TECHNICAL RESEARCH REPORT

Objects for MWR

by Raymond A. Adomaitis

TR 2001-41

Objects for MWR

Raymond A. Adomaitis
Department of Chemical Engineering and
Institute for Systems Research
University of Maryland, College Park MD 20742
adomaiti@isr.umd.edu

September 26, 2001

Keywords: Distributed parameter systems, object-oriented programming, Galerkin's method, spectral methods, weighted residual methods

Abstract

A computational framework has been developed for step-by-step implementation of global spectral projection methods used for solving boundary-value problems and analyzing solutions produced using the numerical techniques of this framework. A set of MATLAB-based functions corresponding to each step in a Galerkin discretization procedure has been developed with emphasis on simplifying the implementation of discretization methods for nonlinear, distributed-parameter system models in up to three-dimensional physical domains. A key feature of this computational approach is that a set of object classes were developed to facilitate implementation of the weighted residual methods (MWR) in an effort to make the connection between the solution procedures and modeling equations as clear as possible. The utility of the computational procedures is demonstrated through applications to two-dimensional reaction-diffusion and fluid flow problems, and a three-dimensional heat transfer problem in semiconductor manufacturing.

1 Introduction

Boundary-value problems (BVPs) in relatively simple geometries define an important class of models describing chemical engineering process systems. Indeed, one can view this class as falling in between highly-simplified, lumped-models and those models generated by complete, highly detailed analyses generating PDEs defined in complex physical domains. In deciding what degree of modeling is necessary for a particular application, a balance must be struck between the level of detail that is attempted to be captured in the model under development and uncertainty in the physical and chemical mechanisms defining the model, and so these "intermediate-level" BVP models can provide a great deal of utility in many engineering applications. One example is chemical vapor deposition processes for electronic materials manufacturing, where distributed models are required to describe across-wafer deposition nonuniformity: in many of these systems, the complexity of equipment design and deposition mechanisms may offset any simulator accuracy gained by fine-tuning detailed, CFD-type calculations. It is these situations that put a premium on flexible simulation strategies where models can be easily modified to test modeling assumptions.

The motivation for the research discussed in this paper is the clear connection object-oriented programming implementations of global spectral methods can create between the BVP model and the MWR used to solve it. In this paper we focus solely on developing these computational procedures for implementing the Galerkin projection. Our goal is to make as clear as possible the connection between model development, discretization, computational implementation, solution interpretation, and application of the simulation results. The computational approach presented in this paper for implementing global spectral methods

simplifies rigorous error analysis, makes possible clear distinctions between spurious and true numerical solutions [2], facilitates implementation of modern model reduction methods [19], and can be integrated with parameter identification, optimization, and other numerical tools for developing validated, predictive simulators [8].

1.1 Computational methods for MWR

Well-established computational procedures exist for implementing the collocation and other MWR, based on both globally and locally defined trial function expansions. Traditionally, (Fortran-based) software developed to implement these methods tended to fall into the categories of programs written for a specific implementation of one element of an MWR solution procedure, or software packages based on one of the MWR solution methods designed for solving a specific problem type. Many of the chemical engineering studies making use of the orthogonal collocation method of Villadsen and Stewart [29] relied on the collocation discretization array subroutines of Villadsen and Michelsen [28], software which falls into the first category. Software in the second category includes the spline-collocation based BVP solver PDECOL by Madsen and Sincovec [14] and the BVP solver COLNEW by Ascher and co-workers [4, 5].

Recent advances in developing environments for scientific computing (such as MATLAB), advances in spectral filtering [13] and other fundamental numerical methods applicable to global spectral projection methods, and increased interest in object-oriented programming methods have contributed to a renewed interest in developing BVP and PDE solvers. Representative software developments include the PseudoPack algorithms of Don and Solomonoff [11] consisting of discrete differentiation, fast transform, and filtering algorithms, the object-oriented (C++) PDE solver of Langtangen and Munthe [17], and the Diffpack finite element package itself [16]. A significant portion of these recent efforts have gone into developing MATLAB-based or MATLAB-compatible software for PDE and BVP systems. Examples include the 3D finite-element based commercial FEMLAB software package, the 2D MATLAB PDE toolbox, the differentiation array suite of Weideman and Reddy [30], and a number of new functions built into Version 6 of MATLAB for solving 1-dimensional BVPs using collocation on cubic splines [24]. Excellent overviews of MATLAB-based numerical techniques for BVPs can be found in the textbook by Cooper [10] or the spectral methods text of Trefethen [27].

The numerical techniques developed in this work contribute to this body of software in that our goal was to develop object-oriented computational tools consisting of a common set of numerical techniques for implementing spectral projection methods inside the MATLAB computational environment. Our intention was to identify the numerical elements common to different applications of global spectral projection methods and then to develop MATLAB functions that form a one-to-one correspondence between the subroutines and

the elemental steps of a solution procedure. Furthermore, we wanted to assess not only the accuracy of the elements of MWR implementation (such as generating basis function sequences, inner product calculations, etc.) but also provide tools and techniques for assessing the accuracy of the solutions computed with these methods. The conceptual goal of making simple weighted inner product computations, computing eigenfunction sequences, etc., was to provide the next step in a "rapid-prototyping" approach to simulator development for distributed parameter systems with simple methods to accurately assess discretization error.

2 Quadrature-based projection methods

In general, we are interested in spectral discretization methods in which approximate solutions to a boundary-value problem are represented by the truncated trial function expansion

$$\bar{u}(x, y, \ldots) = \sum_{i,j,\ldots=1}^{I,J,\ldots} a_{i,j,\ldots}\phi_i(x)\psi_j(y)\ldots \tag{1}$$

In the representative case of a two-dimensional physical domain, the basis functions $\phi_i(x)\psi_j(y)$ are defined globally in the computational domain

$$\Omega: \quad 0 \le x \le 1 \quad 0 \le y \le 1$$

and the basis functions exist in an inner product space defined by

$$\langle f(x,y), g(x,y)\rangle = \int_0^1 \int_0^1 f(x,y)g(x,y)\, x^\alpha \, dx\, dy.$$

In our computational methods, the one-dimensional basis function components $\phi_i(x)$, $\psi_j(y)$, are represented as vectors of the function values at a set of quadrature points $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, respectively (Fig. 1). For example, in the case of the function sequence $\{\phi_i(x)\}_{i=1,\ldots,I}$, the n quadrature points are defined as the combination of the unit interval endpoints and the $n-2$ roots of a shifted orthogonal Jacobi polynomial $J_{n-2}^{\alpha+1,\beta+1}(x)$, a polynomial sequence orthogonal with respect to inner product weight $x^\alpha(1-x)^\beta$ where $J_0^{\alpha+1,\beta+1} = 1$; $\alpha = 0, 1$, or 2, and corresponds to the slab, cylindrical, or spherical geometries, respectively ($\beta = 0$ in this work). Equidistant points $\hat{x}_k = k/n$ are used for periodic physical domains. In all cases, it is required that $n \ge I + 2$; a more detailed discussion on selecting the value n can be found in Section 2.4.

Numerical computation of the quadrature points can be carried out using several approaches [9]; in this work, a two-step procedure is employed, consisting of a root-bracketing and linear interpolation procedure to identify approximate root locations as the first step, followed by Newton iterations to refine the locations of the roots. Recurrence formulas for the Jacobi polynomials and their derivatives are used in each step and efficient computational procedures have been developed for very high degree discretizations [9].

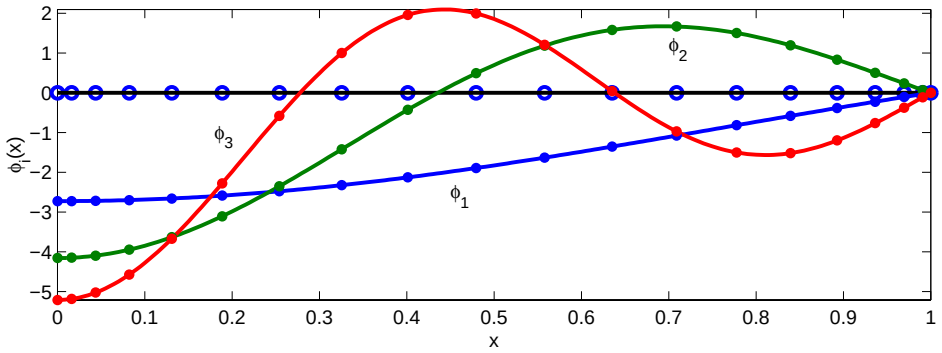

Figure 1: Basis functions represented on the quadrature grid; inter-point values are computed using Lagrange interpolation.

2.1 Interpolation

Lagrange interpolation is used to represent the basis functions at inter-quadrature point locations (see Fig. 1):

$$\phi_i(x) = \sum_{j=1}^{n} \Phi_{j,i} l_j(x) \tag{2}$$

where $l_j(x) = \prod_{k=1,k\neq j}^{n}(x - \hat{x}_k)/(\hat{x}_j - \hat{x}_k)$, $j = 1, \ldots, n$. The coefficient $\Phi_{j,i}$ represents the value of the function $\phi_i(x)$ at quadrature point $\hat{x}_j$ by the definition of the interpolation polynomial (trigonometric functions are used in the case of periodic physical domains). Lagrange interpolation methods form the basis by which quadrature weights are computed in the following section. Computational implementation of interpolation methods is carried out recursively using Neville's algorithm [22] or directly using discrete-transform arrays defined with Jacobi polynomial sequences evaluated on the quadrature and interpolation grids [9].

2.2 Quadrature

Well-known quadrature weight formulas exist for computing $\hat{\mathbf{w}}$ (e.g., [23]) and further modifications to improve the accuracy for high-degree interpolating polynomials have been discussed in the literature [20]. The Gauss-Lobatto quadrature guarantees that the quadrature weights $\hat{\mathbf{w}}$ used to compute

$$\int_0^1 f(x)\, x^\alpha dx = \hat{\mathbf{w}}^T \hat{\mathbf{f}}$$

result in numerically exact integral evaluations (limited by round-off errors in the computational procedures) if f is a polynomial with degree less than $q = 2n-3$. Therefore, in the context of the inner product definition

$$\int_0^1 \phi_i(x)\phi_j(x)\, x^\alpha dx = \langle \phi_i, \phi_j \rangle\,, \tag{3}$$

the $\hat{w}_k$ can be considered as the discrete approximations to the differential element $dv = x^\alpha dx$. For periodic domains, the quadrature weights are $\hat{w}_k = 1/n$, $k = 1, \ldots, n$.

2.3 Discrete differentiation operations

Explicit formulas exist for derivatives of the interpolation polynomials $l_j(x)$ (and the trigonometric functions used for periodic physical domains). Therefore, it is straightforward to derive discrete-ordinate formulations of the first-order derivative $d\hat{\mathbf{T}}/dx = \hat{\mathbf{A}}\hat{\mathbf{T}}$ and Laplacian operator $\nabla^2\hat{\mathbf{T}} = \hat{\mathbf{B}}\hat{\mathbf{T}}$ that give numerically exact results for polynomial functions defined on the quadrature grid. More accurate and efficient computations of the discrete differentiation operators are based on the discrete transform and differentiation arrays produced by Jacobi polynomial recurrence formulas; details are discussed in [9].

2.4 Convergence of functions approximated on the quadrature grid

In our MATLAB implementation of the modified numerical techniques, we have found that these computations can be accurately carried out to over 1000 discretization points. The effect of such finely discretized function representations is that numerical computations on this grid can be treated as (and under some circumstances are) exact, within the limits set by the number of discretization points n. In general, we find $n = 2(I + 2)$ gives satisfactory computational accuracy for quadrature operations. Further details on the computational methods, accuracy, and computational costs of the numerical methods can be found in [9].

2.5 Basis function sequences

Fundamental to spectral projection methods is the definition of the basis functions. Orthogonal polynomial sequences can be readily generated on the quadrature grid using recurrence formulas, and subsequently can be normalized numerically by quadrature. Likewise, we find it convenient to define a basis function sequence by the solutions to Sturm-Liouville problems over $0 < x < 1$ in the form

$$\frac{1}{x^\alpha}\frac{d}{dx}x^\alpha\frac{d\phi}{dx} = \lambda\phi$$

subject to

$$\begin{aligned} a\frac{d\phi(0)}{dx} + b\phi(0) &= 0 \\ c\frac{d\phi(1)}{dx} + d\phi(1) &= 0. \end{aligned}$$

These functions are orthogonal with respect to the quadrature-based inner product operations and are typically normalized prior to use. Eigenfunctions of this form are automatically generated in the BFUN object constructor method described in Section 3.

2.6 n-dimensional array operations

BVPs defined in three-dimensional physical domains give rise to three-dimensional mode amplitude coefficient arrays with elements $a_{i,j,k}$; direct methods for computing the steady-state solution of these systems subsequently generate Jacobian arrays with six-dimensions. While one can re-index the mode-amplitude, residual projection, and Jacobian arrays to obtain conventional one- and two-dimensional arrays, we found implementation procedures were clearer if the variables and equations were left in their original form. This motivated developing computational methods to generalize matrix multiplication to these higher-dimensional systems. For example, if

$$\begin{aligned} \mathbf{A}^{(L_1\times L_2\times\ldots\times L_p)\times(M_1\times M_2\times\ldots\times M_q)} &= \mathbf{A}^{\mathbf{L}\times\mathbf{M}} \\ \mathbf{B}^{(M_1\times M_2\times\ldots\times M_q)\times(N_1\times N_2\times\ldots\times N_r)} &= \mathbf{B}^{\mathbf{M}\times\mathbf{N}} \end{aligned}$$

implies the generalized matrix multiplication operation

$$\mathbf{A}^{\mathbf{L}\times\mathbf{M}}\mathbf{B}^{\mathbf{M}\times\mathbf{N}} = \mathbf{C}^{\mathbf{L}\times\mathbf{N}}$$

where

$$C_{l_1,l_2,\ldots,l_p,n_1,n_2,\ldots,n_r} = \sum_{k_1=1,\ldots,k_q=1}^{M_1,\ldots,M_q} A_{l_1,l_2,\ldots,l_p,k_1,k_2,\ldots,k_q} B_{k_1,k_2,\ldots,k_q,n_1,n_2,\ldots,n_r}$$

Given this definition, we can define the generalized transpose operation as $\left[\mathbf{A}^{\mathbf{L}\times\mathbf{M}}\right]^{\mathrm{T}} = \mathbf{A}^{\mathbf{M}\times\mathbf{L}}$, a square array as $\mathbf{A}^{\mathbf{L}\times\mathbf{M}}$ such that $L_1 = M_1, L_2 = M_2, \ldots L_q = M_q$, and $p = q$, the identity array as a square array $\mathbf{A}$ with all zero elements except $\mathbf{I}^{\mathbf{L}\times\mathbf{L}} : I_{l_1,l_2,\ldots,l_q,l_1,l_2,\ldots,l_q} = 1$, and the matrix inverse as $\left[\mathbf{A}^{\mathbf{L}\times\mathbf{L}}\right]^{-1}\mathbf{A}^{\mathbf{L}\times\mathbf{L}} = \mathbf{I}^{\mathbf{L}\times\mathbf{L}}$. Because these matrix operations are not found as part of the standard MATLAB function library, functions `mdiag.m`, `mprod.m`, and `msolve.m` were developed to create $L \times L$-dimensional diagonal arrays, to perform the generalized matrix multiplication, and to solve by Gaussian Elimination systems described by these high-dimensional arrays.

3 Object classes for MWR

Object-oriented programming concepts can significantly reduce the complexity of implementing the global spectral projection methods. The object-oriented programming features are implemented in the context of MWR computations by identifying those data structures that remain unchanged during a solution procedure and creating a corresponding set of methods that operate on these new objects. In particular, our focus is on discretization methods applied to problems defined in 2 or more physical dimensions with the goal of making the MATLAB code written in the course of solving the problem as compact and as close in syntax to how one would normally write the MWR solution procedure steps.

Our approach to achieving these goals is to define a set of new MATLAB object classes. New object classes in MATLAB are defined by extensions of the MATLAB STRUCT object class; the new and overloaded methods of the class are placed in a separate directory (named @*class-name*) that also contains the constructor method(s) for that class. As one example, consider the scalar-field object class SFIELD which is used to define solutions and other functions in the physical space. Objects of this class have two data fields:

S.pd : *A QGRID object defining the physical domain*
S.val : *A DOUBLE array defining the scalar field values at the quadrature points*

Having defined the data fields of this object class, constructor, display, plotting, and perhaps most importantly, weighted inner product methods are defined for this class. Currently, all object classes created for this project have equal precedence, therefore MATLAB will search for the first appropriate method for the class of objects in the function parameter list, starting from the left-most parameter. The specifics for each new object class will be discussed in the following sections.

3.1 Physical domain object classes

When solving a problem using quadrature-based projection methods, one must recompute the differentiation and quadrature arrays if the number of quadrature points is changed. The fixed relationship between the quadrature points, differentiation and quadrature weight arrays, and coordinate axis names leads naturally to encapsulating these data into a single object; this motivated developing the QGRID object class. Objects within this class contain a physical-space grid of quadrature points and the above-mentioned arrays. No methods were created to modify the data fields of QGRID objects once they are constructed; a new QGRID object must be created if any changes in geometry or grid size are needed. Methods of this class include accessor (`get.m`), grid visualization (`plot.m`), and constructor methods. QGRID (or QGRIDC objects, which are defined next) are aggregated into objects of every other class defined in this paper.

Class name: **qgrid**	
Data fields	({char}) geom ({double}) **qp**, ({double}) **w**, ({double}) **d**, ({double}) **dd**, ({double}) **Q**, ({char}) name
Constructor methods	(qgrid) A = qgrid((char) geom, (double) ndisc, (char) name, ...) (qgrid) A = qgrid((char) geom, (double) **qp**, (double) **w**, (double) **d**, (double) **dd**, (double) **Q**, (char) name)
New methods	display((qgrid) A) B = get((qgrid) A, (char) field, (char) coord)
Overloaded operators	none
Overloaded methods	plot((qgrid) A)

In many problems, it is necessary (or more convenient) to set up several physical domains inside or adjacent to the primary QGRID object (e.g., spherical catalyst pellets inside a tubular chemical reactor). The QGRIDC object class was created to describe these sub- (child) domains; objects of this class inherit the data fields and methods of the QGRID class and add several new data fields to describe the relative positions of the child and parent physical domains.

Class name: **qgridc**	
Data fields	Inherits data fields of QGRID object and adds the following: ({double}) **shift**, ({double}) **scale**, (qgrid) Y, (char) bcloc
Constructor method	(qgridc) A = qgridc((double) **shift**, ({double}) **scale**, (qgrid) Y, (char) bcloc)
Inherited method	@qgrid/display((qgridc) A)
New method	none
Overloaded operators	none
Overloaded methods	B = get((qgridc) A, (char) field, (char) coord) plot((qgridc) A)

3.2 Basis function object class

One of the key elements of implementing a global spectral projection discretization method is defining the basis functions. The BFUN object class was created to store basis function sequences, eigenvalue arrays (if the basis functions are eigenfunctions), and the physical domain over which the basis functions are defined (a QGRID object). Each basis function in each sequence is discretized at the quadrature points of the QGRID object; the basis functions themselves can be approximate solutions to a Sturm-Liouville problem, can be

generated from a recurrence relation, or by any other means.

In multidimensional applications, a single BFUN object is used to store all basis function sequence components, and this BFUN object is created using the overloaded $*$ operator. For example, if

$$\begin{aligned} \text{object Phi} &= \{\phi_i(x)\}_{i=1,\ldots,I} \\ \text{object Psi} &= \{\psi_j(y)\}_{j=1,\ldots,J} \\ \text{object P} &= \text{objects Phi} * \text{Psi} = \{\phi_i(x)\psi_j(y)\}_{i=1,\ldots,I\ j=1,\ldots,J} \end{aligned}$$

Having described methods for creating BFUN objects, the most important method of the class is the weighted inner product method `wip.m`. This method is used for projecting scalar field (SFIELD) objects onto basis function sequences, and for computing inner products of basis function sequences with other sequences; further information can be found in the table below and in the representative applications found in the remainder of this paper.

Class name: **bfun**	
Data fields	(qgrid) pd, ({double}) **fun**, ({double}) **eigv**, ({char}) name
Constructor methods	(bfun) A = bfun((qgrid) X, ({double}) **fun**, ({double}) **eigv**, ({char}) name) (bfun) A = bfun((qgrid) X, (char) name, (double) a, (double) b, (double) c, (double) d)
New methods	(sfield) B = bfun2sfield((bfun) A, (double) **indices**) display((bfun) A) (bfun) B = even((bfun) A) B = get((bfun) A, (char) field) (bfun) B = modes((bfun) A, (double) **modeno**) (bfun) B = odd((bfun) A) (bfun) B = truncate((bfun) A, (double) **N**)
Overloaded operators	(sfield) C = (double) A * (bfun) B (bfun) C = (bfun) A * (bfun) B (bfun) B = (bfun) A((char) xat)
Overloaded methods	(double) B = eig((bfun) A, (double) codir) plot((bfun) A, (double) nofun, (char) fname) (double) C = wip((sfield) A, (bfun) B) (double) C = wip((bfun) A, (bfun) B)

3.3 Scalar field object class

After defining a BFUN object P containing a particular basis function sequence, a state variable or other function can be reconstructed in the physical space (on the quadrature grid) using the overloaded $*$ operation

of the BFUN class and a DOUBLE array of mode amplitude (Fourier) coefficients:

$$\mathrm{F} = a * \mathrm{P}.$$

An SFIELD object F is created by this operation; an SFIELD object has data fields consisting of the function value at the quadrature points and the corresponding QGRID (or QGRIDC) object itself. SFIELD objects frequently are used to represent residual functions evaluated on the quadrature grid in MWR applications.

Class name: **sfield**	
Data fields	(qgrid) pd, (double) **val**
Constructor methods	(sfield) A = sfield((qgrid) X, (double) **val**)
New methods	display((sfield) A) (sfield) B = expand((sfield) A, (qgrid) pdNew) (sfield) B = extrapd((sfield) A, (qgrid) pdNew, (double) **center**, (double) **scale**) get((sfield) A, (char) field)
Overloaded operators	(sfield) C = (sfield) A - (sfield) B (sfield) C = (sfield) A + (sfield) B (sfield) C = (sfield) A .^ (double) B (sfield) C = (sfield) A ./(sfield) B (sfield) C = (sfield) A .*(sfield) B (sfield) B = (sfield) A((char) xat)
Overloaded methods	(double) B = contour((sfield) A, (double) (**limits**) contourf((sfield) A) plot((sfield) A) (double) C = wip((sfield) A, (bfun) B) (double) C = wip((sfield) A, (sfield) B)

3.4 Linear operator object class

The final object class created as part of this MWR framework is the LOPER class, defining discretized linear operators and their corresponding physical domain (QGRID or QGRIDC objects). LOPER objects and the overloaded $*$ operator are used to differentiate SFIELD and BFUN objects. These operations typically are found as part of generating residual functions, Jacobian elements, or interpreting solutions (e.g., computing a diffusive flux).

Class name: **loper**	
Data fields	(qgrid) pd, ({double}) **difop**, ({char}) name
Constructor method	(loper) A = loper((qgrid) X, (double) **difop**, (char) name)
New methods	B = get((bfun) A, (char) field)
Overloaded operator	(sfield) C = (loper) A * (sfield) B (bfun) C = (loper) A * (bfun) B (loper) B = (double) a * (loper) B
Overloaded methods	none

3.5 Additional functions

A number of functions were developed for operating on the n-dimensional arrays discussed previously, and for filtering, inner product, and other operations. Functions used in applications discussed in this paper are listed below.

Class name: **double**	
New methods	(double) **A** = eigarray((double) **eigv1**, (double) **eigv2**, ...) (double) **A** = fsf((double) p, (double) N) (double) **C** = mprod((double) **A**, (double) **B**, (double) p, (double) q, (double) r) (double) **C**, (double) Cnorm = msolve ((double) **A**, (double) **B**) (double) **C** = msum((double) **A**, (double) **B**) (double) **Ip** = wip((double) **f**, (double) **g**, (double) **w**, (double) dir)

4 An elementary application

The benefits of implementing MWR in an object-oriented framework are illustrated using the trivial numerical problem of computing a solution to the BVP

$$\frac{1}{x}\frac{d}{dx}x\frac{dT}{dx} + 1 = 0$$

subject to $dT(0)/dr = 0$ and $T(1) = 1$. The exact solution, $T(x) = (5 - x^2)/4$, will be used to assess the accuracy of the numerical methods used to compute solutions and to approximate the discretization error. An eigenfunction expansion solution is sought in the form

$$\bar{T}(x) = 1 + \sum_{i=1}^{I} a_i \phi_i(x)$$

where the basis functions $\phi_i(x)$ are defined as Bessel's functions of the first kind of order zero, computed as solutions to the Sturm-Liouville problem

$$\nabla^2\phi = \lambda\phi \qquad \frac{d\phi(0)}{dx} = \phi(1) = 0$$

normalized with respect to inner product (3).

Quadrature grid and basis functions

To solve this system, we define a BFUN (trial function) object Phi to store the $I = 3$ basis functions and their 30-point quadrature grid X defined by the QGRID object created below:

```
X   = qgrid('cyln',30,'x');
Phi = truncate( bfun(X,'x',1,0,0,1), 3);
```

Methods for the BFUN class include weighted inner product computational routines, therefore, the orthogonality of the basis function sequence can be checked with the single statement

```
wip(Phi,Phi)
```

which produces the array of inner-product values

```
 1.00000000000000  -0.00000000000003  -0.00000000000002
-0.00000000000003   1.00000000000000  -0.00000000000002
-0.00000000000002  -0.00000000000002   1.00000000000000
```

The mode amplitude coefficients a_i are readily computed as

$$a_i = -\frac{\langle 1, \phi_i(x)\rangle}{\lambda_i}$$

using the MATLAB statements

```
One = sfield(X,1);
a = -wip(One,Phi)./eig(Phi)
```

and the temperature field in the physical space is computed using the overloaded $*$ operator

```
T = One + a*Phi
```

to create a scalar field (SFIELD) object T. Results are shown in Fig. 2.

Error analysis

The discretization error of truncated basis function expansion solution is expressed in the physical space as the function

$$R(x) = \frac{1}{x}\frac{d}{dx}x\frac{d\bar{T}}{dx} + 1$$

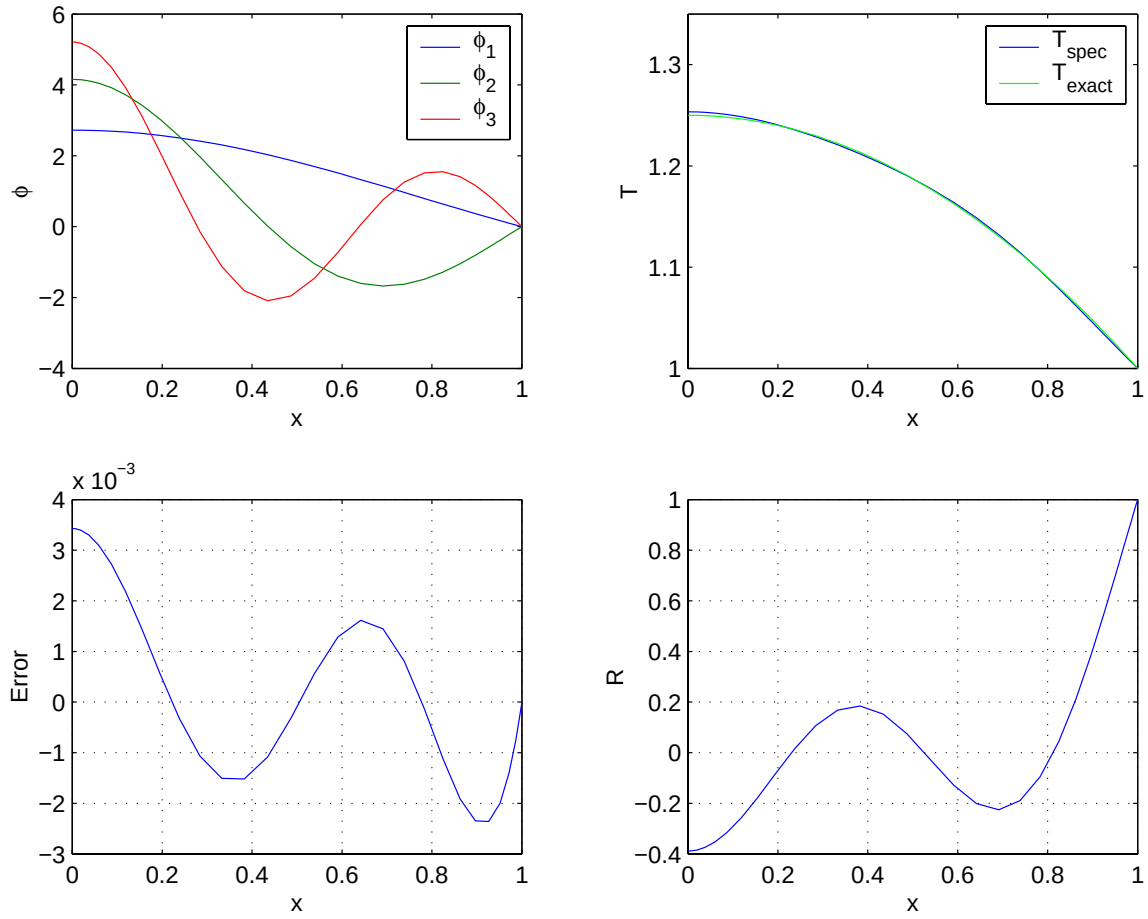

Figure 2: Trial functions (top left) , approximate vs. true solution (top right), discretization error (bottom left), and the residual function (bottom right).

We use the LOPER (linear operator) object DDx to compute the residual function $R(x)$ on the quadrature grid; if the original modeling equation was in dimensional form, plotting $R(x)$ (shown in Fig. 2) would correspond to the error in the energy balance in terms of W/m^3.

```
DDx = loper(X,'dd','x');
R   = DDx*T + One
```

In summary, we see that by extending the basic library of quadrature operations through the used of the object-oriented features of MATLAB, the implementation of Galerkin projection and other MWR can be reduced to a minimum amount of MATLAB code.

5 Steady-state 2D catalyst pellet

Consider the problem of determining the steady-state concentration field inside a cylindrical, isothermal catalyst pellet in which a second-order reaction takes place. If $c(r,z)$ is the reactant species concentration inside the pellet, the modeling equation can be written as

$$\begin{aligned} 0 &= \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial c}{\partial r}\right) + \frac{\partial^2 c}{\partial z^2} - \phi^2 c^2 \\ &= \nabla_r^2 c + \nabla_z^2 c - \phi^2 c^2 \end{aligned} \tag{4}$$

subject to boundary conditions

$$\frac{\partial c(0,z)}{\partial r} = 0 \qquad c(1,z) = 1 \qquad c(r,1) = 1 \qquad \frac{\partial c(r,0)}{\partial z} = 0.$$

The catalyst pellet-phase concentration profile is expressed in terms of the truncated trial function expansion

$$\bar{c}(r,z) = 1 + \sum_{i,j=1}^{I,J} a_{i,j}\eta_i(r)\psi_j(z) \tag{5}$$

where η_i and ψ_j are computed as the eigenfunctions satisfying $\lambda^r\eta = \nabla_r^2\eta$ subject to $\eta'(0) = 0$, $\eta(1) = 0$ and $\lambda^z\psi = \nabla_z^2\psi$ subject to $\psi'(0) = 0$, $\psi(1) = 0$.

The physical domain (QGRID object) is set up and the discrete differentiation operation (LOPER) object is defined using the MWRtools functions:

```
S   = qgrid('cyln',42,'r','slab',40,'z');
DDr = loper(S,'dd','r');
DDz = loper(S,'dd','z');
```

Then we create a BFUN object P corresponding to (5) by:

```
I = 14; J = 14; % truncation numbers
P = truncate( bfun(S,'r',1,0,0,1), I) * ...
    truncate( bfun(S,'z',1,0,0,1), J);
```

The Galerkin projection solution is implemented by substituting (5) into (4), using the current estimate of the solution $\mathbf{a}^\nu$ to define the residual function $R(r,z)$, and projecting the residual onto each trial function to generate the $I \times J$ array **rhs** with elements

$$rhs_{i,j} = \langle R, \eta_i\psi_j \rangle = 0.$$

Linearizing **rhs** at the current solution estimate as part of the Newton-Raphson solution procedure gives the linear system:

$$\mathbf{0} = \mathbf{rhs}(\mathbf{a}^\nu) + \mathbf{Jac}(\mathbf{a}^\nu)[\mathbf{a}^{\nu+1} - \mathbf{a}^\nu]$$

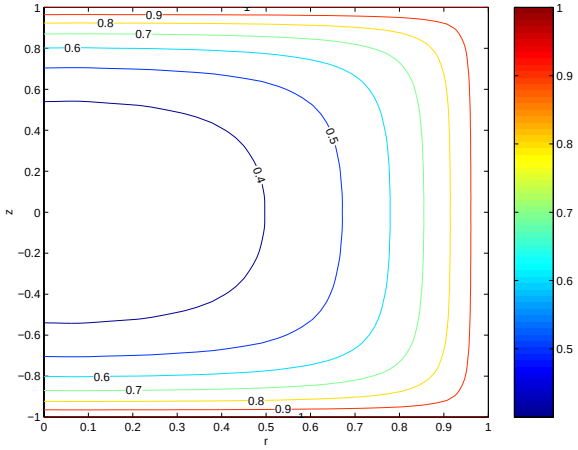

Figure 3: Reactant species concentration in a catalytic pellet at steady state. Trial function truncation numbers along r and z directions are both 14.

$$\mathbf{a}^{\nu+1} \quad = \quad \mathbf{a}^{\nu} - \mathbf{Jac}^{-1}\,\mathbf{rhs}$$

where **Jac** is the $I \times J \times I \times J$ Jacobian array with elements defined as

$$Jac_{i,j,k,l} = \langle \partial R/\partial a_{k,l}, \eta_i \psi_j \rangle \, .$$

Because Gaussian Elimination and a number of other matrix operations are not defined for *ndim* arrays where $ndim > 2$, we use the function `msolve.m` written specially for this purpose. Additional details regarding these numerical procedures were discussed in Section 2.6.

The solution is reconstructed in the physical space using `c = 1 + a * P` where $*$ is the overloaded matrix multiplication operator that accepts a BFUN object argument. These solution steps take the computational form:

```
a   = zeros(I,J); % Solution initial guess
One = sfield(S,1);
tm  = 4.0;        % Thiele modulus

for iters = 1:8
   c = One + a*P;
   R = DDr*c + DDz*c - tm^2*c.^2;
   rhs = wip(R,P);
   Jac = wip(DDr*P,P) + wip(DDz*P,P) - tm^2*2*wip(c.*P,P);
   update = msolve(Jac,rhs);
   a = a - update;
   end
```

The Newton-Raphson procedure above converges within 4 to 8 iterations; representative steady-state solution results are plotted in Fig. 3.

Error analysis

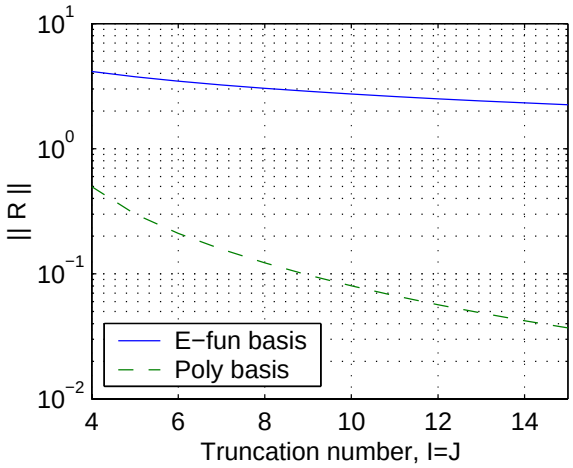

Figure 4: Residual function norm plotted as a function of truncation number for the 2-D catalyst pellet problem comparing the performance of the eigenfunction-based expansion and polynomial basis function solutions.

The residual function norm corresponding to the converged concentration profile solution is computed using

```
Rnorm  = sqrt(wip(R,R))
```

The convergence rate of the Galerkin projection solution as a function of basis function truncation number then can be plotted (Fig. 4) to reveal that while the solution does converge steadily with increasing I, J, better performance can be obtained by using polynomial basis functions of the form

$$\eta_i(r)\psi_j(z) = r^{2(i-1)} z^{2(j-1)} \cos(\pi r/2) \cos(\pi z/2).$$

Plotting the residual function (as was done in Fig. 2) corresponding to solutions computed using the different basis function sequences reveals that the eigen-basis functions do not approximate the residual function well in the neighborhood of the outer boundaries, reducing the convergence rate.

6 Stokes flow in a driven cavity

We consider computing solutions to the Stokes flow problem defined by a cavity filled with liquid set in motion by one cavity wall. This wall is located at $x = 1$ and moves at unit velocity in the axial direction; the remaining walls are stationary. Because we assume both velocity components are zero at the stationary walls, an analytical solution in closed form is not possible due to the jump discontinuity of the boundary conditions at both outer corners [15]. This problem has been studied in the context of modeling plasma flow between adjacent red blood cells moving through a capillary blood vessel [18]; other applications include studies of creeping flow eddy structures and their transitions in rectangular cavities by eigenfunction expansion

solutions to the stream function formulation of the problem (e.g., [6, 15, 21, 25, 26]). In this paper, we present an alternative to the stream-function based solution approaches: a quadrature-based eigenfunction expansion method is investigated where the pressure field is computed from a Galerkin projection of the continuity equation residual.

The equations governing the fluid motion are written in dimensionless form as

$$\begin{aligned} 0 &= -\frac{\partial p}{\partial x} + \frac{\partial^2 v_x}{\partial x^2} + \alpha^2 \frac{\partial^2 v_x}{\partial y^2} \\ 0 &= -\alpha \frac{\partial p}{\partial y} + \frac{\partial^2 v_y}{\partial x^2} + \alpha^2 \frac{\partial^2 v_y}{\partial y^2} \end{aligned}$$

where $\alpha = X/Y$ is the aspect ratio of a rectangular cavity with infinite spanwise dimension. The continuity condition is

$$\frac{\partial v_x}{\partial x} + \alpha \frac{\partial v_y}{\partial y} = 0.$$

Boundary conditions are $v_x = 0$, $v_y = 1$ at $x = 1$ and $v_x = v_y = 0$ at all other walls in the rectangular cavity.

6.1 Basis function expansions

The fluid velocity components and pressure field are represented by globally-defined basis function expansions of the form

$$\begin{aligned} v_x(x,y) &= \sum_{i,j=1}^{M_V} a_{i,j}\phi_i(x)\psi_j(y) \\ v_y(x,y) &= \sum_{i,j=1}^{M_V} b_{i,j}\phi_i(x)\xi_j(y) + \sum_{j=1}^{M_{BC}} c_j\sigma_j x^2 \xi_j(y) \\ p(x,y) &= \sum_{i,j=1}^{M_P} d_{i,j}\gamma_i(x)\delta_j(y) \end{aligned}$$

where the trial function components are computed as nontrivial solutions (including nontrivial eigenfunctions associated with zero-value eigenvalues) to the Sturm-Liouville problems

$$\begin{aligned} \lambda_\phi \phi &= \frac{d^2\phi}{dx^2} & \phi(0) = \phi(1) = 0 \\ \lambda_\psi \psi &= \frac{d^2\psi}{dy^2} & \psi(0) = \psi(1) = 0 \\ \lambda_\xi \xi &= \frac{d^2\xi}{dy^2} & \xi(0) = \xi(1) = 0 \\ \lambda_\gamma \gamma &= \frac{d^2\gamma}{dx^2} & \gamma'(0) = \gamma'(1) = 0 \\ \lambda_\delta \delta &= \frac{d^2\delta}{dy^2} & \delta'(0) = \delta'(1) = 0. \end{aligned} \tag{6}$$

The coefficients σ_j in the axial velocity component basis function expansion are spectral filtering coefficients used to improve point-wise convergence of the solution. More details on filtering methods can be found in [1].

We note that while ψ_i and ξ_i are computed from the same Sturm-Liouville problem, only odd functions are used to define the ψ_i and even for ξ_i; odd functions are used to define the δ_i. The trial functions are orthogonal sequences (and are normalized) with respect to the inner product

$$\begin{aligned}\langle \phi_i\psi_j, \phi_k\psi_l\rangle_{x,y} &= \langle \phi_i, \phi_k\rangle_x \langle \psi_j, \psi_l\rangle_y \\ &= \int_{x=0}^{1} \phi_i(x)\phi_k(x)\,dx \int_{y=0}^{1} \psi_j(y)\psi_l(y)\,dy.\end{aligned}$$

The computational implementation of the solution procedure begins by setting up a 70×70 quadrature grid (QGRID) and discrete differentiation (LOPER) objects in the x and y directions:

```
S   = qgrid('slab',70,'x','slab',70,'y');
Dx  = loper(S,'d', 'x');
DDx = loper(S,'dd','x');
Dy  = loper(S,'d', 'y');
DDy = loper(S,'dd','y');
```

The basis functions are computed as solutions to the corresponding eigenvalue problems listed in (6) and stored as one-dimensional as BFUN objects; odd and even function sequences are generated using the `odd.m` and `even.m` methods of the BFUN class, and all are truncated to the appropriate length using the `truncate.m` method. Representative results plotted with the overloaded `plot.m` method and are presented in Fig. 5.

```
mV = 30; mBC = 8; mP = 24; % truncation numbers

phi = truncate( bfun(S,'x',0,1,0,1), mV); % for the flow field
psi = truncate( odd(  bfun(S,'y',0,1,0,1) ), mV);
xi  = truncate( even( bfun(S,'y',0,1,0,1) ), mV);

x = get(S,'qp','x'); % for nonhomogeneous BC
F = bfun(S,x.^2,[],'x');

gam = truncate( bfun(S,'x',1,0,1,0), mP ); % for the pressure field
del = truncate( odd( bfun(S,'y',1,0,1,0) ), mP);
```

The basis function sequences defined by the dyadic product of the function sequences in the x and y coordinates is performed using the overloaded $*$ operator:

```
Bvx   = phi*psi;
Bvy   = phi*xi;
BvyBC = F*truncate( xi,mBC );
Bp    = gam*del;
```

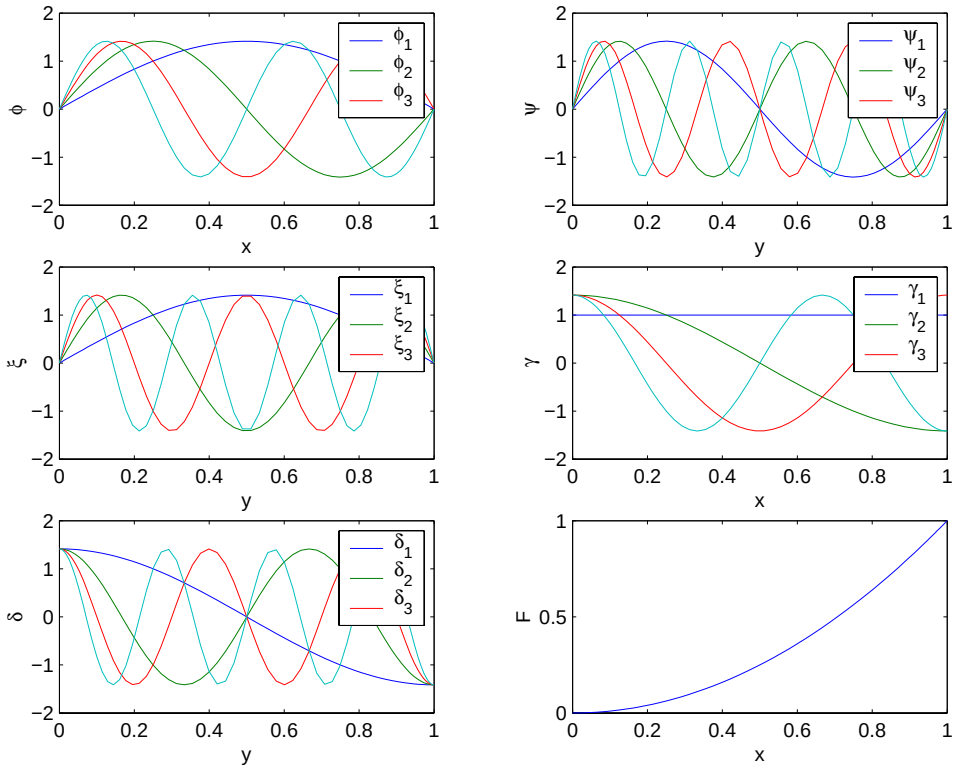

Figure 5: Basis functions for the driven cavity fluid flow problem.

6.1.1 Flow velocity components

Substituting the trial function expansions into the momentum balance equations, we obtain for the velocity component v_x

$$\sum_{i,j=1}^{M_P} d_{i,j}\gamma_i'\delta_j = \sum_{i,j=1}^{M_V} a_{i,j}(\lambda_{\phi_i} + \alpha^2\lambda_{\psi_j})\phi_i\psi_j$$

where the prime denotes differentiation (with respect to x in this case). Solving for the coefficient $a_{p,q}$ by projecting this residual onto $\phi_p\psi_q$, we obtain

$$\begin{aligned} a_{p,q} &= \frac{1}{\lambda_{\phi_p} + \alpha^2\lambda_{\psi_q}} \sum_{i,j=1}^{M_P} \langle\gamma_i'\delta_j, \phi_p\psi_q\rangle_{x,y}\, d_{i,j} \\ \mathbf{a} &= \mathbf{A}\mathbf{d} \end{aligned}$$

where the 4-dimensional coefficient array $\mathbf{A}$ is computed by first defining the eigenvalue arrays and then performing the projection operations:

```
lvx = eigarray( eig(phi), alpha^2*eig(psi) );
A   = msolve( lvx, wip(Dx*Bp,Bvx) );
```

As the first step of determining a solution to the axial velocity component v_y, we compute the coefficients

c_j by

$$c_j = \langle 1, \xi_j \rangle_y = \int_{y=0}^{1} \xi_j \, dy.$$

and filter the result (using the second-order Fourier-space filter coefficients σ_j [1]) to reduce the Gibbs oscillations produced by the boundary condition discontinuity at the upper corners of the physical domain:

```
one   = sfield(S,1);
c     = wip( one('x=1'),BvyBC('x=1') );
c     = fsf(2,mBC)'.*c;
vyBC  = c*BvyBC;
```

The residual function for the axial velocity component is

$$\alpha \sum_{i,j=1}^{M_P} d_{i,j} \gamma_i \delta_j' = \sum_{i,j=1}^{M_V} b_{i,j} (\lambda_{\phi_i} + \alpha^2 \lambda_{\xi_j}) \phi_i \xi_j + \nabla^2 v_{y_{\partial\Omega}}$$

with $\nabla^2 v_{y_{\partial\Omega}} = \sum_{j=1}^{M_{BC}} c_j \left[2(\beta+1) + \alpha^2 x^2 \lambda_{\xi_j} \right] \xi_j$. Projecting this function onto $\phi_p \xi_q$ and solving for the $b_{p,q}$ gives

$$b_{p,q} = \frac{\alpha}{\lambda_{\phi_p} + \alpha^2 \lambda_{\xi_q}} \sum_{i,j=1}^{M_P} \left\langle \gamma_i \delta_j', \phi_p \xi_q \right\rangle_{x,y} d_{i,j} - \frac{c_q \left\langle \left[2(\beta+1) + \alpha^2 x^2 \lambda_{\xi_q} \right], \phi_p \right\rangle_x}{\lambda_{\phi_p} + \alpha^2 \lambda_{\xi_q}}$$

which translates to

```
lvy = eigarray( eig(phi), alpha^2*eig(xi) );
B = msolve( lvy, wip(Dy*Bp,Bvy) )/alpha;
C = - msolve( lvy, wip(DDx*vyBC + alpha^2*DDy*vyBC,Bvy) )/alpha;
```

Having represented the discretized velocity fields in terms of the still-unknown pressure field, we complete the computational procedure by substituting both velocity field expansions into the continuity equation and project the resulting residual onto the pressure-field trial functions to generate a set of linear equations in $\mathbf{d}$:

$$\mathbf{D}\mathbf{d} = -\mathbf{E}$$

which can be solved directly for $\mathbf{d}$. This procedure is described by the following computational steps, containing both the solution steps and the procedures for reconstructing the solution in the physical space:

```
D = mprod( wip( Dx*Bvx,Bp ),A,2,2,2 ) ...
      + alpha*mprod( wip( Dy*Bvy,Bp ),B,2,2,2 );

E = - alpha*mprod( wip( Dy*Bvy,Bp ),C,2,2,0 ) ...
      - alpha*mprod( wip( Dy*BvyBC,Bp ),c,2,2,0 );

d = msolve(D,E)

press = d*Bp
vx    = mprod(A,d)*Bvx;
vy    = (mprod(B,d)+C)*Bvy + c*BvyBC;
```

Representative results are presented in Fig. 6.

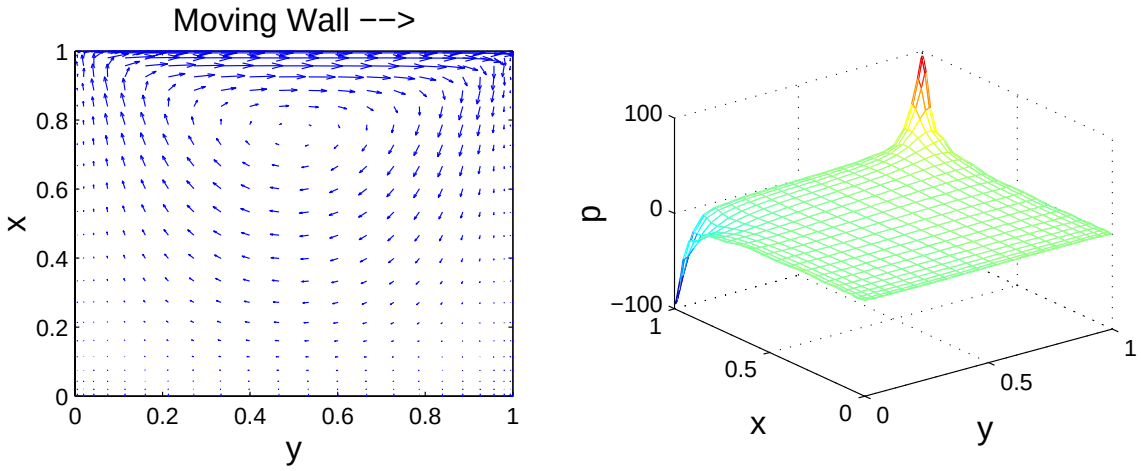

Figure 6: Driven cavity flow problem velocity field (left) and pressure field (right).

7 Heat transfer in CVD

Chemical vapor deposition (CVD) is a unit operation in semiconductor processing used for conformal deposition of thin films of electronic materials. Important considerations in CVD processes include the spatial uniformity with which films are deposited, and run-to-run consistency of the deposited films. These issues have motivated a number of model-based control, sensing, and process optimization studies to meet the uniformity challenges associated with increasing substrate (wafer) sizes and the continuous reduction of device length scales.

As one example of such processes, we consider modeling gas-phase heat transfer in a single-wafer tungsten CVD system. Model development for this system was motivated by the large discrepancy found between the single thermocouple available on the commercial CVD system and measurements taken with an instrumented wafer [8], prompting the development of a dynamic model relating the limited available process measurements to the temperature profile of the wafer during a processing cycle. In experimental studies performed with this system, inert gas of varying composition was flowed through the reactor at low pressure (0.5 Torr); a diagram of the reactor chamber is shown in Fig. 7. It has been shown through previous analysis [7] and verified by experiments [8] that the rate of heat transfer in this system under typical operating conditions is relatively insensitive to the details of the gas flow field. Therefore, the assumptions of fully developed and temperature-independent velocity field can be justified, resulting in the dimensionless modeling equation:

$$\frac{\partial^2 v_y}{\partial x^2} + \alpha_v \frac{\partial^2 v_y}{\partial z^2} = \frac{1}{\langle v_y \rangle} \tag{7}$$

subject to no-slip boundary conditions at $z = 0, 1$ and $x = 0, 1$. The definition of the dimensionless states and parameters found in this model are listed in Table 8.

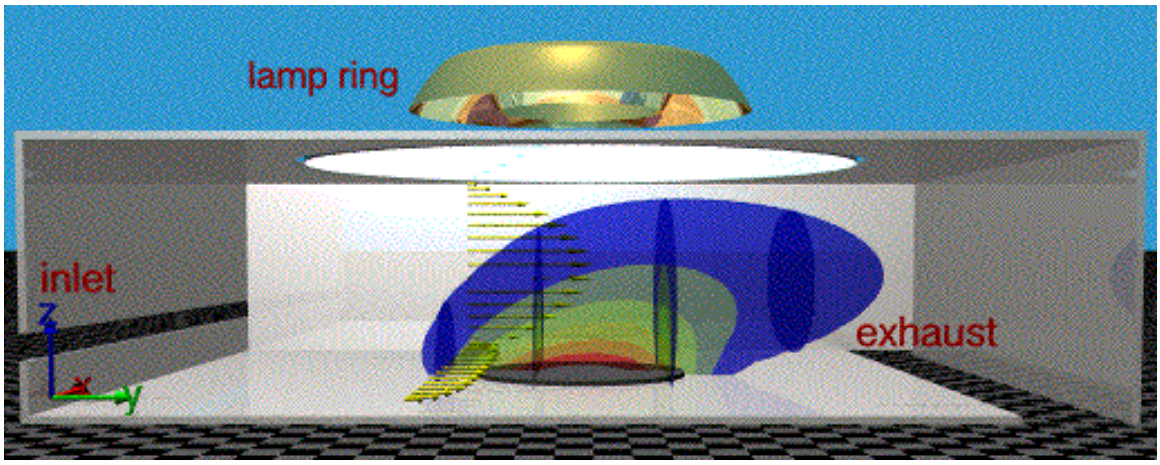

Figure 7: Three-dimensional CVD chamber geometry and gas velocity and temperature profiles (displayed in 43K temperature contour increments).

The solution procedure begins by defining the truncated trial function expansion used to represent the gas velocity field

$$v_y(x,z) = \sum_{i,j}^{I,J} d_{i,j}\phi_i(x)\eta_j(z) \tag{8}$$

where

$$\frac{d^2\phi}{dx^2} = \lambda^\phi \phi \qquad \phi(0) = \phi(1) = 0$$

$$\frac{d^2\eta}{dz^2} = \lambda^\eta \eta \qquad \eta(0) = \eta(1) = 0$$

The quadrature grid and basis functions defined above are computed using

```
XZ    = qgrid('slab',40,'x','slab',40,'z');
Bflow = truncate(bfun(XZ,'x',0,1,0,1),20) ...
          * truncate(bfun(XZ,'z',0,1,0,1),20)
```

Substituting (8) into (7) gives a simple eigenvalue problem that can be solved immediately by projecting the resulting residual function onto each basis function and solving for $\mathbf{d}$:

$$d_{i,j} = \frac{\langle 1, \phi_i(x)\eta_j(z)\rangle}{\lambda_i^\phi + \alpha_v \lambda_j^\eta}.$$

This solution is computed using the following computational steps

```
lam     = eigarray(eig(Bflow,1),av*eig(Bflow,2));
Bvfield = sfield(XZ,1);
d       = msolve(lam,wip(Bvfield,Bflow));
```

$X = 0.3$ m	reactor width
$Y = 0.4$ m	reactor length
$Z = 0.1$ m	reactor height
$\langle v_y^* \rangle = 0.23$ m/s	mean gas velocity for a feedrate of 250 sccm
$T_w = T_{ref} = 600$ K	wafer (reference) temperature
$R = 0.05$ m	wafer radius
$k = 0.0168$	gas thermal conductivity W/m/K

$$
\begin{aligned}
x &= x^*/X \\
y &= y^*/Y \\
z &= z^*/Z \\
v_y &= v_y^*/\langle v_y^* \rangle \\
T &= T^*/T_w \\
\alpha_v &= X^2/Z^2 = 9 \\
\kappa &= k/\rho C_p = 0.025 \text{ m/s}^2 \\
\beta_{gt} &= \kappa Y/X^2/\langle v_y^* \rangle = 0.48 \\
\gamma_{gt} &= \kappa 1/Y/\langle v_y^* \rangle = 0.27 \\
\delta_{gt} &= \kappa Y/Z^2/\langle v_y^* \rangle = 4.31
\end{aligned}
$$

Figure 8: Parameters and dimensionless variable definitions for the CVD reactor.

The (dimensional) solution is reconstructed in the physical space using the following commands and results are presented in Fig. 9:

```
Vmean = 0.23;
V     = d*Bflow;
V     = Vmean/wip(V)*V
```

7.1 Gas temperature field

With the gas flow field v_y in hand and under the assumption that gas properties do not depend on gas temperature, the chamber gas dimensionless temperature $T(x, y, z)$ equation and boundary conditions are written as

$$
\beta_{gt}\frac{\partial^2 T}{\partial x^2} + \gamma_{gt}\frac{\partial^2 T}{\partial y^2} + \delta_{gt}\frac{\partial^2 T}{\partial z^2} = v_y(x,z)\frac{\partial T}{\partial y} \tag{9}
$$

$$
T = 0 \quad \text{on} \quad \partial\Omega_1, \qquad T = T_w \quad \text{on} \quad \partial\Omega_2, \qquad \frac{\partial T}{\partial y} = 0 \quad \text{on} \quad \partial\Omega_3
$$

where

$$
\partial\Omega_2 : \qquad z = 0, \quad (x^* - X/2)^2 + (y^* - Y/2)^2 < R^2;
$$

$$
\partial\Omega_3 : \qquad y = 1, \quad 0 < x^* < X, \quad 0 < y^* < Y;
$$

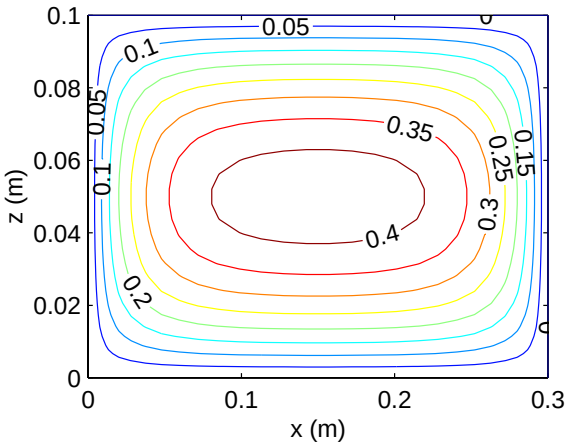

Figure 9: Velocity field contours in the $x-z$ plane (m/s) computed using an eigenfunction expansion method.

$$\partial\Omega_1: \qquad \text{remaining boundaries.}$$

Two physical domains must be defined for this system: the primary physical domain Q defining the physical region occupied by the reactant gas field and the circular region S ($\partial\Omega_2$) defined by the heated wafer:

```
Q = qgrid('slab',40,'x','slab',40,'y','slab',40,'z');
S = qgrid('cyln',30,'r','peri',31,'s');
```

Furthermore, we must specify the relationship between the two domains: that the wafer region S has radius R and is located at $z = 0$, centered at $x = 0.5$, $y = 0.5$ (in dimensionless coordinates). We store this information in a QGRIDC object:

```
S = qgridc(S,[0.5 0.5],[R/X R/Y],Q,'z=0');
```

Differentiation operations are now defined for this system:

```
Dy  = loper(Q,'d' ,'y');
Dz  = loper(Q,'d' ,'z');
DDx = loper(Q,'dd','x');
DDy = loper(Q,'dd','y');
DDz = loper(Q,'dd','z');
```

We express the solution for T in terms of the truncated global basis function expansions

$$\begin{aligned} T(x,y,z) &= \sum_{i,j,k=1}^{I,J,K} a_{i,j,k}\phi_i(x)\psi_j(y)\eta_k(z) + \sum_{i=1}^{I,J} b_{i,j}\phi_i(x)\psi_j(y)(1-z)^2 \\ &= T_{hm} + T_{bc} \end{aligned} \tag{10}$$

where the $\phi_i(x)$ and $\eta_k(z)$ are the same as the trial function sequences used to compute the gas flow velocity

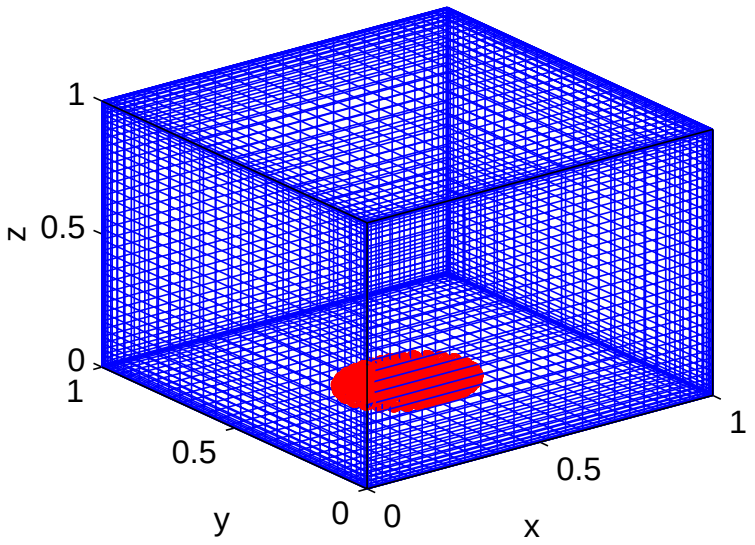

Figure 10: The computational domain; wafer domain appears distorted due to $Y/X \neq 1$ aspect ratio in the physical domain.

field and

$$\frac{d^2\psi}{dx^2} = \lambda^{\psi}\psi \qquad \psi(0) = \frac{d\psi(1)}{dy} = 0.$$

We refer to T_{hm} as the homogeneous contribution to the trial function expansion and T_{bc} as the nonhomogeneous contribution. The basis function objects are set up as follows:

```
L = 7; M = 18; N = 18; % truncation numbers

Phi = even(bfun(Q,'x',0,1,0,1));
Psi = bfun(Q,'y',0,1,1,0);
Eta = bfun(Q,'z',0,1,0,1);

z = get(Q,'qp','z');
F = bfun(Q,(1-z).^2,[],'z');

Phm = Phi * Psi * Eta; % Homogeneous basis functions
Pbc = Phi * Psi * F;   % Inhomogeneous at z = 0;

Phm = truncate(Phm,[L M N]);
Pbc = truncate(Pbc,[L M 1]);
```

The nonhomogeneous contribution to the temperature field basis function expansion is computed by

$$b_{i,j} = \langle T_w\left(r(x,y),\theta(x,y)\right),\phi_i(x)\psi_j(y)\rangle \qquad i = 1,\ldots,I, \quad j = 1,\ldots,J$$

which translates to

```
Tw = sfield(S,1);
b  = wip(Tw,Pbc);
```

in our computational framework. Note how the projection of the wafer temperature defined on physical domain S in polar coordinates is automatically projected onto the $z = 0$ boundary of QGRID object Q in this procedure; this is made possible by the information stored in QGRIDC object S specifying the relationship between the two physical domains. The nonhomogeneous contribution to the temperature field (10) is reconstructed in the three-dimensional physical space using

```
Tbc = b*Pbc;
```

The solution procedure then reduces to minimizing the residual function formed by substituting (10) into (9)

$$\beta_{gt}\frac{\partial^2(T_{hm}+T_{bc})}{\partial x^2}+\gamma_{gt}\frac{\partial^2(T_{hm}+T_{bc})}{\partial y^2}+\delta_{gt}\frac{\partial^2(T_{hm}+T_{bc})}{\partial z^2}-v_y(x,z)\frac{\partial(T_{hm}+T_{bc})}{\partial y}=0.$$

Because the basis functions of T_{hm} are defined as eigenfunctions of the Laplacian operator, projecting the residual function above onto the basis functions of T_{hm} gives

$$\left(\beta_{gt}\lambda_p^{\phi}+\gamma_{gt}\lambda_q^{\psi}+\delta_{gt}\eta_r^{\eta}\right)a_{p,q,r}-\left\langle v_y(x,z)\frac{\partial T_{hm}}{\partial y},\phi_p(x)\psi_q(y)\eta_r(z)\right\rangle$$
$$=\left\langle-\beta_{gt}\frac{\partial^2 T_{bc}}{\partial x^2}-\gamma_{gt}\frac{\partial^2 T_{bc}}{\partial y^2}-\delta_{gt}\frac{\partial^2 T_{bc}}{\partial z^2}+v_y(x,z)\frac{\partial(T_{hm}+T_{bc})}{\partial y},\phi_p(x)\psi_q(y)\eta_r(z)\right\rangle$$

for $p = 1,\ldots,I$, $q = 1,\ldots,J$, $r = 1,\ldots,K$. The computational procedure follow an analogous path of forming a residual function from the nonhomogeneous contribution to the solution, projecting it on the basis functions of the homogeneous contribution, and solving the linear system:

```
R     = bgt*DDx*Tbc + ggt*DDy*Tbc + dgt*DDz*Tbc - V.*(Dy*Tbc);
Rproj = wip(R,Phm);
Jproj = eigarray( bgt*eig(Phm,1),ggt*eig(Phm,2),dgt*eig(Phm,3) ) - wip(V.*(Dy*Phm),Phm);
a     = - msolve(Jproj,Rproj);
```

7.2 Solution analysis

The gas temperature field is reconstructed in the physical space using

```
T = a*Phm + b*Pbc;
```

A representative solution is shown in terms of constant-x and y gas temperature contours in Fig. 7; the gas temperature SFIELD object for $x = 0.5$ is created from the SFIELD object T simply by evaluating

```
T('x=0.5')
```

Nonuniform gas/wafer heat transfer is one mechanism leading to nonuniform wafer temperature and potentially nonuniform deposition profiles. We compute the heat transfer rate between the wafer/floor and gas

phases by

$$q_{flux} = \frac{k}{Z}\frac{\partial T}{\partial z}\bigg|_{z=0}$$

Results are displayed in Fig. 11 showing the increased heat transfer rate in the region the wafer leading edge; we note this plot correctly reflects that energy is transferred from the heated gas phase to the chamber floor downstream of the wafer and near the wafer edge (this is represented by negative flux values). We note that this plot corresponds to a total gas flow rate of 1000 sccm, which is four times the reactant gas flow normally used in experiments conducted with this system; the heat transfer rate is significantly more uniform for a feed gas total flow rate of 250 sccm, a result consistent with experimental observations.

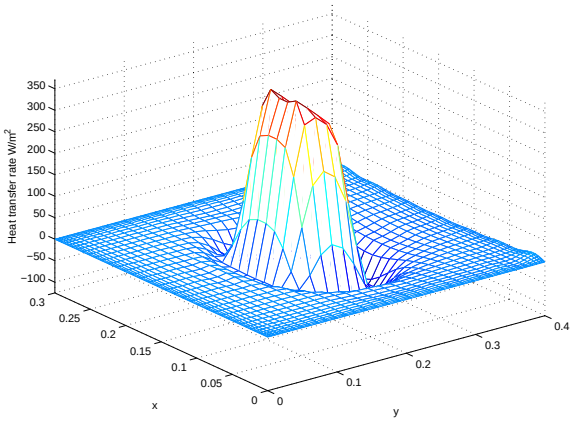

Figure 11: Heat transfer from the wafer/floor to the gas phase.

8 Concluding remarks

An object-oriented approach to implementing global spectral projection methods in the MATLAB computing environment was developed and applied in solving several two- and three-dimensional BVP-based modeling problems. It was shown that the computational implementation of these MWR in this framework can be carried out using very compact MATLAB scripts; however, because the emphasis in developing this numerical approach was based on the idea of creating computational techniques that have a direct correspondence to each step in an MWR procedure, use of this library requires some knowledge of implementing the MWR. While this flexibility may be important in research applications, such as developing model reduction methods for distributed parameter systems, currently we are studying whether a graphical user interface would simplify many of the implementation steps.

Additional research is underway in defining object classes that facilitate solving problems consisting a set of BVPs in multiple, connected physical domains, such as those encountered in spectral element method

applications. Likewise, recent research on inexact Newton methods [12] promises to eliminate the difficult, and computationally and memory intensive step of explicitly computing Jacobian arrays.
Additional documentation, sample scripts, and the library of functions can be found at the project website `http://www.ench.umd.edu/software/MWRtools`.

9 Acknowledgment

The author acknowledges the support of the National Science Foundation through grants ECS-0082381 and CTS-0085633.

References

[1] Adomaitis, R. A. 'Spectral filtering for improved performance of collocation discretization methods,' *Computers & Chem. Enging*, in press (2001).

[2] Adomaitis, R. A. and Y.-h. Lin, 'A technique for accurate collocation residual calculations,' *Chem. Engng J.*, **71** 127-134 (1998).

[3] Adomaitis, R. A., Y.-h. Lin, and H.-Y. Chang, 'A computational framework for boundary-value problem based simulations,' *Simulation* **74** 30-40 (2000).

[4] Ascher, U., J. Christiansen, and R. D. Russell, 'Collocation software for boundary-value ODEs,' *ACM Trans. Math. Software* **7** 209-222 (1981).

[5] Ascher, U., R. M. Mattheij, and R. D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* Prentice Hall, Englewood Cliffs, NJ (1988).

[6] Burggraf, O. R., 'Analytical and numerical studies of the structure of steady separated flows', *J. Fluid Mech.* **24** 113-151 (1966).

[7] Chang, H. -Y. and R. A. Adomaitis, 'Analysis of heat transfer in a chemical vapor deposition reactor: An eigenfunction expansion solution approach,' *Int. J. Heat Fluid Flow*, **20** 74-83 (1999).

[8] Chang, H. -Y., R. A. Adomaitis, J. N. Kidder, Jr., and G. W. Rubloff, 'Influence of gas composition on wafer temperature in a tungsten chemical vapor deposition reactor: Experimental measurements, model development, and parameter estimation,' *J. Vac. Sci. Tech., B*, **19** 230-238 (2001).

[9] Choo, J. and R. A. Adomaitis, 'Assessment of computational methods for implementing weighted residual methods based on very high degree polynomial approximations,' manuscript in preparation.

[10] Cooper, J. M. *Introduction to Partial Differential Equations with* MATLAB, Birkhauser, Boston, MA (1998).

[11] Don, W. S. and A. Solomonoff *PseudoPack - Pseudo-Spectral Differentiation Software Package User Manual for version 2.3 beta* Brown University, Providence, RI (1997).

[12] Fokkema, D. R. G. L. G. Sleijpen, and H. A. Van der Vorst, 'Accelerated inexact Newton schemes for large systems of nonlinear equations,' *SIAM J. Sci. Comput.*, **19** 657-674 (1998).

[13] Gottlieb, D. and C. -W. Shu, 'On the Gibbs phenomenon and its resolution,' *SIAM Rev.* **39** 644-668 (1997).

[14] Madsen, N. K. and R. F. Sincovec 'Algorithm 540: PDECOL, general collocation software for partial differential equations,' *ACM Trans. Math. Software*, **5** 326-351 (1979).

[15] Joseph, D. D. and L. Sturges, 'The convergence of biorthogonal series for biharmonic and Stokes flow edge problems: part II,' *SIAM J. Appl. Math.* **34** 7-26 (1978).

[16] Langtangen, H. P. *Computational Partial Differential Equations: Numerical Methods and Diffpack Programming*, Lecture Notes in Computational Science and Engineering, Vol. 2, Springer-Verlag (1999).

[17] Langtangen, H. P. and O. Munthe, 'Solving systems of partial differential equations using object-oriented programming techniques with coupled heat and fluid flow as example,' *ACM Trans. Math. Software*, **27** 1-26 (2001).

[18] Lew, H. S. and Y. C. Fung, 'The motion of the plasma between the red blood cells in bolus flow,' *Biorheology* **6** 109-119 (1969).

[19] Lin, Y.-h. and R. A. Adomaitis, 'Simulation and model reduction methods for an RF plasma glow discharge,' *J. Comp. Phys.*, **171** 731-752 (2001).

[20] Lin, Y. -h., Chang, H. -Y., and R. A. Adomaitis, 'MWRtools: A library for weighted residual methods,' *Computers & Chem. Engng* **23** 1041-1061 (1999).

[21] Pan, F. and A. Acrivos, 'Steady flows in rectangular cavities,' *J. Fluid Mech.* **28** 643-655 (1967).

[22] Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C* Cambridge University Press, New York (1988).

[23] Rice, R. G. and D. D. Do, *Applied Mathematics and Modeling for Chemical Engineers*, J. Wiley, New York (1995).

[24] Shampine, L. F., J. Kierzenka, and M. W. Reichelt, 'Solving boundary value problems for ordinary differential equations in MATLAB with `bvp4c`,' preprint (2000).

[25] Shankar, P. N., 'The eddy structure in Stokes flow in a cavity,' *J. Fluid Mech.* **250** 371-383 (1993).

[26] Srinivasan, R., 'Accurate solutions for steady plane flow in the driven cavity. I. Stokes flow,' *ZAMP* **46** 524-545 (1995).

[27] Trefethen, L., *Spectral Method in MATLAB*, SIAM, (2000).

[28] Villadsen, J. and M. L. Michelsen, *Solution of Differential Equation Models by Polynomial Approximation*, Int. Series in Phys. and Chem. Engng. Sci., Prentice-Hall, Englewood Cliffs, NJ (1978).

[29] Villadsen, J. V. and W. E. Stewart, 'Solution of boundary-value problems by orthogonal collocation,' *Chem. Engng. Sci.* **22** 1483-1501 (1967).

[30] Weideman, J. A. C and S. C. Reddy, 'A MATLAB differentiation matrix suite,' *ACM Trans. Math. Software*, **26** 465-519 (2000).