Springer Proceedings in Mathematics & Statistics

Slawomir Koziel
Leifur Leifsson
Xin-She Yang *Editors*

Solving Computationally Expensive Engineering Problems

Methods and Applications

Springer

Springer Proceedings in Mathematics & Statistics

Volume 97

Slawomir Koziel • Leifur Leifsson
Xin-She Yang
Editors

Solving Computationally Expensive Engineering Problems

Methods and Applications

Springer

Editors

Slawomir Koziel
School of Science and Engineering
Reykjavik University
Reykjavik, Iceland

Leifur Leifsson
School of Science and Engineering
Reykjavik University
Reykjavik, Iceland

Xin-She Yang
School of Science and Technology
Middlesex University
London, United Kingdom

ISSN 2194-1009 ISSN 2194-1017 (electronic)
ISBN 978-3-319-08984-3 ISBN 978-3-319-08985-0 (eBook)
DOI 10.1007/978-3-319-08985-0
Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014950072

Mathematics Subject Classification (2010): 97M10, 65K10, 65D17, 90C26, 90C31, 49Q10, 76B75, 74P05, 74P10

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The costs of extensive computational simulations used for engineering designs can be very expensive, and thus can be a serious bottleneck for the design process in many applications. Nowadays prototyping is heavily involved in design and verification using computer models, and such computational approaches can have many advantages such as the reduction of the overall design costs and design cycles as well as finding good solutions to 'what-if' scenarios. However, the computation costs incurred by extensive computational time can still be very high. Though the speed of the computer power has steadily increased over past the decades, computationally extensive tasks are still a challenging issue. One of the reasons is the ever-increasing demand of the high-accuracy, high-fidelity models for simulating complex systems. For many applications such as those in aerospace engineering, microwave engineering and biological applications, a single simulation task can take hours, even days or weeks on modern computers. While in other applications such as combinatorial optimization problems, the evaluations of every possible combination can be prohibitive because such numbers of combinations can be astronomical. For continuous problems such as computational fluid dynamics and electromagnetic wave simulation, some forms of efficient approximations such as surrogate-based models are needed, while for combinatorial problems, efficient algorithms should be used, though there are no efficient algorithms for genuinely NP-hard problems.

In addition, other challenges associated with such problems include numerical noise in the simulation data, multimodality with multiple local optimum designs due to high nonlinearity, as well as multiple (potentially conflicting) objectives. All these make computationally expensive design tasks even more challenging. Thus, it is timely to edit a book to address such problems with the focus on the latest developments.

From the computational point of view, three key issues should be emphasized: approximation models, optimization algorithms and multi-objectives. Approximation models often use the so-called surrogates that can reliably represent the expensive, simulation-based model of the system/device of interest. If such surrogates are designed properly, they can speed up the simulation significantly. However, such surrogates tend to work for the local, smooth design landscape, and for multimodal problems, good approximations are not easy to construct. This book will include some of the latest developments in this area when dealing with nonlinear problems with complex design objectives.

Even with efficient, computational models, efficient optimization algorithms are also crucial to ensure design optimization that can be carried out successfully in a practically acceptable time scale. Traditional algorithms such as the trust-region method, the interior-point method and gradient-based algorithms can work well for local search, but for multimodal global optimization, heuristic and meta-heuristic algorithms start to demonstrate their efficiency. Swarm intelligence based approaches will be introduced and reviewed in this book.

In almost all engineering applications, there are multiple design objectives and these objectives can often be conflicting, resulting in very complex objective landscapes in the design space. In addition, complex constraints can often modify the search regions significantly and thus make it even more challenging for search algorithms. Furthermore, the computational costs for multi-objective optimization will increase multifold, compared to the counterpart of single objective optimization problems. For example, multi-objective optimization can be very challenging in image processing applications, and we will also briefly touch this area in this book.

This edited book provides a timely snapshot of some of the latest developments in surrogate-based models, optimization algorithms and multi-objective design applications. Topics include surrogate models in engineering design, surrogate-based and PDE-constrained models in climate applications, shape-preserving response predictions, simulation-driven design for antenna designs, space dimension reduction for multi-objective design, large-scale optimization via swarm intelligence, clustering of radar images, classification of laser point clouds, knowledge-based modelling by artificial neural networks and others. However, as the length of the book is limited, it is not our intention to cover everything. As a result, many topics that are very active in the field may not be covered at all. But we hope all the topics we have covered can form a basis with enough literature for further research in the relevant areas.

The editors hope that topics covered in this book will allow the readers to gain understanding of basic mechanisms of surrogate modeling process and surrogate-based optimization algorithms, to follow the trend of swarm intelligence and image processing, and to see the ways of dealing with multi-objective optimization. Ultimately, this may help to reduce the costs of the design process aided by computer simulations. Therefore, this book can serve as a timely reference to researchers, lecturers and engineers in engineering design, modelling and optimization as well as industry.

May 2014
Reykjavik, Iceland
London, UK

Slawomir Koziel
Leifur Leifsson
Xin-She Yang

Contents

Surrogate-Based and One-Shot Optimization Methods for PDE-Constrained Problems with an Application in Climate Models 1
Thomas Slawig, Malte Prieß, and Claudia Kratzenstein

Shape-Preserving Response Prediction for Surrogate Modeling and Engineering Design Optimization 25
Slawomir Koziel and Leifur Leifsson

Nested Space Mapping Technique for Design and Optimization of Complex Microwave Structures with Enhanced Functionality 53
Slawomir Koziel, Adrian Bekasiewicz, and Piotr Kurgan

Automated Low-Fidelity Model Setup for Surrogate-Based Aerodynamic Optimization 87
Leifur Leifsson, Slawomir Koziel, and Piotr Kurgan

Design Space Reduction for Expedited Multi-Objective Design Optimization of Antennas in Highly Dimensional Spaces 113
Adrian Bekasiewicz, Slawomir Koziel, and Wlodzimierz Zieniutycz

Numerically Efficient Approach to Simulation-Driven Design of Planar Microstrip Antenna Arrays By Means of Surrogate-Based Optimization 149
Slawomir Koziel and Stanislav Ogurtsov

Optimal Design of Computationally Expensive EM-Based Systems: A Surrogate-Based Approach 171
Abdel-Karim S.O. Hassan, Hany L. Abdel-Malek, and Ahmed S.A. Mohamed

Atomistic Surrogate-Based Optimization for Simulation-Driven Design of Computationally Expensive Microwave Circuits with Compact Footprints 195
Piotr Kurgan and Adrian Bekasiewicz

Knowledge Based Three-Step Modeling Strategy Exploiting Artificial Neural Network 219
Murat Simsek

Large-Scale Global Optimization via Swarm Intelligence................... 241
Shi Cheng, T.O. Ting, and Xin-She Yang

Evolutionary Clustering for Synthetic Aperture Radar Images 255
Chin Wei Bong and Xin-She Yang

Automated Classification of Airborne Laser Scanning Point Clouds 269
Christoph Waldhauser, Ronald Hochreiter, Johannes Otepka, Norbert Pfeifer, Sajid Ghuffar, Karolina Korzeniowska, and Gerald Wagner

A Novel Approach to the Common Due-Date Problem on Single and Parallel Machines... 293
Abhishek Awasthi, Jörg Lässig, and Oliver Kramer

On Gaussian Process NARX Models and Their Higher-Order Frequency Response Functions.. 315
Keith Worden, Graeme Manson, and Elizabeth J. Cross

Surrogate-Based and One-Shot Optimization Methods for PDE-Constrained Problems with an Application in Climate Models

Thomas Slawig, Malte Prieß, and Claudia Kratzenstein

Abstract We discuss PDE-constrained optimization problems with iterative state solvers. As typical and challenging example, we present an application in climate research, namely a parameter optimization problem for a marine ecosystem model. Therein, a periodic state is obtained via a slowly convergent fixed-point type iteration. We recall the algorithm that results from a direct or black-box optimization of such kind of problems, and discuss ways to obtain derivative information to use in gradient-based methods. Then we describe two optimization approaches, the One-shot and the Surrogate-based Optimization method. Both methods aim to reduce the high computational effort caused by the slow state iteration. The idea of the One-shot approach is to construct a combined iteration for state, adjoint and parameters, thus avoiding expensive forward and reverse computations of a standard adjoint method. In the Surrogate-based Optimization method, the original model is replaced by a surrogate which is here based on a truncated iteration with fewer steps. We compare both approaches, provide implementation details for the presented application, and give some numerical results.

Keywords Optimization • Climate model • Marine ecosystem model • One-shot method • Surrogate-based optimization

1 Introduction

Climate simulations are a very challenging task in applied mathematics and scientific computing. The underlying mathematical systems have a high number of uncertainties with respect to initial values, model parameters, or the relevant processes to be included. Moreover, the state equation solvers often involve iterative algorithms to compute steady or periodic solutions. To identify model parameters and to assess the models, model-to-data misfit functions are minimized using

T. Slawig (✉) • M. Prieß • C. Kratzenstein
Department of Computer Science and KMS Centre for Interdisciplinary Marine Science, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany
e-mail: ts@informatik.uni-kiel.de

S. Koziel et al. (eds.), *Solving Computationally Expensive Engineering Problems*, Springer Proceedings in Mathematics & Statistics 97,
DOI 10.1007/978-3-319-08985-0_1

numerical methods. Since iterative processes are involved, it is crucial to derive highly efficient optimizers.

In this respect, such kind of models and the corresponding optimization or control problems are only one representative for PDE-constrained optimization problems with iterative state solvers. Similar problems arise in Computational Fluid Mechanics and many other application areas. Thus, we will consider here a more general class of PDE-constrained optimization problems.

2 PDE-Constrained Optimization Problems with Iterative State Equation Solvers

We study optimization problems governed by partial differential equations (PDEs), given in the following general form

$$\min_{(y,u)\in Y\times U_{ad}} \quad J(y,u) \tag{1}$$

$$\text{s.t.} \qquad e(y,u) = 0 \text{ in } Z. \tag{2}$$

Here $J : Y \times U \to \mathbb{R}$ is the cost or objective function defined on the cartesian product of state and control (or parameter) spaces Y and U, respectively. The admissible set U_{ad} characterizes additional constraints on the controls (parameters) u.

- either as infinite-dimensional function spaces
- or, e.g., when studying the discretized problem, as finite-dimensional spaces, being isomorphic to $\mathbb{R}^{n_Y}, \mathbb{R}^{n_U}$ with n_Y, n_U the dimensions of Y, U, respectively.

It is also possible (and often the case) that Y is a function space and U is finite-dimensional, e.g., a space of real-valued parameters. In many cases, the admissible set is then given by simple bounds, i.e., as

$$U_{ad} := \{u \in U : u_{min} \leq u \leq u_{max}\}$$

with some fixed $u_{min}, u_{max} \in \mathbb{R}^{n_U}$, and the inequalities meant component-wise.

The state equation is defined by the often nonlinear mapping $e : Y \times U \to Z$. If it is given in an infinite-dimensional setting, e.g., as weak form of a PDE, the space Z is the dual of the test space. When considering an already discretized state equation, Z is isomorphic to $\mathbb{R}^{n_Z}$ with appropriate dimension n_Z.

We will also use the reduced cost functional $\hat{J} : U \to \mathbb{R}$ defined by

$$\hat{J}(u) := J(y(u), u) \quad \text{where } y = y(u) \Longleftrightarrow e(y,u) = 0. \tag{3}$$

It is characteristic for the two optimization methods we describe that the solution of the state equation (2) is computed by a fixed-point type iteration of the form

$$y_j = G(y_{j-1}, u), \quad j = 1, 2, \dots, \tag{4}$$

with iteration function $G : Y \times U \to Y$. In climate models, this iteration is called *spin-up*. Each iteration step includes one or more time-integration steps with constant or periodic external forcing, thus leading to a steady solution, in climate models often a steady annual cycle. A similar procedure is also very common in fluid dynamics to compute stationary or periodic solutions with transient solvers. This fact also motivates the notion *pseudo time-stepping scheme* for (4).

We will briefly write $G^j(y, u)$ for the result of j subsequent iterations with the same control variable u, i.e.

$$y_j = G^j(y_0, u), \quad j = 1, 2, \dots \tag{5}$$

The solution of the state equation (2) is now given as the limit

$$y^* = \lim_{j \to \infty} y_j, \tag{6}$$

We will assume here that this limit exists for all feasible controls $u \in U_{ad}$, guaranteed, for example, by some contraction or quasi-contraction property of G. Assuming that G is continuous w.r.t. its first argument, (6) implies

$$e(y, u) = 0 \text{ in } Z \iff y = G(y, u) \text{ in } Y. \tag{7}$$

In practice, the iteration (4) has to be terminated when a stopping criterion is satisfied. Thus, instead of using the limit y^* from (6) in the evaluation of J, an approximation $\hat{y} := y_{j_{max}}$ with an appropriate value of j_{max} is taken.

3 Exemplary Application: Parameter Optimization in a Marine Ecosystem Model

In this section we show a model problem from climate research that fits in the above general setting, and where both methods can be and have been applied. The model problem is an application in marine science. It deals with the identification of climate model parameters using experiment or model data. In climate models, the iterative computation of the state variables is rather common and usually very time-consuming. Three-dimensional climate model simulations may take several days or more of computer time. In this section, we introduce the underlying model, i.e. the state equation, its discretization, and the resulting iterative scheme (4).

3.1 Marine Ecosystem Models as Example for Climate Models

We here give the basic structure of marine ecosystem models which serve as one possible example for climate models. A typical optimization problem for this kind of models is parameter identification or optimization, i.e., poorly known or not measurable model parameters shall be adjusted such that the model output fits given observational (or other model) data. This task is also called model calibration. We give one example that we actually used in both optimization strategies.

Marine ecosystem models consist of two parts, namely the ocean circulation and the biogeochemistry. The former is basically given by the Navier-Stokes equations including temperature and salinity transport, whereas the latter describes the reaction of and interactions between nutrients and different species of ocean biota, e.g., photosynthesis, dying and growth of plankton species, etc. The marine ecosystem plays an import role within the global carbon cycle, but its complex organic and inorganic cycles are challenging when formulating a comprehensive biogeochemical model, see for example, [23].

The coupling between ocean circulation and the biogeochemical interactions is mostly regarded as one-way coupling. The influence of the circulation (including temperature and salinity distribution) on the biota is assumed to be much more important as vice versa. This is mainly motivated by the high complexity and the enormous computational effort that is necessary to solve the time-dependent and spatially three-dimensional coupled system of (1) an ocean circulation model (consisting of the Navier-Stokes equations with free ocean surface, energy and salinity transport equations) together with (2) the biogeochemical model (consisting of between two and about 50 transport equations for the different species, depending on the chosen model). Thus often an *off-line computation* is performed: Velocity, turbulent diffusion, temperature, and salinity fields are computed beforehand by the ocean circulation model and used as input or *forcing* data for the biogeochemical simulations. This significantly reduces the amount of computation. By using pre-computed circulation data, all tracers necessarily are regarded as *passive*, i.e., they do not have any influence on the circulation.

In our example, we use this off-line mode. The model equations then form a system of coupled transport or advection-diffusion-reaction equations, with reaction terms given by the biogeochemical processes. Our model equations then read

$$\frac{\partial y_i}{\partial t} = \operatorname{div}(\kappa \nabla y_i) - \operatorname{div}(v y_i) + q_i(y, u), \quad \text{in } \Omega \times [0, T], \quad i = 1, \ldots, n_{state} \tag{8}$$

together with given initial data $y_{init} = y(t = 0) \in Y$ and usually Neumann boundary conditions. Here, $\Omega \subset \mathbb{R}^3$ is the spatial domain, $[0, T]$ the considered time interval, $y = (y_i)_{i=1,\ldots,n_{state}}$ the vector of state variables (biogeochemical tracers), where $y_i(x, t)$ denotes a single tracer concentration at (x, t).

The time dependent turbulent mixing or diffusion coefficient κ and the velocity vector field v, together with temperature and salinity distributions (entering in the q_i, but omitted for brevity in the notation above) are precomputed data from an ocean model and thus *not* subject to identification here. The turbulent mixing dominates the molecular tracer diffusion in this application, and thus κ is the same for all tracers. Since velocity v and κ are given, the nonlinearity in the above system w.r.t. to the state variables $y = (y_i)_i$ comes from the nonlinear coupling terms q_i, whereas the transport part (diffusion and advection, the first two terms on the right) is linear.

The parameters to be identified are summarized in the vector u and appear in the nonlinear biogeochemical coupling terms q_i. These are non-autonomous extensions of predator–prey models, since, for example, the growth rate of phytoplankton (algae) depends on the sunlight and thus on space and time. Nearly all of these coupling terms q_i are spatially local, i.e., they describe processes happening at point x and not depending on neighborhood points. Some of them include sinking processes and thus become non-local. The sinking velocity of dead material, for example, is one parameter that is crucial to identify.

3.2 *Example: The N-DOP Model*

In this section we describe the biogeochemical model we used both in the One-shot and the Surrogate-based Optimization approach. Since there are many different biogeochemical models, the one used here is only an example. Modelers are interested not only in the right parameters for a single model, but moreover try to assess and compare different models, a task where parameter identification becomes important.

The N-DOP model (see [19]) consists only of the two tracers phosphate (nutrients, N) and dissolved organic phosphorus (DOP), denoted by $y = (y_1, y_2)$. Thus $n_{state} = 2$ in (8), which is at the lower limit of complexity for such kind of model. Typical for a biogeochemical model are different biogeochemical interactions in two horizontal layers in the ocean, namely the upper, *euphotic zone* Ω_1 (where light enables photosynthesis) and the lower, non-euphotic zone Ω_2. The model consists of the following coupling terms:

$$q_1(y,u) = \begin{cases} -g(y_1, I) + \lambda y_2 & \text{in } \Omega_1 \\ (1-\sigma)\dfrac{\partial \tilde{g}}{\partial x_3}(y_1, I) + \lambda y_2 & \text{in } \Omega_2 \end{cases}$$

$$q_2(y,u) = \begin{cases} \sigma g(y_1, I) - \lambda y_2 & \text{in } \Omega_1 \\ -\lambda y_2 & \text{in } \Omega_2. \end{cases}$$

The vertical spatial coordinate here is x_3. On the left-hand side, we have summarized here as above all parameters (see below) in the vector u. The *biological production*

$$g(y_1, I) = \alpha \frac{y_1}{y_1 + K_N} \frac{I}{I + K_I}$$

depends on nutrients y_1 and light I, and is limited by a maximum production rate parameter α. Light is computed from the short wave radiation (as a function of latitude and time, thus making g a non-autonomous function), the photosynthetically available radiation, the ice cover and the exponential attenuation of water K_{H2O}. A fraction σ of the biological production remains suspended in the water column as dissolved organic phosphorus, which remineralizes with rate λ. The remainder of the production sinks as particulate to the bottom where it is remineralized according to an empirical power law relationship:

$$\tilde{g}(y_1, I) = \left(\frac{x_3}{x_{depth}}\right)^{-b} \int_0^{x_{depth}} g(y_1(x,t), I(x,t))\, dx_3.$$

Here $x_{depth} = x_{depth}(x_1, x_2)$ is the depth of the upper layer Ω_1 which depends on the horizontal coordinate (x_1, x_2). The parameters to be optimized are given in Table 1.

Here, the parameters are assumed to be constant w.r.t. to space and time, i.e., the parameter space U equals $\mathbb{R}^{n_U}$, with U_{ad} defined by box constraints that are also given in Table 1. For further model details we refer to [19].

3.3 Discretization

In this section we describe a discretization scheme that is adapted to the mentioned one-way coupling and was used in our numerical tests. It is built upon a matrix representation of the linear part of system (8), namely the pure transport operators.

The *Transport Matrix Method (TMM)* introduced in [13] computes the effect of the ocean circulation on the tracer distributions. It avoids using ocean circulation data $\kappa, \mathbf{v}$ directly and discretizing the corresponding diffusion and advection operators in the tracer transport simulation. In contrast, the TMM builds up a set of pairs of explicit and implicit matrices (corresponding to the discretization in the ocean model which is based on an operator splitting) in every time step. The transport matrices are generated by several runs of one time step of the ocean model, each for a given initial tracer distribution (designed similar to a linear finite element ansatz function) in every grid point. Each resulting tracer distribution builds one column of the pair of transport matrices for one ocean model time step, and all evaluations together build up the whole matrix pair. Since the discretization of the transport in ocean models typically involves nonlinear schemes (like flux limiters etc.), this generation of the matrix pair can be seen as a way of linearization of the scheme. Since the external forcing depends on time, even with climatological (i.e., annually periodic) forcing data and due to the typical time step-size of 3 h, the amount of storage for all these matrices would be prohibitively large, even though the matrices are block-diagonal and sparse. Thus, the matrices are usually averaged in time. In our case, they are monthly averaged. More details on the temporal and spatial discretization and the evaluation of transport matrices, especially in combination with operator splitting schemes, can be found in [13].

Table 1 Parameters to be optimized in the N-DOP model with bounds $u_{i,min}$, $u_{i,max}$, $u_{i,guess}$ (used in the regularization) and initial value u_{i0} for the optimization methods

u_i	Symbol	Description	Unit	$u_{i,min}$	$u_{i,max}$	$u_{i,guess}$	u_{i0}
u_1	λ	Remineralization rate of DOP	d^{-1}	0.25	0.75	0.5	0.3
u_2	α	Maximum community production rate	d^{-1}	1.5	200	2.0	5.0
u_3	σ	Fraction of DOP	–	0.05	0.95	0.67	0.40
u_4	K_N	Half saturation constant of N	mmolPm^{-3}	0.25	1.5	0.5	0.8
u_5	K_I	Half saturation constant of light	Wm^{-2}	10.0	50.0	30.0	25.0
u_6	K_{H2O}	Attenuation of water	m^{-1}	0.01	0.05	0.02	0.04
u_7	b	Sinking velocity exponent	–	0.7	1.5	0.858	0.78

We now turn to the resulting discretized version of (8). Let Ω_h be the set of discrete spatial points $x \in \overline{\Omega}$, usually arranged on a rectangular grid, adapted to the bottom ocean topography and coastlines. This set Ω_h is determined by the spatial discretization of the used ocean model which was used to compute the transport matrices. In our case, the latitudinal and longitudinal resolution of the underlying ocean model grid is 2.8125°, with 15 vertical levels. This results in a dimension of the discretized state space Y of $n_Y = 105,498$.

Let $\mathbf{y}_l$ be the appropriately arranged vector of values $(y_i(x, t_l))_i$ of all n_{state} tracers on all spatial grid points $x \in \Omega_h$, and, in a similar way, $\mathbf{q}_l(\mathbf{y}_l, u)$ the vector of discretized coupling terms q_i for all $x \in \Omega_h$, both at fixed time step l. The time integration scheme for 1 year model time with a fixed step-size τ then reads

$$\mathbf{y}_{l+1} = \mathbf{A}_{imp,l}\,(\mathbf{A}_{exp,l}\,\mathbf{y}_l + \tau\,\mathbf{q}_l(\mathbf{y}_l, u)) =: \varphi_{l+1}(\mathbf{y}_l, u), \quad l = 0, 1, \dots, l_{year} - 1. \quad (9)$$

Here $\mathbf{A}_{imp,l}, \mathbf{A}_{exp,l}$ are the implicit and explicit transport matrices at time step l, which are linearly interpolated between the pre-computed set of monthly matrices to the corresponding time t_l. In our case, the number of time steps in 1 year model ranges may vary from $l_{year} = 45$ (a very coarse temporal resolution, resulting in a step-size of $\tau = 192\,\text{h}$) to $l_{year} = 2,880$ (which is the original one of the model, resulting in $\tau = 3\,\text{h}$). Each step in the time-integration scheme (9) now consists of the evaluation of the coupling term $\mathbf{q}_l$ and two matrix–vector multiplications.

In climate model calibration or parameter optimization, as a first step a steady annual cycle, in our case a periodic solution of (8), is computed and used in the cost function evaluation. As a consequence, the iteration function G in (4) is given as

$$G := \varphi_{n_{year}} \circ \dots \circ \varphi_1.$$

We thus regard one step in (4) as 1 year model time, and j there counts model years. The set of all discrete time instants used in a simulation is denoted by

$$[0, T]_\tau := \left\{((j-1)l_{year} + l)\tau : j = 1, \dots, j_{max}, l = 1, \dots, l_{year}\right\},$$

where j_{max} is the number of model years simulated. For the numerical computation, the iteration starts with a constant distribution $\mathbf{y}_0$. It can be observed that after $j_{max} \approx 3,000$ to $10,000$ iterations of G in (4), i.e. years model time, an acceptable approximately steady periodic solution is obtained, see also [16]. Since a typical step-size is 3 h, this means that about 10^6–10^8 discrete time steps of the spatially three-dimensional system of transport equations are necessary to attain a steady periodic solution. Even on parallel high performance hardware, the computation of a numerically converged steady periodic solution may take several minutes. This is the reason for the high demand for fast optimization methods, since usually one optimization may take hundreds of function evaluations, i.e., the mentioned computations of steady periodic solutions. More details and results can be found in [21].

3.4 *Parameter Optimization Problem*

In our numerical tests, the considered minimization problem was a least squares cost functional with regularization term given by

$$J(y,u) := \frac{1}{2}\|y - y_{data}\|_Y^2 + \frac{\alpha}{2}\|u - u_{guess}\|_U^2, \quad \alpha > 0. \tag{10}$$

Working in the finite-dimensional spaces for the discretized models, the used norms are Euclidean vector norms, optionally with weighting coefficients. Components of u are the parameters in Table 1. We follow [16] in the choice of the initial parameter guess u_{guess} and took the initial value u_0 (both given in Table 1) for the parameters in both optimization methods. For the choice of the desired state or target data y_{data} we use here model-generated test data, obtained with parameter vector $u = u_d$, in order to evaluate the two methods and the quality of their results compared to the known optimal parameter values.

4 Direct Optimization

In this section we describe a direct optimization algorithm, i.e., a method where the state equation iteration (4) is numerically converged before the parameters u are updated. Such kind of algorithm is used for example, when a black-box optimizer is applied on the original problem (1–2). In an iterative optimization algorithm, there will be two nested iterations then, and it can be conceptually written as follows.

Optimization Algorithm with Iterative State Equation Solver:

1. Choose an initial value for the control u_0.
2. For $k = 0, 1, \dots, k_{max}$:
   a. Choose an initial value y_0 for the state corresponding to control u_k.
   b. Compute an approximation of the state for u_k:
   $$\hat{y}_k = G^{j_k}(y_0, u_k).$$
   c. Optionally: Compute the gradient of the cost J w.r.t. u, i.e.,
   $$\frac{d}{du} J(y(u), u)\bigg|_{(\hat{y}_k, u_k)}, \tag{11}$$
   using
   - either the gradient $\hat{y}_k' := \dfrac{d\hat{y}_k}{du}\bigg|_{u=u_k}$
   - or the adjoint state $\bar{y}_k$.

d. Perform an update $u_k \to u_{k+1}$ of the control.
e. If some criterion for u or J or its gradient is satisfied, stop.

The number j_k of inner iterations in step 2b might be varying with k or be fixed beforehand to some value constant j_{max}. To make things simpler in notation, we will here assume in this section that $j_k = j_{max}$ is constant.

A typical case which motivates the two methods compared here is that the evaluation of G is costly and/or that the convergence in (4) and thus in step 2b (and then presumably also in the gradient iterations usually needed in step 2c) is slow. Both strategies aim to reduce this high computational effort by using some kind of reduced accuracy or low-fidelity approximation in step 2b by taking low values j_{max} there to reduce the computationally effort or to make the whole algorithm feasible at all. As a result, the $\hat{y}_k$ used to evaluate the cost are different (and eventually far away) from the limits

$$y_k^* := \lim_{j\to\infty} G(y_j, u_k). \tag{12}$$

We call the above algorithm a *direct* or *direct fine model optimization* if the number j_{max} of inner state iterations equals a high number denoted by j^f, for example given by the original model or simulation code before used in an optimization.

4.1 Gradient Evaluation

The optional gradient computation in step 2c of the above algorithm can be performed either by a sensitivity or by an adjoint approach. Here, we assume that all derivatives of G, J and y used below exist as Fréchet derivatives and that Y and U are Hilbert spaces.

4.1.1 Sensitivity Equation Approach

Using a sensitivity equation, the iteration (4), with $u = u_k$ and up to step $j = j_k$, is differentiated w.r.t. u. This leads to the following iteration for the derivatives:

$$y'_j := \left.\frac{dy_j}{du}\right|_{u=u_k} = G_y(y_{j-1}, u_k)y'_{j-1} + G_u(y_{j-1}, u_k), \quad j = 1, \dots, j_k. \tag{13}$$

Here subscripts y, u denote partial derivatives of G. This iteration is initialized by $y'_0 = \frac{dy_0}{du}$ which usually is zero, except in the case when the initial data are to be optimized as well (which is possible).

Now, the two iterations in steps 2b and 2c can be computed in different ways and orders. We use here the notions introduced in [7, 11].

- In the *two-phase approach*, the two iterations are performed after another, i.e., the two steps 2b and c remain separate. It can be seen from (13), where the iterates y_j of the state are used, that these have to be stored during step 2b in order to use them in step 2c.
- In the *piggy-back approach*, both iterations are combined to

$$\left.\begin{aligned} y'_j &= G_y(y_{j-1}, u_k)y'_{j-1} + G_u(y_{j-1}, u_k),\\ y_j &= G(y_{j-1}, u_k) \end{aligned}\right\} \quad j = 1, \dots, j_k.$$

  This approach avoids the storing of the state iterates.
- The *Christianson approach* presented in [2] performs the sensitivity iteration (13) with the previously computed (numerically) converged state $\hat{y}_k$ instead of using its iterates y_j, $j = 0, \dots, j_k - 1$, thus also avoiding storage of the iterates.

Once $\hat{y}'_k$ is computed, the gradient of $\hat{J}$ with respect to the control can be computed by the chain rule as

$$\left.\frac{d}{du} J(y(u), u)\right|_{(\hat{y}_k, u_k)} = J_y(\hat{y}_k, u_k)\hat{y}'_k + J_u(\hat{y}_k, u_k). \tag{14}$$

4.1.2 Adjoint Approach

In the adjoint approach, the Lagrangian associated with problem (1–2) is used to compute the gradient (14) without knowing or evaluating $\hat{y}'_k$. We compute the directional derivative of the state equation in its fixed-point form, namely the right-hand side of (7), w.r.t. u in direction v and obtain

$$y'(u)v = \frac{d}{du} G(y(u), u)v = G_y(y(u), u)y'(u)v + G_u(y(u), u)v \quad \text{inŁ } Y, \tag{15}$$

i.e.,

$$\left[G_y(y(u), u) - Id_Y\right] y'(u)v = -G_u(y(u), u)v \quad \text{inŁ } Y. \tag{16}$$

Here Id_Y is the identity in Y, and the subscripts y, u denote partial derivatives of G.

To eliminate $y'(u)$ (or its approximation $\hat{y}'_k$) from the last equation, we introduce the Lagrange multiplier or adjoint state $\bar{y} \in Y'$ (the dual of Y) and the Lagrangian $L : Y \times Y' \times U \to \mathbb{R}$ given by

$$L(y, \bar{y}, u) = J(y, u) + \langle \bar{y}, G(y, u) - y \rangle_{Y', Y}.$$

Here $\langle \cdot, \cdot \rangle_{Y',Y}$ denotes the dual pairing in the function space setting. It can be replaced by an inner product in $\mathbb{R}^{n_Y}$ in finite dimensions. In a solution point $(y^*, \bar{y}^*, u^*) \in Y \times Y' \times U$ of problem (1–2), the Lagrangian is stationary w.r.t. to variations in all three variables. This leads to the three Karush-Kuhn-Tucker (KKT) conditions:

$$\left.\begin{array}{lll} 0 = L_y(y^*, \bar{y}_*, u_*) = J_y(y^*, u_*) + \bar{y}_* \circ G_y(y^*, u_*) - \bar{y}_* & \text{in } Y', \\ 0 = L_{\bar{y}}(y^*, \bar{y}_*, u_*) = G(y^*, u_*) - y^* & \text{in } Y'' \cong Y, \\ 0 = L_u(y^*, \bar{y}_*, u_*) = J_u(y^*, u_*) + \bar{y}_* \circ G_u(y^*, u_*) & \text{in } U'. \end{array}\right\} \quad (17)$$

For arbitrary state y and control u, the first equation (called the adjoint equation) can be used to compute the adjoint variable or state $\bar{y}$ from

$$\langle \bar{y}, \left[G_y(y, u) - Id_Y\right] w \rangle_{Y',Y} = -J_y(y, u)w \quad \text{for all } w \in Y. \quad (18)$$

With the adjoint state $\bar{y}$ computed, we take $w = y'(u)v$ in this equation and get with (16) the representation

$$J_y(y, u)y'(u)v = -\langle \bar{y}, \left[G_y(y, u) - Id_Y\right] y'(u)v \rangle_{Y',Y} = \langle \bar{y}, G_u(y, u)v \rangle_{Y',Y}. \quad (19)$$

Thus the gradient representation (14) can be written as

$$\frac{d}{du} J(y(u), u)\bigg|_{(\hat{y}_k, u_k)} = \bar{y}_k \circ G_u(\hat{y}_k, u_k) + J_u(\hat{y}_k, u_k). \quad (20)$$

5 One-Shot Optimization Method

The approach described here was in this form developed by Hamdi and Griewank, and can be seen as an extension of the piggy-back strategy. Theoretical results were published in [8, 9], and summarized also in [5]. An engineering application was presented in [18] and results from an ocean model calibration in [15]. Two examples in infinite-dimensional spaces are studied in [12]. In the One-shot approach described here, the motivation is to update the control u already during the state iteration. In the above algorithm, this means that the number j_{max} of steps in the state equation iteration (step 2b) is set to 1 or (in the so-called multistep One-shot method to some low value).

The motivation for this method is again taken from the KKT system (17). The adjoint equation, i.e., the first equation in (17), can be formulated (omitting the stars) as a fixed-point equation for the adjoint state:

$$\bar{y} = \bar{G}(y, \bar{y}, u) := J_y(y, u) + \bar{y} \circ G_y(y, u) \quad \text{in } Y, \quad (21)$$

and a corresponding fixed-point iteration (only for the adjoint, with state and control fixed) can be defined by

$$\bar{y}_j = \bar{G}(y, \bar{y}_{j-1}, u), \quad j = 1, \ldots, j_k. \tag{22}$$

Similar to the iteration for the state in (5), we write for j subsequent iterations:

$$\bar{y}_j = \bar{G}^j(y, \bar{y}_0, u), \quad j = 1, 2, \ldots \tag{23}$$

We now formulate the following algorithm:

Multistep One-Shot Optimization Algorithm:

1. Choose initial values for state, adjoint state, and control $(y_0, \bar{y}_0, u_0)$.
2. For $k = 0, 1, \ldots, k_{max}$:
   a. Compute an approximation of the state for u_k:
   $$y_{k+1} = G^{j_k}(y_k, u_k).$$
   b. Update the adjoint state:
   $$\bar{y}_{k+1} = \bar{G}^{\bar{j}_k}(y_{k+1}, \bar{y}_k, u_k).$$
   c. Update the control using the formula
   $$u_{k+1} = u_k - B_k^{-1}\left[J_u(y_{k+1}, u_k) + \bar{y}_{k+1} \circ G_u(y_{k+1}, u_k)\right]$$
   d. If some criterion for $(y, \bar{y}, u)$ or J is satisfied, stop.

The term *multistep* comes from the usage of the $j_k, \bar{j}_k$ subsequent state and adjoint updates in steps 2a and b, respectively, before a control update is performed in step 2c. The operators $B_k : U \to U'$ can be seen as control preconditioners. They are chosen such that the whole coupled iteration defined in step 2 converges. In the finite-dimensional setting, the B_k are matrices.

If contractivity of the state iteration is given, i.e., there exists $\rho < 1$ satisfying

$$\|G(y, u) - G(\tilde{y}, u)\| \leq \rho \|y - \tilde{y}\|, \qquad \forall y, \tilde{y} \in Y, \tag{24}$$

the first equation in the coupled iteration (step 2a) converges linearly for fixed u. Although the second equation exhibits a certain time-lag, it converges with the same asymptotic R-factor (see [10]). For the coupled iteration, the goal is to find B_k that ensure that the spectral radius of the coupled iteration stays below 1 and as close as possible to the one of the Jacobian of the original iteration function G.

5.1 Choice of Preconditioner B_k

We now briefly describe the choice of appropriate preconditioners B_k according to [8,9] that we used in our study. For the derivation of B_k, the authors of [8,9] use the doubly augmented Lagrangian L^a, defined as

$$L^a(y,\bar{y},u) := L(y,\bar{y},u) + \frac{\alpha_L}{2}\|G(y,u)-y\|_Y^2 + \frac{\beta_L}{2}\|\bar{G}(y,\bar{y},u)-\bar{y}\|_{Y'}^2,$$

which is the Lagrangian of the original problem augmented by the errors in the state and adjoint fixed-point equations, with $\alpha_L, \beta_L > 0$ being weighting coefficients. In [9] it is proved that under certain conditions on α_L and β_L (see below), stationary points of problem (1–2) are also stationary points of L^a and that L^a is an exact penalty function. This leads to the idea to choose B_k as an approximation to the Hessian of L^a, i.e. $B_k \approx \frac{d^2}{du^2}L^a(y_k,\bar{y}_k,u_k)$. In [9], it is also proved that—in the finite-dimensional setting—descent of L^a is provided for any preconditioner B_k fulfilling

$$B_k \succeq B_0 := \frac{1}{\sigma}(\alpha_L G_u^\top G_u + \beta_L L_{yu}^\top L_{yu}) \tag{25}$$

i.e., $B_k - B_0$ is positive semidefinite, with

$$\sigma := 1-\rho-\frac{(1+\frac{\|L_{yy}\|}{2}\beta_L)^2}{\alpha_L\beta_L(1-\rho)}.$$

In order to make the maximal eigenvalue of B_0 as small as possible (but still positive), under the assumptions $\sqrt{\alpha_L\beta_L}(1-\rho) > 1+\frac{\beta_L}{2}\|L_{yy}\|$ and $\|L_{yy}\| \neq 0$ the choice

$$\alpha_L = \frac{\|L_{yu}\|^2\beta_L(1+\frac{\|L_{yy}\|}{2}\beta_L)}{\|G_u\|^2(1-\frac{\|L_{yy}\|}{2}\beta_L)}, \qquad \beta_L = \frac{3}{\sqrt{\|L_{yy}\|^2+3\frac{\|L_{yu}\|^2}{\|G_u\|^2}(1-\rho)^2}+\frac{\|L_{yy}\|}{2}}$$

was made in [9]. At a stationary point of L^a the Hessian of L^a w.r.t. u is

$$\frac{d^2}{du^2}L^a = \alpha_L G_u^\top G_u + \beta_L L_{yu}^\top L_{yu} + L_{uu}.$$

As L^a is an exact penalty function, $\frac{d^2}{du^2}L^a \succ 0$ in a neighborhood of the constrained optimization solution. Assuming that also $\frac{d^2}{du^2}L \succ 0$ implies that the preconditioner

$$B = \frac{1}{\sigma}(\alpha_L G_u^\top G_u + \beta_L L_{yu}^\top L_{yu} + L_{uu})$$

fulfills (25) and thus the update in u of the coupled iteration yields descent on L^a. In the more recent paper [8], the same authors perform a different approach in the choice of the weighting factors and obtain two alternative versions, namely:

$$\sigma = 1, \quad \alpha_L = \frac{2\|L_{yy}\|}{(1-\rho)^2}, \beta_L = \frac{2}{\|L_{yy}\|} \text{ or } \alpha_L = \frac{6\|L_{yy}\|}{(1-\rho)^2}, \beta_L = \frac{6}{\|L_{yy}\|}. \quad (26)$$

To simplify the computations even more, in [8] the choice $\|L_{yy}\| = 1$ is proposed.

5.2 Required Derivatives and Automatic Differentiation

In the One-shot iteration including the preconditioner B_k, first and second order derivative information is needed. The cost for its calculation is small compared to the one of the direct method, since here only one iteration step, i.e., G and not G^{j_k}, has to be differentiated. In the discretized setting, the iteration function $G : \mathbb{R}^{n_Y \times n_U} \to \mathbb{R}^{n_Y}$ consists of up to 2,880 intermediate time steps, compare Sect. 3.3. Thus, the computation of the needed derivatives G_y, G_u using, for example, forward finite differences would mean to perform those time steps n_Y, n_U times only for G_y, G_u, with high computational costs. To reduce the effort and moreover avoid the approximation error of finite differences, we used here the technology of *Automatic or Algorithmic Differentiation (AD)*, see [11]. We used the tool *Transformation of Algorithm in Fortran* (*TAF*, [6]) on the nonlinear biogeochemical model terms q_i, whereas the linear transport matrix part was differentiated analytically.

There are two modes of AD, namely the *forward* and the *reverse mode* (corresponding to an adjoint equation). The forward mode enhances an iteration with the corresponding derivative iteration and thus is the discrete analogue of the sensitivity equation approach from Sect. 4.1.1. Here, the cost for evaluating derivatives increases linearly with the number of unknowns, in our case n_Y or n_U, which is comparable to a finite difference approximation, but avoids the approximation errors. In our application, it is only recommended for derivatives w.r.t u, since in our application $n_U = 7 \ll n_Y \approx 100,000$. In contrast, the reverse mode stores all intermediate variables of the function evaluation and then, in a reverse sweep reverting the order of operations, computes *all partial derivatives* of the function with respect to intermediate variables *at once*. It is the discrete analogue of the adjoint equation approach described in Sect. 4.1.2. In particular, the gradient of a scalar valued function as J can be evaluated as a cost independent from the number of independent variables. Therefore, the reverse mode is appropriate for derivatives w.r.t. y. The concatenation of a reverse and a forward sweep yields second order derivatives. Due to the very small number of parameters $n_U = 7$ and the complexity of the code, we chose the reverse sweep followed by a finite differences approach to compute second order derivatives. Table 2 summarizes the applied strategies for the computation of the needed derivatives.

Table 2 Computation of derivatives using different approaches

Derivative	Mode of computation
J_y	Analytically
J_u	Analytically
$\bar{y}^\top G_y, \bar{y}^\top G_u$	One reverse sweep of AD combined with transport matrices by hand
G_u	Forward mode of AD combined with transport matrices by hand
J_{yu}	Analytically (= 0)
$\bar{y}^\top G_{yu}$	After computation of $\bar{y}^\top G_y$, application of finite differences

For the computation of the weights σ, α_L and β_L of the preconditioner B_k, see Sect. 5.1, we chose the first of the cheaply computable versions defined in (26) and fixed $\|L_{yy}\| = 1$. Furthermore, we set the unknown contraction factor ρ of the state iteration function G to $\rho = 0.9$. We observed for the N-DOP model that the contraction property (24) is violated for some steps in the state iteration. However, it converges to a steady solution and the average contraction factor is close to, but less than 1. Fixing ρ in such a way simplifies the code. Another option is to update y and $\bar{y}$ without an update of u (i.e., increasing the iteration numbers $j_k, \bar{j}_k$) until the contraction factor ρ is less than 1 again.

6 Surrogate-Based Optimization

The surrogate-based optimization strategy (SBO, see, e.g., [1,4,17,22]) is built upon a *coarse or low-fidelity model* that can be evaluated much faster than the original, in this context then called *fine or high-fidelity model* used in the direct optimization approach. Since a coarse model naturally does not include as much information or have the accuracy of the fine one, an (ideally computationally cheap) *alignment* or *correction* of the coarse model is performed. The aim of this alignment is to keep the output of the aligned coarse model, the so-called *surrogate*, close to the output of the fine model, also when the optimization parameters are changed to a certain limit. When this optimization of the surrogate is numerically converged, the fine model is evaluated again and the alignment is updated. Then the surrogate is optimized again, and the process is iterated. The benefit is that fine model optimization runs are completely avoided, which reduces the overall effort tremendously.

Surrogates can be created by approximating sampled fine model data (*functional* surrogates). Popular techniques include polynomial regression, kriging, artificial neural networks, and support vector regression [22, 24, 25]. Another possibility, exploited in this work, is to construct the surrogate model through appropriate correction/alignment of a low-fidelity or coarse model (*physics-based* surrogates, [26]). Physics-based surrogates inherit physical characteristics of the original fine model so that only a few fine model data is necessary to ensure their good alignment with the fine model. Moreover, generalization capability of the physics-based models is typically much better than for functional ones. As a result, SBO schemes working with this type of surrogates normally require small number of fine model evaluations to yield a satisfactory solution. On the other hand, their transfer to other applications is less straightforward since the underlying coarse model and chosen correction approach is rather problem specific. The specific correction technique exploited in this work is described below (see also [20]).

In applications that use iterative state equation solvers, a simple way to construct a coarse model is just to stop the iteration after fewer steps or with a relaxed stopping criterion. Then, the term *coarse* refers not to a coarser discretization in space and/or time, but to a model and state with reduced accuracy compared to the original one. This way of constructing a coarse model is much simpler than to use a coarser discretization scheme, which of course is also possible, but involves prolongation and restriction operations on the model output. We now give the structure of an SBO algorithm based on this coarse model construction.

Surrogate-Based Optimization Algorithm (With Iterative Solver):

1. Choose initial value for the control u_0.
2. For $k = 0, 1, \dots, k_{max}$:
    a. Compute an approximation of the state for u_k with a fine model:
    $$y_k^f = G^{j_k^f}(y_k, u_k).$$
    b. Compute an approximation of the state for u_k with a coarse model:
    $$y_k^c = G^{j_k^c}(y_k, u_k).$$
    c. Compute the correction or alignment operator
    $$A_k = A_k(y_k^f, y_k^c) : Y \to Y \qquad \text{satisfying} \qquad A_k y_k^c = y_k^f$$
    $$\text{and optionally} \qquad \frac{dA_k y_k^c}{du} = \frac{dy_k^f}{du}$$
    and define the surrogate
    $$s_k : U \to Y, \qquad s_k(u) := A_k G^{j_{k,c}}(y_k, u)$$
    d. Compute
    $$u_{k+1} = \operatorname*{argmin}_{u \in U_{ad}} J(s_k(u), u),$$
    or approximate it by i_k steps of an iterative optimization method.
    e. If some criterion for y, u or J is satisfied, stop.

The iteration numbers chosen in steps 2b (j_k^c) and step 2d (i_k) can be either kept constant or adapted. The latter variant is called a *hybrid SBO strategy*. The two conditions imposed on the alignment operator A_k in step 2c are called *zeroth* and *first order consistency*. The second one might be relaxed to be valid only approximately. If the surrogate s_k satisfies both conditions at $u = u_k$, the SBO algorithm is provable convergent to at least a local optimum under conditions regarding the coarse and fine model smoothness, and provided that the algorithm is enhanced by the trust-region safeguard, i.e., in step 2d the minimum is just taken over the set

$$U_k := \{u \in U_{ad} : \|u - u_k\|_U \leq \delta_k\}$$

with δ_k being a trust-region radius updated according to the usual trust region rules. We refer the reader to, e.g., [3, 14] for more details.

One example for the alignment operator we used in our application is the point-wise multiplicative operator

$$A_k(y_k^f, y_k^c)y(x,t) := \frac{y_k^f(x,t)}{y_k^c(x,t)}\, y(x,t) \qquad \text{for all grid} - \text{points } (x,t) \in \Omega_h \times [0,T]_\tau,$$

which is very easy to compute. It just satisfied the zeroth order consistency condition.

For the inner optimization iteration in step 2d, any algorithm is possible. We used the MATLAB (registered trademark of The MathWorks, Inc.) function `fmincon`, exploiting the active-set algorithm and using the option setting {'TolCon', 1e-6, 'TolX', 1e-6, 'TolFun', 1e-6}. To ensure convergence of the SBO, we enhanced each surrogate optimization in step 2d by restricting the current step-size to a certain trusted region δ_k. The gradients used in this inner optimization were supplied externally as finite difference approximations, but took special care about the step-sizes for their computation.

We used the absolute difference (measured in the Euclidean norm) between two successive iterates u_k and u_{k-1} as well as a lower bound for the trust-region radius δ_k as stopping criterion for the outer iteration (over k). The inner optimization for each surrogate (i.e., for each k) is terminated after $i_k = 10$ iterations (for all k), except in the examples below using a hybrid strategy (where it varies between $i_k = 2, 3$).

7 Optimization Results

In this section we present a brief summary of results of the two optimization methods for a parameter optimization problem for the above presented ecosystem model. For both methods, results for twin-data experiments are available. In such kind of experiments, model-generated data (with parameters u_d) are used to evaluate the applicability and computational efficiency of the methods.

7.1 Surrogate-Based Optimization

Results for the SBO method for this application were in detail presented in [21]. Therein, the coarse model uses a constant number $j^c = j_k^c = 25$ (for all k) of iteration steps in the state equation solver, compared to $j^f = j_k^f = 3,000$ (also constant for all k) ones in the original fine model. On these results, we thus here give only a brief summary. Additionally, we present some recent results obtained using a hybrid SBO strategy that uses different values of j_k^c in the different optimization steps.

The number $j^c = 25$ of coarse model iterations used in [21] leads to a poor identification of two of the seven parameters, see also Fig. 1. The usage of higher values of j^c or a hybrid strategy solved this problem. Concerning performance, Figs. 1 and 2 show that the gain compared to the direct optimization can be significantly enlarged when using a hybrid strategy. On the other hand, finding adequate sequences of inner optimization steps $(i_k)_k$ and number of coarse model iteration steps $(j_k^c)_k$ requires considerable testing or experience.

7.2 One-Shot Optimization

The results for the One-shot optimization available so far are preliminary and not that detailed as the one for the SBO approach. A difference by design of the method is that, in its current version, the One-shot method does not treat parameter bounds explicitly, whereas the SBO method can take them into account in the inner optimization loop. The considered model problem is the same as for the SBO method, concerning (model-generated) data y_d, the least-squares cost function (10), and the underlying simulation model. Here we show results for a version of the

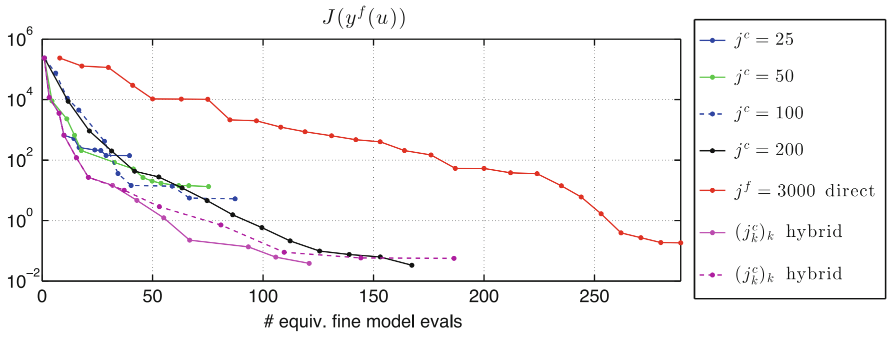

Fig. 1 Cost function value J during SBO runs using (1) different constant coarse models' iteration numbers j^c, (2) two hybrid strategies both with $(j_k^c)_k = (25, 100, 200, 200, \ldots)$, but once $i_k = 3$ for all k (*top*) and the other time $(i_k)_k = (3, 2, 2, \ldots)$, (3) a direct fine model optimization. The computational cost of the optimization is decreased by about 75–90 %

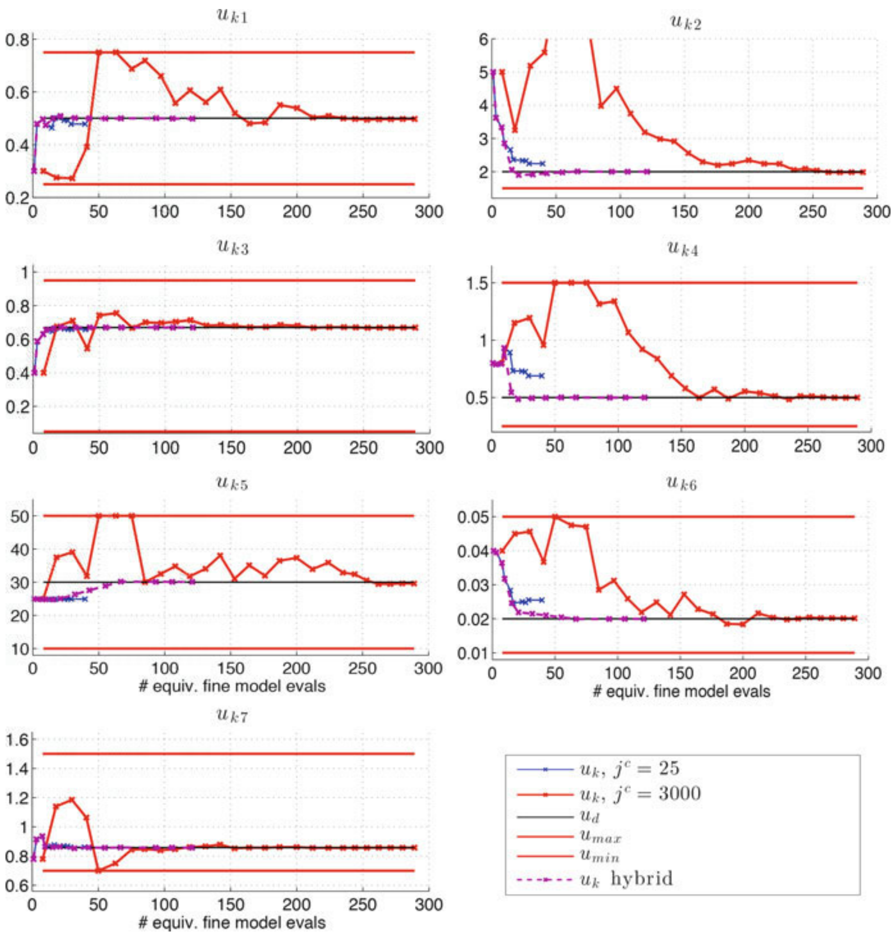

Fig. 2 Convergence of the model parameters $(u_{ki})_{i=1,\ldots,7}$ during the optimization (step counter k), here only for one SBO run, one hybrid run and for the direct fine model optimization. Also shown are target parameter vector u_d and the constant bounds u_{min}, u_{max}

one-step variant ($j_k = \bar{j}_k = 1$), i.e. after one iteration of the state equation solver one for the adjoint and another for the parameter is performed. The only exception is the first step, where a higher number j_0 of state iterations are performed before the first adjoint step.

In this one-step variant, one of the seven parameters provides some problems. The parameters and therewith the tracer concentrations oscillate and the cost function is not reduced anymore after a certain time even though the weighting factor α was chosen very large ($\alpha = 100$) to force parameters towards u_{guess}, see the plot on the left of Fig. 3. The problematic parameter turned out to be $u_7 = b$,

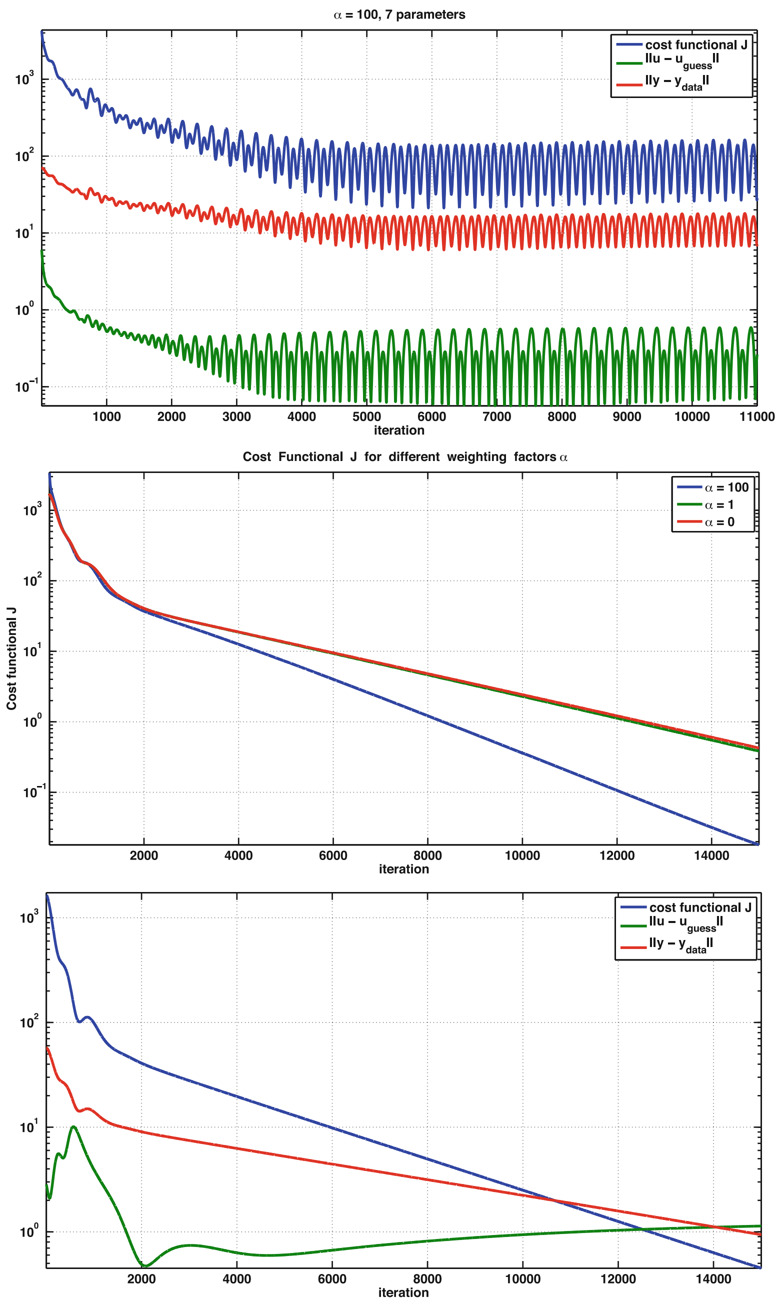

Fig. 3 Results of the One-shot method for seven parameters showing oscillations (*top*), for six parameters, $u_{guess} = u_d$ and different α (*middle*), and for $u_{guess} \neq u_d$ and $\alpha = 0.01$ (*bottom*)