Preface

Give a man a fish, feed him for a day.
Teach a man to fish, feed him for a lifetime.
– Chinese Proverb

The objective of this text is to present the mathematical methodology known as *parameter continuation* in the context of a treatment that lends equal importance to theoretical rigor, algorithm development, and software engineering. It has been the authors' experience, in the development of the text and the associated software platform, that great mileage comes from accounting for all three of these fundamental pillars of computational science and engineering throughout the design process. Each benefits from careful attention to the other two, collectively ensuring successful and sharable implementations and trustworthy numerical results. In contrast, delayed attention to any one of these aspects likely leads down the road of dead ends, poor documentation, and incompatible computational and theoretical formulations. Although well-trodden, this is hardly a path of progress. If nothing else, we hope that the reader takes this message to heart in all subsequent pursuits in applied mathematics.

The theoretical concept of parameter continuation, as described in the text, is based on the simple observation that solutions to parameterized mathematical equations often belong to solution families, in turn parameterized by the problem parameters. Indeed, the exceptions to this situation continue to form a fertile field of study for generations of applied mathematicians. To a degree, it is possible to suggest that much of applied mathematics emphasizes those exceptional, singular points, where the simple picture breaks down; and that the subject then proceeds to investigate the solution behavior on neighborhoods thereof. Parameter continuation is the complementary tool to such a study, whereby solutions located near exceptional points in parameter space may be traced far outside of their immediate neighborhoods, in many cases until other singular points are reached and the analysis repeats.

To the beginning student of nonlinear equations, we expect that a truly mature appreciation for the subject will come only with the opportunity

- to explore the character and persistence of solutions across parameter domains, wherein no closed-form mathematical analysis is available, and
- to translate novel problem formulations into algorithms suitable for parameter continuation.

A trifecta of accomplishment is the final encoding of such novel problem formulations within a framework that supports their embedding as elements of more complicated continuation problems. An initiate of the paradigm advocated herein is a master of all three levels of understanding and skill: able to intuit the behavior of solutions, to translate statements about their existence to computational algorithms, and to support their modular interface to other larger formulations.

It is vital to recognize that no amount of computation is a substitute for a careful reality check, grounded in the basic and advanced techniques of algebra, calculus, and numerical analysis. The reader will, likely from experience, recognize the difference between being able to compute—so aptly made possible by an ever-growing set of high-level platforms—and being able to argue that what is computed is indeed the desired solution. This is particularly evident when infinite-dimensional problems are reduced, through some method of discretization, to problems of finite dimension, and ultimately approximated by finite-precision arithmetic in a numerical implementation. Examples include boundary-value problems for ordinary differential equations, partial differential equations, integral equations, and combinations thereof. Suffice it to say that confidence in the results of numerical implementations should come only as a result of combining computational tools with theoretical results about existence and convergence, without which a significant degree of skepticism is surely warranted.

Beyond the promulgation of a fundamental paradigm of applied mathematics and computational science and engineering, this text seeks to enable and stimulate its reader, whether novice or adept. We do this with a combination of explicit recipes for implementing classes of continuation problems within the *Computational Continuation Core* (referred to below by the abbreviation COCO) software framework for MATLAB®, developed by the authors, as well as open-ended continuation projects that challenge the reader to take that imaginative leap and believe in the opportunity to be original and creative. To this end, we cite literature only to a minimum, and only in the context of exercises wherein the reader is requested to repeat or reproduce the treatment from an archival source.

It is widely recognized in the applied mathematics community that tools of continuation afford a desirable alternative method of analysis to the more commonly deployed forward-simulation tools that pervade applied design and optimization. Increasingly, this realization is being appreciated also by those in industry whose products need to accommodate a range of design objectives, while being certifiably safe within a challenging set of performance criteria. As a means of reducing the number of simulations necessary in the design and development cycle, continuation methods, when applied to intelligently formulated continuation problems, are likely to become a staple of engineering design. While the recipes and formulations in this text are not production-ready for this context, we hope they afford the reader the necessary foundation for a skillful deployment in practice.

The intended audience for this text includes students and teachers of nonlinear dynamics and engineering, as well as scientists and engineers engaged in modeling and simulation. Of particular interest to us are potential developers of continuation toolboxes. The development framework mentioned above provides a broad suite of basic functionality, common across a range of continuation contexts, leaving the essential development task to those algorithmic decisions that encode a particular problem class. The text assumes some familiarity with MATLAB programming and a theoretical sophistication expected of upper-level undergraduate or first-year graduate students in an applied mathematics and/or computational science and engineering curriculum. Although no particular experience is required in topics traditionally associated with continuation methods, such as bifurcation analysis or nonlinear dynamics, this certainly could be beneficial. To this end, we highly recommend several comprehensive and quite readable textbooks and edited volumes that capture both the theoretical and computational aspects:

1. W.J.F. Govaerts, *Numerical Methods for Bifurcations of Dynamical Equilibria*, SIAM, Philadelphia, 2000.

2. Y.A. Kuznetsov, *Elements of Applied Bifurcation Theory*, Springer-Verlag, New York, 1998.

3. B. Krauskopf, H.M. Osinga, and J. Galán-Vioque (Eds.), *Numerical Continuation Methods for Dynamical Systems*, Canopus Publishing Ltd., Bristol, UK, 2007.

Krauskopf et al. also contains a range of interesting applications of continuation methods as well as state-of-the-art algorithms for a variety of continuation problems, which are not treated here, or mentioned only in passing.

We have sought to organize this text according to the sequence of steps we expect a majority of continuation projects would take. Thus, in Part I, we introduce a fundamental mathematical paradigm within which the subsequent tool development is framed. We provide an illustration of the utility of parameter continuation in the context of an optimization problem from the calculus of variations in Chap. 1. The relevant terminology and notation are then introduced in Chap. 2. Chap. 3 provides an introduction to the syntax particular to command-line interaction with the COCO framework, as well as in the development of special purpose continuation toolboxes for this platform. The framework is expanded in Chaps. 4 and 5 with a discussion of toolbox development and task embedding as a fundamental approach to continuation problems inspired by software engineering. Mastery of the basic object-oriented principles of these chapters is essential for the subsequent treatment.

Part II of this text presents a sequence of *toolbox templates* that build on the task-embedding paradigm, and that also demonstrate basic algorithms that take advantage of a vectorized formulation of the nonlinear equations. A workhorse throughout this part is the collocation toolbox presented in Chaps. 6 and 7 for discretizing first-order systems of ordinary differential equations. This toolbox appears embedded in a sequence of encapsulating toolboxes for the solution of single-segment boundary-value problems of ordinary differential equations in Chap. 8 and for multisegment boundary-value problems in Chap. 9. The treatment includes toolboxes for continuation of single-segment periodic orbits in smooth dynamical systems, multisegment periodic orbits in hybrid dynamical systems, and quasi-periodic invariant tori in smooth dynamical systems. A generalization of the collocation toolbox to solving the linearized, variational problem along solution trajectories for ordinary differential equations is then shown in Chap. 10. The analysis demonstrates the application of this toolbox, and the task-embedding paradigm, to the continuation of connecting orbits between equilibria and periodic orbits, while characterizing local stability properties of the periodic orbit.

In Part III, emphasis is on the development of *atlas algorithms*, implementations of a finite-state machine for generating a collection of charts that covers a portion of the solution manifold of the continuation problem. Although the COCO framework includes default implementations of general-purpose atlas algorithms, certain problem classes naturally benefit from appropriate design not only of the original continuation problem but also of the algorithm used to cover the corresponding solution manifold. A general theory for atlas algorithms and continuation is presented in Chap. 11. An incremental sequence of implementations of 1-dimensional atlas algorithms is then discussed in Chap. 12. Chap. 13 illustrates the generalization of such implementations to the multidimensional context, with several example implementations for covering 2-dimensional manifolds. Further extensions of the 1-dimensional and 2-dimensional atlas algorithms are provided in Chap. 14 in order to support constraints on the computational domain.

An essential ingredient in parameter continuation is the detection of special points associated with critical properties of the solution or the solution manifold. This concept of *event handling* is the topic of Part IV of this text. Chap. 15 defines event handling in the context of detection and location of special points along curve segments on the solution manifold. The basic syntax for command-line handling of events in COCO is introduced there. The integration of event handling in COCO-compatible atlas algorithms and toolboxes is described in Chap. 16. Finally, we conclude the part with an exploration of the use of *event handlers* for associating particular data or actions with candidate special points. Specifically, Chap. 17 provides template event handlers, using a reverse communication protocol, for enabling a selective treatment of bifurcation points of equilibria and periodic orbits, as well as for automated or semiautomated branch switching associated with special points along the solution manifold.

Part V provides an introductory treatment of the problem of adaptive changes to the discretization of a continuation problem during parameter continuation. We discuss several distinct approaches to adaptation that are supported by the COCO framework and explore their associated computational cost. In Chap. 18, we present two paradigms of adaptation that can be directly accommodated within the framework developed in previous chapters. In the first case, we consider a *brute-force mesh-refinement strategy*, in which continuation is terminated when an estimate of the discretization error exceeds a desired tolerance, and an entirely new continuation problem is formulated in order to accommodate a change in discretization. In the second case, we describe a *comoving-mesh strategy*, in which variables defining the discretization are included among the unknowns and solved for as part of the continuation problem without a change in their number. Chap. 19 describes the use of adaptive changes to the order of truncation in a Fourier approximation of a periodic orbit of a smooth dynamical system. Finally, *moving mesh strategies*, in which adaptive changes to the discretization order and discretization parameters are integrated with an atlas algorithm, are considered in Chap. 20. The analysis concludes by a numerical comparison between the different adaptive discretization strategies when applied to the approximation of homoclinic orbits, as well as in the continuation of canard orbits in a slow-fast dynamical system.

At the end of each chapter, we collect exercises for use in self-study or as course assignments. These range from reflections on the theoretical content of a chapter to implementations in the COCO framework of algorithms and toolboxes that generalize the treatment in the text to broader problem classes. As suggested above, some of the latter take the form of open-ended projects that could be used as an alternative source for summative assessment. The epilogue, Part VI, includes a number of proposed development projects for students and junior investigators.

The COCO framework source code is freely available from http://www.siam.org/books/cs11. This site also includes a user manual as well as extensive complete examples. The example code explicitly included in this text can also be downloaded from this site and reused or developed further without explicit permission from the authors. We ask simply that the source be acknowledged in any derivative work.

This text and the COCO framework are a result of a joint effort by the authors since the inception of their collaboration, following the 2007 workshop on *Advanced Algorithms and Numerical Software for the Bifurcation Analysis of Dynamical Systems* organized in honor of Eusebius J. Doedel's 60th birthday at the University of Montréal in Montréal, Canada. We are indebted for the inspiration for this effort to the creativity of our fellow attendees at this and other similar meetings and hope that this text will contribute to the impact that their work has in the scientific community. Among our immediate colleagues, we kindly acknowledge Claudia Wulff and Jens Starke for their patience with this long-term effort, and Michael Henderson for his hospitality and help with recreating MULTIFARIO within COCO. We also gratefully acknowledge David Barton and Mehdi Saghafi, who as developers of COCO-compatible toolboxes have provided invaluable feedback to us. Our sincere thanks also go to Barbara Brickman for her help with editing the text for language and format. As with any work of this magnitude, the countless hours of debugging, editing, and imagining would not be possible without the endless support of family and friends, especially Helena, Melanie, Even, Lia, and Zachary Dankowicz and Bärbel, Anette, and Karsten Schilder.

Champaign, Illinois, and Lyngby, Denmark, October 2012

Notation

We collect below a summary of notational conventions and terminology relied upon in this text, in the event that these are not detailed in the main body. As anyone who has ever attempted authorship of a text of this sort will appreciate, notational consistency is a monumental, but invaluable task. If anything, this summary has been useful in reminding us of the intended meaning of our chosen notation.

Index sets

An index set is a finite sequence of nonrepeating, positive integers. We use the capital letters $\mathbb{I}$, $\mathbb{J}$, $\mathbb{K}$, and $\mathbb{L}$ to represent index sets. The capital letters $\mathbb{R}$ and $\mathbb{Z}$, on the other hand, are used to denote the spaces of all reals and all integers, respectively. The capital letter $\mathbb{S} := \mathbb{R}/2\pi$ denotes the quotient set on $\mathbb{R}$ obtained by defining two numbers as equivalent if they differ by a multiple of 2π.

Denote by $\{k\}_{k=1}^{n} := \{1,\ldots,n\}$ the sequence of positive consecutive integers between 1 and n, for some positive integer n, and let $\{1,\ldots,0\} = \emptyset$. Then $\mathbb{K} \subseteq \{1,\ldots,n\}$ provided that the elements of $\mathbb{K}$ all belong to $\{1,\ldots,n\}$. In contrast to $\{1,\ldots,n\}$, the elements of $\mathbb{K}$ need not be in increasing order.

Given an index set $\mathbb{K}$, the notation $|\mathbb{K}|$ denotes its *cardinality*, i.e., the number of integers contained in $\mathbb{K}$. It follows that

$$\mathbb{K} := \{k_i\}_{i=1}^{|\mathbb{K}|}, \tag{1}$$

where $k_i \in \mathbb{Z}$ for all $i \in \{1,\ldots,|\mathbb{K}|\}$. Given an index set $\mathbb{K} \subseteq \{1,\ldots,n\}$, we define the set

$$\{1,\ldots,n\} \setminus \mathbb{K} \tag{2}$$

as the index set obtained by omitting the elements of $\mathbb{K}$ from the sequence $\{1,\ldots,n\}$.

An index set can be used to extract a subsequence from a given sequence of scalars. For example, if $u \in \mathbb{R}^n$ and $\mathbb{K} = \{k_i\}_{i=1}^{|\mathbb{K}|} \subseteq \{1,\ldots,n\}$, then

$$u_{\mathbb{K}} = \{u_k\}_{k\in\mathbb{K}} := \left\{u_{k_i}\right\}_{i=1}^{|\mathbb{K}|}. \tag{3}$$

Functions

Throughout this text, we identify functions in terms of the nature of their domain and range spaces, as in $f : \mathbb{R}^n \to \mathbb{R}^m$, or in terms of their explicit evaluation, as in

$$f : u \mapsto \begin{pmatrix} u_1^2 - u_2 \\ u_1 \end{pmatrix} \tag{4}$$

or

$$f(y,p) := \begin{pmatrix} p_1 y_1 - p_2 y_2 \\ 1 + y_2^2 \end{pmatrix}. \tag{5}$$

We often reference functions only by their name, as in f, rather than their value, as in $f(x)$, unless the latter is preferred by the context. The evaluation of a function f at a particular point $x = x^*$ in its domain is also sometimes denoted by $f|_{x=x^*}$. In figures, we use function names to represent coordinates or as a reference to functions, unless the interpretation is ambiguous.

For derivatives of functions of a single variable, we either use the Leibniz operator notation $\frac{dy}{dx}$ or rely on a superscript to denote differentiation with respect to the independent variable, e.g., $y'(x)$ or $h'(t)$. In the case of functions of multiple variables, we try to use the ∂ symbol consistently in order to distinguish between different partial derivatives. In this context, the notation $\partial_{(x,p)}$ denotes a matrix of derivatives with respect to the components of x and p.

Software

The *Computational Continuation Core* (COCO) framework described above is a package of MATLAB routines, libraries, and classes that implement the core functionality described in this text and that support the command-line and toolbox interface illustrated throughout all its parts. One of the key design features of this framework is the possibility to substitute alternate implementations of various numerical algorithms for those that may be assigned by default. To this end, COCO relies on the use of global and local project settings for identifying the source of key algorithms. As an example, consider the encoding below of the `coco_project_opts` function.

```
function prob = coco_project_opts(prob)

prob = coco_set(prob, 'cont', 'linsolve', 'recipes');
prob = coco_set(prob, 'cont', 'corrector', 'recipes');
prob = coco_set(prob, 'cont', 'atlas_classes', ...
  { [] 'atlas_kd' ; 0 'atlas_0d_recipes' ; 1 'atlas_1d_recipes' });

end
```

This uses COCO core utilities to assign references to the source of the linear and nonlinear solvers and default atlas algorithms for use in the application of the coco entry-point function. In each example given in this text, a function file with this content is included in the same directory as the example files. Where appropriate, we make explicit any overriding of its default references.

Throughout this text, the prompt >> is used to indicate the beginning of a command line in MATLAB. When we speak of an *extract*, we mean a verbatim copy of a portion of the MATLAB command window. In the first several parts of the text, we include these extracts in their full glory, but successively begin to omit these in later chapters, as the novelty wears off. In MATLAB, the use of a single variable name to represent both an input and an output argument takes advantage of a call-by-reference mechanism that guarantees that changes to the corresponding variable survive the execution of the function without temporary duplication of its content within the function. Finally, throughout this text, *string identifiers* are used to reference various primitive and composite objects. Such identifiers must adhere to the rules that apply to MATLAB field names.