFCOSS
from efcoss_utilities import *
from efcoss_codegen import *
getBrokerRef("broker1")

### (B) Setup simulation server
# specification of input variables
theta   = newInputVar("theta",[2.44E-3,1.52E-3,-0.52E-3] )
L       = newInputVar("L",   25.0)
Lsensor = newInputVar("Lsensor", 10.0)
zsensor = newInputVar("zsensor", 15.0)
z0      = newInputVar("z0",   17.24)
vol     = newInputVar("V", [ 106.2756632 , 108.0385109 ] )
tmax    = newInputVar("tmax",  6.0)
xL      = newInputVar("xL",  1.0)
xU      = newInputVar("xU",  0.0)
nsensor = newInputVar("nsensor", 401)
nt      = newInputVar("nt",  26)

# specification of output variables
c       = newOutputVar("c",[nsensor,nt])

# definition of the ordering of variables in the simulation
setSimulationCallingSequence([theta,L,Lsensor,zsensor,z0,
                              tmax,vol,xL,xU,nsensor,nt,c])

# specification variables for which derivatives are required
setOptVars([theta,zsensor,tmax])

# generaton of control files for AD and CORBA interfaces
TopLevelRoutine="diffusion_con"
AD_Tool="adifor3"
generateADControlFile(TopLevelRoutine,AD_Tool)
generateServerIF(TopLevelRoutine,TopLevelRoutine+"IF.cc",AD_Tool)

### (C) Setup objective function
# specify objective function and initial design
obj = setObjectiveFunction("OED","D_opt")
obj._set_simulation_result([c])
obj._set_design_parameters([zsensor,tmax])
obj._set_model_parameters([theta])
obj.addDesign([zsensor,tmax], [ 10.0 , 4.0 ], 1.0)
```

Fig. 6. Problem setup for the simulation with concentration-dependent diffusivities

```
### (D) Setup optimization
import port

# setting of specific control parameters and
# work arrays for PORT omitted (see Appendix)

# set bound constraints for [ zsensor, tmax ]
lower_bound = [  0.001 ,  0.5 ] # lower bounds
upper_bound = [ 15.0   , 24.0 ] # upper bounds
b = array([lower_bound,upper_bound],Float64)

# modify default values (list indexing starts with 0)
v[32-1] = 1.0e-6 # relative function-convergence tolerance
v[33-1] = 1.0e-6 # X-convergence tolerance

### (E) Start optimization
port.efcoss_dmngb(d,X,b,iv,v,ui,ur)

print "result vector:",X
```

Fig. 7. Setup and execution of the PORT 3 routine MNGB (double precision version)

In a more rigorous analysis we carry out the optimization again, with different initial values for $\mathbf{X}$ and also other optimization criteria mentioned in Section 3.1. Practically, this requires only slight modifications in section (C) of Fig. 6. For instance, the user can switch to the A-optimality criterion with an initial design $\mathbf{X} = (10.0, 2.0)^T$ by the following two statements

```
obj = setObjectiveFunction("OED","A_opt")
obj.addDesign([zsensor,tmax], [ 10.0 , 2.0 ], 1.0)
```

replacing the corresponding ones in that figure. The results are summarized in Table 2.

For three different initial configurations, the PORT optimization routine determines similar D-optimal designs $\widehat{\mathbf{X}}$, indicating a robust solution. Using the A-optimality criterion with initial values $\mathbf{X_0} = [10.0, 2.0]$, the PORT optimization routine terminates after 1 iteration with no significant improvement. So we changed the X-convergence tolerance (parameter #33) from 10^{-6} to 10^{-7} for this initial configuration. With this setting, however, the optimizer terminates with the exit message *false convergence* after 19 iterations. For the other initial values, i.e., $\mathbf{X_0} = [10.0, 4.0]$ and $\mathbf{X_0} = [10.0, 6.0]$, normal termination occurs after 15 and 23 iterations, respectively. Since the A-optimal solutions $\widehat{\mathbf{X}}$ for the various configurations differ significantly, these results should be faced with some scepticism. In the bottom section of Table 2 the E-optimal solu-

Table 2
Results obtained by the MNGB optimization routine from the PORT library. The symbol $(*)$ indicates the termination message *false convergence* reported by the optimizer.

Initial values $\mathbf{X_0}$	Optimality-criterion	Number of iterations	Optimal values $\widehat{\mathbf{X}}$	Final value of the cost function
$[10.0, 2.0]$	D	11	$[15.0, 4.30264556]$	-43.2
$[10.0, 4.0]$	D	10	$[15.0, 4.28468065]$	-43.2
$[10.0, 6.0]$	D	8	$[15.0, 4.29087691]$	-43.2
$[10.0, 2.0]$	A	1	$[9.99999972, 1.99999001]$	0.376527e-04
$[10.0, 2.0]$	A	$19(*)$	$[10.01018374, 1.05496558]$	0.321131e-04
$[10.0, 4.0]$	A	15	$[10.45870127, 0.86274301]$	0.311027e-04
$[10.0, 6.0]$	A	23	$[15.0, 2.0525141]$	0.182432e-04
$[10.0, 2.0]$	E	$26(*)$	$[12.2, 0.66394406]$	-38194
$[10.0, 4.0]$	E	10	$[15.0, 2.04309667]$	-55739
$[10.0, 6.0]$	E	10	$[15.0, 2.04302172]$	-55739

tions found by the PORT optimizer are shown. Again, for $\mathbf{X_0} = [10.0, 2.0]$, the optimizer terminates abnormally after 26 iterations. So, this result might not be reliable. For $\mathbf{X_0} = [10.0, 4.0]$ and $\mathbf{X_0} = [10.0, 6.0]$ one consistent E-optimal solution $\widehat{\mathbf{X}}$ is obtained.

The observed difficulties in convergence for several cases emphasize the need for a flexible simulation and optimization environment. Due to their complexity OED optimization results should be carefully analyzed. This analysis can be further supported by EFCOSS. As an example, the EFCOSS function

```
F = obj.Fisher( X )
```

can be used to compute the Fisher information matrices for the various optimal designs in order to compare the quality of the results. Let $\mathbf{F}_D$, $\mathbf{F}_A$, and $\mathbf{F}_E$ denote the Fisher matrices for the D-, A-, and E-optimal designs, which are computed by the PORT optimizer with the initial configuration $\mathbf{X_0} = [10.0, 4.0]$. By a further analysis, it turns out that the solution computed by the PORT optimizer using the A-optimality criterion is not A-optimal compared to the other solutions. More precisely, $\mathrm{tr}(\mathbf{F}_E^{-1}) < \mathrm{tr}(\mathbf{F}_D^{-1}) < \mathrm{tr}(\mathbf{F}_A^{-1})$. That is, applying the A-optimality criterion to $\mathbf{F}_D$ and $\mathbf{F}_E$ yields a better result than applying it to $\mathbf{F}_A$. Such a situation can occur because the gradient-based optimization algorithm employed here computes only a local optimum rather than a global one.

Since there are still reasonable doubts about the quality of the results, in particular for the case of A-optimal design, we switch to another optimizer, namely DONLP2 (Spellucci, 1998a,b), and perform the optimization tasks again. The corresponding results, using only $\mathbf{X_0} = [10.0, 4.0]$ as initial values, are displayed in Table 3.

Table 3
Results obtained by the DONLP2 optimizer.

Initial values $\mathbf{X_0}$	Optimality-criterion	Number of iterations	Optimal values $\widehat{\mathbf{X}}$	Final value of the cost function
$[10.0, 4.0]$	D	13	[15.0 , 4.28469791]	-43.2
$[10.0, 4.0]$	A	7	[15.0 , 2.05006086]	0.182433E-04
$[10.0, 4.0]$	E	14	[15.0 , 2.0430214]	-55739

For the D- and E-optimality criteria, the results obtained by DONLP2 confirm the optimal designs computed before by the PORT optimizer; cf. Table 2. If the A-optimality criterion is employed, the solution found by DONLP2 is significantly different from the PORT solution for the corresponding initial configuration, but similar to the PORT solution for the initial configuration $\mathbf{X_0} = [10.0, 6.0]$.

Noteworthy, all optimal designs of the experiment place the measurements zone at one end of the diffusion cell. This is in strong contrast to the common practice where so-called infinite diffusion cells are employed and the wall region is strictly avoided (Miller et al., 1993). The advantage of using restricted diffusion cells predicted by the OED calculations has already been successfully exploited in practice (Bardow et al., 2003, 2005, 2006).

7 Conclusions

In optimal experimental design, the design of an experiment for reliable parameter estimation is formulated as an optimization problem whose objective function consists of a scalar measure of the Fisher information matrix. EFCOSS is an interactive, component-based, distributed software environment that supports optimal experimental design by allowing its user to choose from a set of several widely-used optimality criteria or to easily implement new optimality criteria. Since the Fisher information matrix involves the first-order derivative of the underlying mathematical model with respect to the model parameters, a gradient-based algorithm to optimize the experimental conditions needs the second-order derivatives of the model. By seamlessly interfacing to several software tools for automatic differentiation, EFCOSS is capable of efficiently and accurately evaluating the objective function. To this end, EFCOSS supports the formulation of the objective function, the integration of the user's code of the underlying mathematical model, and the specification of the variables representing model parameters and experimental conditions. From within an interactive Python session, the user can then choose from a set of different optimization algorithms without the tedious and error-prone task to manually adapt the code implementing the mathematical model to different optimization codes. Through its component-based software architecture, EFCOSS makes available the concepts of optimal experimental design to computational scientists with little human effort.

The case studies on diffusion experiments exemplify the flexibility of the EFCOSS environment. The predicted improvements through optimized experiments have already been proven in practice in a novel experimental setup employing one-dimensional Raman spectroscopy (Bardow et al., 2003, 2005, 2006).

Nomenclature

$c(z,t)$	concentration of component α at location z and time t
c_L and c_U	initial concentration in lower and upper part of measuring cell
D	diffusion coefficient
E	expectation operator
$\mathbf{F}$	Fisher information matrix
$\mathbf{F}_D$, $\mathbf{F}_A$, and $\mathbf{F}_E$	Fisher information matrices for D-, A-, and E-optimal designs
L	height of liquid in cell
L_{sensor}	total height of sensor
$\mathcal{L}$	logarithm of likelihood function
n	number of experimental conditions
n_{sensor}	spatial resolution of sensor
n_t	number of measurements in time
Q_i	matrix of model derivatives wrt parameters for i-th experiment
$\mathbb{R}$	set of real numbers
$t_1, \ldots, t_{\max}$	points in time specifying when measurements are taken
V_α and V_β	molar volume of components α and β
V_i	covariance matrix for measurement errors of i-th experiment
V_θ	covariance matrix of model parameters
x_L and x_U	initial mole fraction of component α in lower and upper part of cell
$\mathbf{x}$	explanatory variables
$\mathbf{X}$	set of explanatory variables
$\mathbf{X_0}$	initial values for set of explanatory variables
$\widehat{\mathbf{X}}$	optimal values for set of explanatory variables
z_{sensor}	position of sensor
z_0	initial height of lower phase
η	model
Φ	scalar measure function on Fisher matrix
$\boldsymbol{\theta}$	model parameters
$\boldsymbol{\theta}^*$	true values of model parameters
σ_i^2	variance for measurement errors of i-th experiment

Acknowledgements

This research is partially supported by the Deutsche Forschungsgemeinschaft (DFG) within SFB 540 "Model-based experimental analysis of kinetic phenomena in fluid multi-phase reactive systems," RWTH Aachen University, Germany. The Aachen Institute for Advanced Study in Computational Engineering Science (AICES) provides a stimulating research environment for our work.

References

Abate, J., Bischof, C., Carle, A., Roh, L., 1997. Algorithms and design for a second-order automatic differentiation module. In: Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computing (ISSAC'97), Maui, Hawaii, USA, July 21–23, 1997. ACM Press, New York, pp. 149–155.

Arellano-Garcia, H., Schöneberger, J., Körkel, S., 2007. Optimale Versuchsplanung in der chemischen Verfahrenstechnik. Chemie Ingenieur Technik 79 (10), 1625–1638.

Atkinson, A. C., Donev, A. N., 1992. Optimum Experimental Designs. Vol. 8 of Oxford Statistical Science Series. Oxford University Press, Oxford, UK.

Baltes, M., Schneider, R., Sturm, C., Reuss, M., 1994. Optimal experimental design for parameter estimation in unstructured growth models. Biotechnology Progress 10 (5), 480–488.

Bardow, A., 2004. Model-based Experimental Analysis of Multicomponent Diffusion in Liquids. Vol. 821 of Fortschritt-Berichte VDI: Reihe 3. VDI-Verlag, Düsseldorf.

Bardow, A., 2008. Optimal experimental design of ill-posed problems: The METER approach. Computers & Chemical Engineering 32 (1-2), 115–124.

Bardow, A., Göke, V., Koß, H. J., Lucas, K., Marquardt, W., 2005. Concentration-dependent diffusion coefficients from a single experiment using model-based Raman spectroscopy. Fluid Phase Equilibria 228, 357–366.

Bardow, A., Göke, V., Koß, H. J., Marquardt, W., 2006. Ternary diffusivities by model-based analysis of Raman spectroscopy measurements. AIChE Journal 52 (12), 4004–4015.

Bardow, A., Marquardt, W., Göke, V., Koß, H. J., Lucas, K., 2003. Model-based measurement of diffusion using Raman spectroscopy. AIChE Journal 49 (2), 323–334.

Bauer, I., Bock, H. G., Körkel, S., Schlöder, J. P., 2000. Numerical methods for optimum experimental design in DAE systems. Joural of Computational and Applied Mathematics 120 (1-2), 1–25.

Benabbas, L., Asprey, S., Macchietto, S., 2005. Curvature-based methods for

designing optimally informative experiments in multiresponse nonlinear dynamic situations. Industrial & Engineering Chemistry Research 44 (18), 7120–7131.

Bernaerts, K., Servaes, R. D., Kooyman, S., Versyck, K. J., Van Impe, J. F., 2002. Optimal temperature input design for estimation of the square root model parameters: parameter accuracy and model validity restrictions. International Journal of Food Microbiology 73 (2-3), 145–157.

Bird, R. B., 2004. Five decades of transport phenomena. AIChE Journal 50 (2), 273–287.

Bischof, C., Carle, A., Khademi, P., Mauer, A., 1996. Adifor 2.0: Automatic differentiation of Fortran 77 programs. IEEE Computational Science & Engineering 3 (3), 18–32.

Bischof, C. H., Bücker, H. M., Lang, B., Rasch, A., 2003. An interactive environment for supporting the transition from simulation to optimization. Scientific Programming 11 (4), 263–272.

Bischof, C. H., Roh, L., Mauer, A., 1997. ADIC — An extensible automatic differentiation tool for ANSI-C. Software: Practice and Experience 27 (12), 1427–1456.

Bouaricha, A., Schnabel, R. B., 1997. Algorithm 768: TENSOLVE: A software package for solving systems of nonlinear equations and nonlinear least-squares problems using tensor methods. ACM Transactions on Mathematical Software 23 (2), 174–195.

Chow, T.-T., Eskow, E., Schnabel, R. B., 1994. Algorithm 739: A software package for unconstrained optimization using tensor methods. ACM Transactions on Mathematical Software 20 (4), 518–530.

Crank, J., 1975. The Mathematics of Diffusion. Clarendon Press, Oxford.

Fagan, M., Carle, A., 2000. Adifor 3.0 overview. Tech. Rep. CAAM–TR00–03, Rice University, Department of Computational and Applied Mathematics.

Fox, P. A., Hall, A. P., Schryer, N. L., 1978. The PORT mathematical subroutine library. ACM Transactions on Mathematical Software 4 (2), 104–126.

Franceschini, G., Macchietto, S., 2008. Model-based design of experiments for parameter precision: State of the art. Chemical Engineering Science 63 (19), 4846–4872.

Galvanin, F., Macchietto, S., Bezzo, F., 2007. Model-based design of parallel experiments. Industrial & Engineering Chemistry Research 46 (3), 871–882.

Gay, D. M., 1990. Usage summary for selected optimization routines. Computing Science Technical Report 153, AT&T Bell Laboratories, Murray Hill, NJ, USA.

Griewank, A., 2000. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. SIAM, Philadelphia, PA.

Körkel, S., 2002. Numerische Methoden für optimale Versuchsplanungsprobleme bei nichtlinearen DAE-Modellen. Doctoral thesis, University of Heidelberg.

Körkel, S., Kostina, E., Bock, H. G., Schlöder, J. P., 2004. Numerical methods for optimal control problems in design of robust optimal experiments for

nonlinear dynamic processes. Optimization Methods and Software 19 (3–4), 327–338.

Kriesten, E., Voda, M., Bardow, A., Göke, V., Casanova, F., Blümich, B., Koss, H.-J., Marquardt, W., 2008. Direct determination of the concentration dependence of diffusivities using combined model-based Raman and NMR experiments. Fluid Phase Equilibria(10.1016/j.fluid.2008.10.012).

Lawrence, C. T., Tits, A. L., 1996. Nonlinear equality constraints in feasible sequential quadratic programming. Optimization Methods and Software 6 (4), 265–282.

Lindström, P., 1982. A stabilized Gauss-Newton algorithm for unconstrained least squares problems. Tech. Rep. UMINF-102.82, Institute of Information Processing, University of Umeå, Umeå, Sweden.

Lindström, P., 1983. A general purpose algorithm for nonlinear least squares problems with nonlinear constraints. Tech. Rep. UMINF-103.83, Institute of Information Processing, University of Umeå, Umeå, Sweden.

Madsen, N. K., Sincovec, R. F., 1979. Algorithm 540: PDECOL, general collocation software for partial differential equations [D3]. ACM Transactions on Mathematical Software 5 (3), 326–351.

Miller, D. G., Albright, J. G., Mathew, R., Lee, C. M., Rard, J. A., Eppstein, L. B., 1993. Isothermal diffusion-coefficients of $NaCl - MgCl_2 - H_2O$ at 25°C. 5. Solute concentration ratio of 1:1 and some Rayleigh results. Journal of Physical Chemistry 97 (15), 3885–3899.

Moré, J. J., Garbow, B. S., Hillstrom, K. E., 1980. User guide for MINPACK-1. Tech. Rep. ANL–80–74, Argonne National Laboratory, Argonne, IL, USA.

Moré, J. J., Sorensen, D. C., Garbow, B. S., Hillstrom, K. E., 1984. The MINPACK project. In: Cowell, W. R. (Ed.), Sources and Development of Mathematical Software. Prentice-Hall series in computational mathematics. Prentice-Hall, Englewood Cliffs, NJ, USA, pp. 88–111.

NAG, 2006. NAG Library manual. The Numerical Algorithms Group Ltd., Oxford, UK.

Navarro-Laboulais, J., Cardona, S. C., Torregrosa, J. I., Abad, A., Lopez, F., 2008. Practical identifiability analysis in dynamic gas-liquid reactors: Optimal experimental design for mass-transfer parameters determination. Computers & Chemical Engineering 32 (10), 2382–2394.

Petzold, L., Li, S., Cao, Y., Serban, R., 2006. Sensitivity analysis of differential-algebraic equations and partial differential equations. Computers & Chemical Engineering 30 (10-12), 1553–1559.

Powell, M. D. J., 1994. A direct search optimization method that models the objective and constraint functions by linear interpolation. In: Gomez, S., Hennart, J.-P. (Eds.), Advances in Optimization and Numerical Analysis: Proceedings of the Sixth workshop on Optimization Numerical Analysis, Oaxaca, Mexico. Kluwer Academic Publishers, Dordrecht, NL, pp. 51–67.

PSE, 2004. gPROMS Advanced User Guide. Process Systems Enterprise Ltd., London, UK.

Rall, L. B., 1981. Automatic Differentiation: Techniques and Applications.

Vol. 120 of Lecture Notes in Computer Science. Springer Verlag, Berlin.
Rasch, A., Bücker, H. M., 2008. EFCOSS: An interactive environment facilitating optimal experimental design. Submitted for publication.
Schnabel, R. B., Koontz, J. E., Weiss, B. E., 1985. A modular system of algorithms for unconstrained minimization. ACM Transactions on Mathematical Software 11 (4), 419–440.
Sedrati, Y., Cabassud, M., Le Lann, M., Casamatta, G., 1999. Sequential experimental design strategy for kinetic parameters estimation. Computers & Chemical Engineering 23 (Suppl. S), S427–S430.
Spellucci, P., 1998a. A new technique for inconsistent QP problems in the SQP method. Mathematical Methods of Operations Research 47 (3), 355–400.
Spellucci, P., 1998b. An SQP method for general nonlinear programs using only equality constrained subproblems. Mathematical Programming 82 (3), 413–448.
Vlachos, D. G., Mhadeshwar, A. B., Kaisare, N. S., 2006. Hierarchical multiscale model-based design of experiments, catalysts, and reactors for fuel processing. Computers & Chemical Engineering 30 (10-12), 1712–1724.
Wächter, A., Biegler, L. T., 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming 106 (1), 25–57.
Walter, E., Pronzato, L., 1990. Qualitative and quantitative experiment design for phenomenological models – a survey. Automatica 26 (2), 195–213.
Walter, E., Pronzato, L., 1997. Identification of Parametric Models from Experimental Data. Communications and Control Engineering. Springer Verlag, Berlin.
Yang, A., Braunschweig, B., Fraga, E. S., Guessoum, Z., Marquardt, W., Nadjemi, O., Paen, D., Pinol, D., Roux, P., Sama, S., Serra, M., Stalker, I., 2008. A multi-agent system to facilitate component-based process modeling and design. Computers & Chemical Engineering 32 (10), 2290–2305.
Zhou, J. L., Tits, A. L., Lawrence, C. T., 1997. User's Guide for FFSQP Version 3.7 — A FORTRAN Code for Solving Constrained Nonlinear (Minimax) Optimization Problems. Institute for Systems Research, University of Maryland, College Park, MD, USA.
Zhu, C., Byrd, R. H., Lu, P., Nocedal, J., 1997. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization. ACM Transactions on Mathematical Software 23 (4), 550–560.

Appendix

```
### (D) Setup optimization
from ffsqp import *

mode=100       # algorithm
iprint=2       # level of information printed
miter=200      # max. number of iterations
bigbnd=1.0e+10 # infinite bound
eps=1.0e-02    # final norm requirement for Newton direction
epseqn=0.0     # max. violation of nonlinear equality cons.
udelta=0.0     # step size for approximating gradient
nf=1           # number of objective functions
nineqn=0       # number of nonlinear inequality constraints
nineq=N-1      # total number of inequality constraints
neqn=0         # number of nonlinear equality constraints
neq=0          # total number of equality constraints

# allocate memory for output variables
f = zeros((nf),Float64)   # values of objective functions
g = zeros((nineq+neq),Float64) # values of constraints

# size of work arrays
iwsize = 6*N + 8*max(1,nineq+neq) + 7*max(1,nf) + 30
nwsize = 4*N*N + 5*max(1,nineq+neq)*N + 3*max(1,nf)*N
         + 26*(N+max(1,nf)) + 45*max(1,nineq+neq) + 100

# allocate memory for work arrays
iw=zeros((iwsize),Int32)
w=zeros((nwsize),Float64)

# set bound constraints
bl=array([ 0.0]*N,Float64)    # lower bounds
bu=array([24.0]*N,Float64)    # upper bounds

### (E) Start optimization
inform = efcoss_ffsqp(nf,nineqn,nineq,neqn,neq,mode,
                      iprint,miter,bigbnd,eps,epseqn,
                      udelta,bl,bu,X,f,g,iw,w)
print "result vector:",X
print "inform:",inform
```

Fig. 8. Setup and execution of the FFSQP optimization algorithm (extended version of Fig. 4)

```
### (A) Initialize EFCOSS
from efcoss_utilities import *
import port
getBrokerRef("broker1")
setSimulationServer("diffusion_con")
initval = getInitialValues()
N = len(initval)

### (C)+(D) Setup PORT general optimization with simple bounds
liv = 59+(3*N)
lv = 78+N*(N+15)
iv = zeros((liv),Int32)
v = zeros((lv),Float64)
ui = zeros((1),Int32)
ur = zeros((1),Float64)
d = ones((N),Float64)
X = array(initval,Float64)

# PORT setup routine to supply default values
# for arrays iv and v
port.divset(2,iv,v)

# set bound constraints for [ zsensor, tmax ]
lower_bound = [  0.001 ,  0.5 ] # lower bounds
upper_bound = [ 15.0   , 24.0 ] # upper bounds
b = array([lower_bound,upper_bound],Float64)

# modify default values (list indexing starts with 0)
v[32-1] = 1.0e-6 # relative function-convergence tolerance
v[33-1] = 1.0e-6 # X-convergence tolerance

### (E) Start optimization
port.efcoss_dmngb(d,X,b,iv,v,ui,ur)

print "result vector:",X
```

Fig. 9. Setup and execution of the PORT 3 routine MNGB (extended version of Fig. 7)