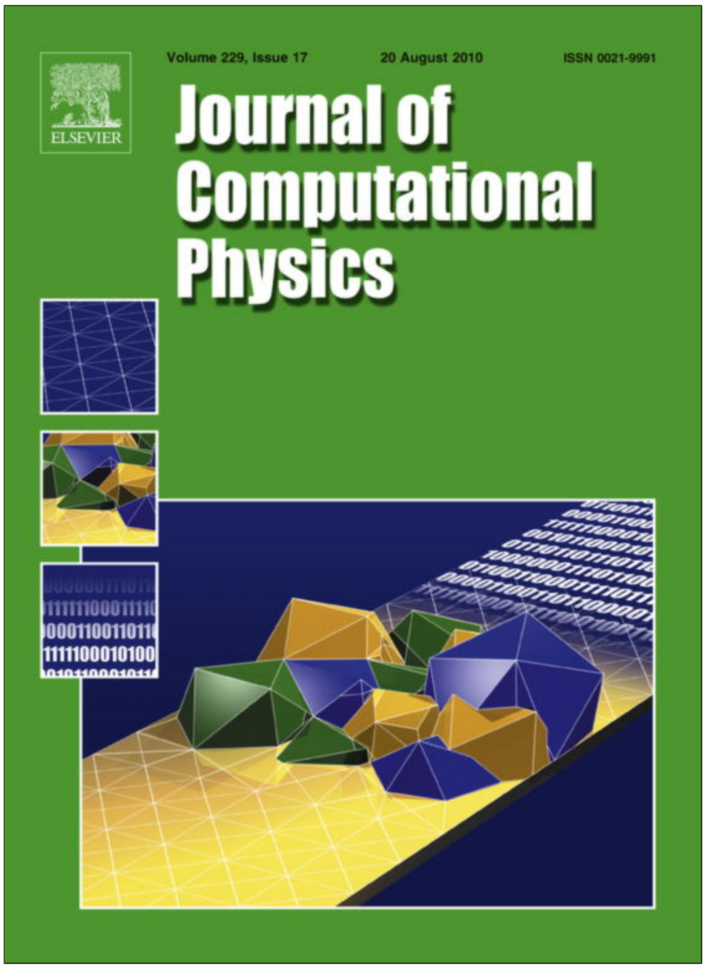
Volume 229, Issue 17
20 August 2010
ISSN 0021-9991
ELSEVIER
Journal of Computational Physics

Journal of Computational Physics 229 (2010) 5980–5998

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Automatic spectral collocation for integral, integro-differential, and integrally reformulated differential equations ☆

Tobin A. Driscoll *

Department of Mathematical Sciences, University of Delaware, Newark, DE 19716, USA

ARTICLE INFO

Article history:
Received 14 July 2009
Received in revised form 14 April 2010
Accepted 15 April 2010
Available online 28 April 2010

Keywords:
Chebfun
Spectral collocation
Integral equation
Integro-differential equation
Iterative method

ABSTRACT

Automatic Chebyshev spectral collocation methods for Fredholm and Volterra integral and integro-differential equations have been implemented as part of the chebfun software system. This system enables a symbolic syntax to be applied to numerical objects in order to pose and solve problems without explicit references to discretization. The same objects can be used in matrix-free iterative methods in linear algebra, in order to avoid very large dense matrices or allow application to problems with nonsmooth coefficients. As a further application of the ability to implement operator equations, a method of Greengard [1] for the recasting of differential equations as integral equations is generalized to mth order boundary value and generalized eigenvalue problems. In the integral form, large condition numbers associated with differentiation matrices in high-order problems are avoided. The ability to implement the recasting process generally follows from implementation of the operator expressions in chebfun. The integral method also can be extended to first-order systems, although chebfun syntax does not currently allow easy implementation in this case.

© 2010 Elsevier Inc. All rights reserved.

1. Introduction

In scientific computing one often implicitly encounters a basic dichotomy: for analytical problem expression and manipulation, functions and operators are the primary building blocks, while for numerical computations, scalar numbers, or perhaps vectors and matrices in a high-level language, are most prominent. Mapping between these dual representations is so commonplace that it may seem inconsequential, particularly for simple correspondences such as the one between differentiation and second-order finite differences. Even with such easily understood connections, however, the complications introduced by variable coefficients, high-order derivatives, nonlinearity, eigenvalue problems, boundary conditions, etc. can quickly cloud the picture and complicate the translation process. The situation is exacerbated by more sophisticated discretizations such as spectral methods, creating an increasing sense of distance between the way we prefer to express problems and the ways in which they are solved computationally.

The chebfun software system [2–4] exploits mathematical results from approximation theory, fast algorithms from spectral methods, and object-oriented software design to greatly reduce or eliminate the distance between analytical expressions and numerical solutions for one-dimensional problems. Chebfun, which is a free MATLAB add-on package, makes functions and operators the fundamental objects that are created and manipulated. Algorithmically, chebfun uses automatic Chebyshev polynomial interpolation to represent functions and automatic spectral collocation methods to approximate operators.

☆ This work was supported by UK EPSRC Grant EP/E045847.
* Tel.: +1 302 831 3383; fax: +1 302 831 4511.
E-mail address: driscoll@math.udel.edu

0021-9991/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2010.04.029

The Chebyshev representations are typically refined until accurate at a level comparable to machine precision (or a larger user-selected value), and they can be manipulated very quickly for processing steps such as integration, rootfinding, and global optimization. An object-oriented interface is used to give the illusion of symbolic manipulation, even though the underlying objects themselves are numerical. Much of the familiar MATLAB syntax for vectors and matrices extends naturally to functions and operators; in particular, the commands \, `eigs`, and `expm` solve boundary-value problems, find eigenfunctions, and create operator exponentials (C_0-semigroups), respectively [3].

Section 2 introduces some aspects of chebfun in more detail. The section focuses on the practical construction of Fredholm and Volterra integral operators with continuous kernels, showing how previously known algorithms from spectral methods can become permanently encapsulated within the chebfun system for simple re-use. These are applied to two integro-differential equations, a model of neuronal transmission [5] and a model of traveling dispersive corner waves [6].

The remainder of the paper is devoted to the transformation of differential operators into equivalent integral operators, and realizing those operators as code in the chebfun system for general-purpose use. It is well known that spectral discretizations of high-order differentiation typically lead to poorly conditioned matrices [7], with a possible serious or even catastrophic resulting loss of accuracy in the computed solution. Greengard [1] proposed rewriting a second-order, constant-coefficient differential equation as an integral equation for the second derivative of the solution. Coutsias and coauthors modified Greengard's techniques and extended them to more general problems [8,9], proving that condition numbers remain bounded as the discretization is refined (i.e., mesh independence). Mai–Duy [10,11] apparently rediscovered the basic Greengard technique and applied it also to some eigenvalue and, using tensor products, multidimensional problems.

In Section 3 the Greengard approach is extended and generalized to mth order nonsingular boundary value and eigenvalue problems. The translation of the mathematical operator formulations into chebfun codes provides a case study in how chebfun offers an alternative to traditional numerical discretization and implementation. The ideas are extended further in Section 4, where direct, finite-dimensional numerical linear algebra is replaced by GMRES for boundary-value problems and inverse iteration for eigensystems. Because the chebfun operator objects maintain both functional and discretized forms, switching to matrix-free iteration techniques is a coding triviality. One benefit of the iterative methods is that they only apply integration and multiplication operators, so they work seamlessly with piecewise function representations to provide spectrally accurate solutions to problems with nonsmooth or discontinuous data. This fact is demonstrated by examples on nonlinear boundary value problems and eigensystems in Section 4. Finally, Section 5 mathematically describes the process of extending integral formulations to systems of equations; implementation is left for future work.

2. Fredholm and Volterra integral operators

Most work in the chebfun system[1] consists of creating and manipulating functions and operators. Within the system these are represented as objects of class *chebfun* and either *chebop* ("cheb-op") or *linop*, respectively, depending on whether the operator is nonlinear or linear.[2] In basic usage, a chebfun object is created by sampling the function at increasing numbers of points in an interval, converting the sample values to the coefficients of an interpolating Chebyshev series, and inspecting the relative size of the coefficients in order to determine whether convergence has been achieved to double precision. For instance, the function $f(x) = \sin(e^x)$ on [0,4] requires 108 sample values, or a global polynomial of degree 107:

```
 ≫ f = chebfun( @(x) sin(exp(x)), [0,4] )
f =
  chebfun column (1 smooth piece)
    interval     length   endpoint values
(     0,        4)     108     0.84      − 0.93
vertical scale = 1
```

The sample points are not equally spaced, but instead are Chebyshev (second-kind or Gauss–Lobotto) points scaled to an interval $[a, b]$:

$$x_j = \frac{a+b}{2} + \frac{a-b}{2}\cos\left(\frac{(j-1)\pi}{n-1}\right), \quad j = 1, \dots, n. \tag{1}$$

The construction above is typical for a user-defined function with an explicitly known formula. However, functions that are defined only implicitly, such as solutions of differential and integral equations, are "sampled" by using Chebyshev spectral collocation (also known as pseudospectral) methods [12,13]. Consequently, the representation of a linear operator maintains two parallel descriptions of that operator: an operational form, which applies a functional expression to a chebfun, and a matrix form, which can be realized at any finite value of n with the understanding that it applies to vectors of samples at the Chebyshev nodes. In the case of the indefinite integration operator `cumsum`, for example,

[1] Chebfun is available for free download at http://www.maths.ox.ac.uk/chebfun/. The version of chebfun used throughout this work is 3.1000.

[2] The linop class is a subclass of the chebop class, so while it is technically correct to refer to a linop object as a chebop, the converse is not true.

```
 ≫ [d, x] = domain(0, 1);
 ≫ C = cumsum(d);
 ≫ C(inf)

ans = oparray
@(u)cumsum(u)

 ≫ C(4)

ans =
   -0.0000   0.0000   0.0000    -0.0000
   0.0868    0.1944   -0.0556   0.0243
   0.0313    0.5000   0.2500    -0.0312
   0.0556    0.4444   0.4444    0.0556
```

Denote the entries of the $n \times n$ instantiation of `C` by $c_{ij}^{(n)}$. Then

$$\sum_{j=1}^{n} c_{ij}^{(n)} f(x_j) \approx \int_a^{x_i} f(t)dt. \tag{2}$$

Because a global interpolant is constructed for f, we do not get a triangularity condition $c_{ij}^{(n)} = 0$ for $j > i$, as one would expect for piecewise linear interpolation, for example. Note in particular that the entries in the nth row,

$$w_j^{(n)} = c_{nj}^{(n)}, \quad j = 1, \ldots, n, \tag{3}$$

are the Clenshaw–Curtis quadrature weights – i.e., the weights that give exact integration over $[a,b]$ of the Chebyshev polynomial interpolant of the data. The operational form stored as `C(inf)`, when applied to a chebfun u, operates somewhat differently, integrating the internal Chebyshev polynomial expansion analytically. Up to roundoff, the two representations lead to the same result when n is as large as the degree of the underlying chebfun polynomial representation of u.

Most linear operators arising in practice can be constructed from the building blocks in Table 1, together with standard algebraic operations and a separate syntax to specify boundary conditions for differential operators [3]. However, in the codes that follow we sometimes construct a linop object directly by specifying the operational and matrix parts explicitly. In order to set the ideas clearly, we briefly describe how this is done for the `fred` and `volt` commands that build Fredholm and Volterra integral operators, i.e. the operators F and V defined by

$$(Fu)(x) = \int_a^b k(x,y)u(y)dy, \tag{4}$$

$$(Vu)(x) = \int_a^x k(x,y)u(y)dy \tag{5}$$

for $x \in [a,b]$. We shall assume that the kernel function $k(x,y)$ is continuous on $[a,b]^2$. Although in many applications integral operators have kernels that are singular in the domain, these are not implemented within the present chebfun system.

We begin with the Fredholm operator (4). Given a chebfun u, its operational form should construct a new chebfun by sampling the values of the integral (4) at automatically selected values of x. For each new x, we create a chebfun representation of $k(x,\cdot)$, then integrate the result after multiplication by u. The result is a nested chebfun construction:

```
g = @(x) chebfun(@(y) k(x,y), d);            % inner construction for k (.,y)
v = chebfun(@(x) sum(g(x).*u), d);           % outer construction for result
```

Table 1
Basic operator building blocks in the chebfun system.

Construction	Meaning
`eye(d)`	Identity operator for functions on domain `d`
`zeros(d)`	Zero operator for functions on domain `d`
`diff(d)`	Differentiation operator for functions on domain `d`
`cumsum(d)`	Indefinite integration operator for functions on domain `d`
`diag(f)`	Pointwise multiplication by chebfun `f`
`fred, volt`	Fredholm and Volterra integral operators (see text)
`[A B; C D]`	Horizontal and vertical concatenation

The matrix form of F should map a vector $\mathbf{u}$, whose entries are values of u at the n Chebyshev nodes (1), to another vector $\mathbf{v} = F\mathbf{u}$ that contains values of the integral (4) at the same nodes. We use a discrete collocation model [14] with Clenshaw–Curtis quadrature to get

$$v_i = \sum_{j=1}^{n} w_j^{(n)} k(x_i, x_j) u_j, \quad i = 1, \dots, n,$$

where the $w_j^{(n)}$ are as in (3). In matrix form we have

$$\mathbf{v} = \begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & \cdots & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & \cdots & k(x_2,x_n) \\ & \vdots & & \\ k(x_n,x_1) & k(x_n,x_2) & \cdots & k(x_n,x_n) \end{bmatrix} \begin{bmatrix} w_1^{(n)} & & & \\ & w_2^{(n)} & & \\ & & \ddots & \\ & & & w_n^{(n)} \end{bmatrix} \mathbf{u}. \tag{6}$$

This equation, and the code above for the operational form of F, can be used to create a generic Fredholm operator constructor, as shown in Fig. 1.

The construction of a Volterra chebop for V in (5) is similar and is shown in Fig. 2. The function that implements the operational form of V changes because of the triangular domain for k; it creates a chebfun on demand that represents the entire integrand $k(x,y)u(y)$ for fixed x. The Volterra analog of the n-point equation (6) is

$$\mathbf{v} = \begin{bmatrix} \gamma_{11}k(x_1,x_1) & \gamma_{12}k(x_1,x_2) & \cdots & \gamma_{1n}k(x_1,x_n) \\ \gamma_{21}k(x_2,x_1) & \gamma_{22}k(x_2,x_2) & \cdots & \gamma_{2n}k(x_2,x_n) \\ & \vdots & & \\ \gamma_{n1}k(x_n,x_1) & \gamma_{n2}k(x_n,x_2) & \cdots & \gamma_{nn}k(x_n,x_n) \end{bmatrix} \mathbf{u}, \tag{7}$$

where the γ_{ij} for fixed i are quadrature weights for integration over the interval $[a,x_i]$ of a Chebyshev interpolant defined on the entire interval $[a,b]$. In other words, $\gamma_{ij} = c_{ij}^{(n)}$, the entries of the n-point realization of the `cumsum` operator! In MATLAB syntax, the matrix multiplying $\mathbf{u}$ in (7) is found using the elementwise multiplication operator `.*`, as shown in the code.

Both Figs. 1 and 2 give compact codes that work well in most cases but are not optimal. The versions found in the chebfun distribution are much faster at creating the Clenshaw–Curtis weights, protect against cancellation errors in the kernel, and include an alternate syntax that can allow the faster creation and application of the operators for some kernels. One use of this alternate syntax is given in Example 2.

2.1. Numerical experiments

With the functions `fred` and `volt`, we can easily solve second-kind integral equations and more complex problems involving such operators.

```
function F = fred(k,d)
% Chebop for Fredholm integral operator with kernel k over domain d.
C = cumsum(d);
w = C(end,:);   % Clenshaw-Curtis weights for any n
F = linop(@matrix,@op,d);

  function v = op(u)       % operational definition
    v = chebfun( @(x) sum( u.*chebfun(@(y) k(x,y),d) ), d );
  end

  function A = matrix(n)   % finite-dimensional definition
    [X,Y] = ndgrid( chebpts(n,d) );
    A = k(X,Y) * spdiags(w(n).',0,n,n);
  end

end
```

Fig. 1. Skeleton code for construction of a Fredholm integral operator with continuous kernel k.

```
function V = volt(k,d)
% Chebop for Volterra integral operator with kernel k over domain d.
C = cumsum(d);
V = linop(@matrix,@op,d);

  function v = op(u)       % operational definition
    g = @(x) chebfun(@(y) u(y).*k(x,y),[d.ends(1) x]);
    v = chebfun( @(x) sum(g(x)), d );
  end

  function A = matrix(n)   % finite-dimensional definition
    [X,Y] = ndgrid( chebpts(n,d) );
    A = k(X,Y) .* C(n);
  end

end
```

Fig. 2. Skeleton code for construction of a Volterra integral operator with continuous kernel k.

Example 1. The integro-differential equation [15]

$$\begin{aligned} &y'(x) + y(x) - x(1+2x)\int_0^x e^{t(x-t)} y(t)dt = 1+2x, \quad x \in [0,1], \\ &y(0) = 1, \end{aligned} \tag{8}$$

has the exact solution $y = e^{x^2}$. We can combine a Volterra operator with a differentiation operator, add a boundary condition, and solve the equation, as shown here.

```
≫[d,x] = domain (0,1);
≫V = volt ( @(x,t) exp (t.*(x-t)), d );                    % Volterra operator
≫A = diff (d) + 1 - diag (x.*(1 + 2 * x))*V;               % int-diff operator
≫A.lbc = 1;                                                % set y (0)=1
≫y = A\(1 + 2 * x);                                        % solve for y
```

The \ solution operator applies spectral collocation on 9, 17, 33,... nodes until the resulting solution is fully resolved, as measured by the Chebyshev series coefficients. In this case, the solution is successful nearly to machine precision.

Example 2. The following integro-differential equation has been proposed as a model for a neuronal transmission line [5]:

$$\frac{\partial}{\partial t}\theta(x,t) = f(\theta,t) + \mu\int_{-\infty}^{\infty} K(x-y)\frac{\partial}{\partial t}\theta(y,t)dy, \tag{9}$$

where $\theta(x,t)$ represents a signal phase on the problem domain $-\infty < x < \infty$, $t > 0$, the excitatory kernel K is given by, e.g.,

$$K(s) = \frac{1}{\sigma\sqrt{\pi}} e^{-s^2/\sigma^2}, \tag{10}$$

and the driving function f is given by

$$f(\theta,t) = 2 - H(t-10) + \cos\theta,$$

with H as the unit step function. After $t = 10$, the solution everywhere tends to relax towards values of θ such that $\cos\theta = -1$, with increasingly steplike transitions between constant-valued domains.

Note that (9) has the form $(I - \mu F)\theta_t = f(\theta,t)$ for a Fredholm operator F. By formally inverting the operator $I - \mu F$ we can solve for θ_t in terms of θ at any t and thus can choose any standard time integrator; for the computations below, we use fourth-order Runge–Kutta.

The phase θ does not asymptote to a fixed value as $x \to \pm\infty$, which makes truncation of the domain an interesting issue. Motivated by the decaying kernel (10), for an accurate solution on the truncated domain $[-L,L]$ we simply solve on the extended domain $[L - 8\sigma, L + 8\sigma]$ by truncating the integral in the Fredholm operator accordingly. While the solution is not accurate near the boundaries of the extended domain, the effects of these errors are vanishingly small in the domain of interest.

An implementation of the numerical solution of this problem is given in Appendix A and is available from the author. It is straightforward in most regards, with one exception. The tendency of the solution to relax towards a stairstep function means that the number of discretization points in x needs to be in the thousands, to avoid Gibbs oscillations. Using dense linear algebra to invert the operator $I - \mu F$ would be prohibitively slow. However, the decaying kernel (10) means that the matrix representation of F is naturally sparse, without introducing approximation. This representation is not available in the default `fred` syntax, but an alternate syntax can be used to allow the code to set up the kernel in sparse form. The alternate syntax can also be useful when the kernel is separable, i. e., $K(x,y) = k_1(x)k_2(y)$, since a vector outer product can then be used to evaluate K on the underlying tensor product quadrature grid.

Fig. 3 shows some computational results using $\mu = 0.2$, $\sigma = 1$, $L = 100$, and a time step of 0.2. The results shown took about 124 s to compute.[3] For $t \leqslant 10$, the phase changes in roughly periodic fashion, and the polynomial degree chosen to represent the solution never exceeds 250. For $t > 10$ the phase relaxes toward a stairstep shape. The large gradients that develop cause the chebfun constructor to require a relatively large number of points in the discretization, making the use of a sparse Fredholm operator critically important. Note also that these steep transitions are not uniformly spaced. In fact, they are highly sensitive to the details of the computation; relaxing the chebfun accuracy parameter `eps` to 10^{-8} qualitatively changes their locations, whereas making the parameter smaller than the 10^{-10} chosen for the figure has no discernible effect.

Example 3. The equation

$$(u(x) - c)u''(x) + (u'(x))^2 - (Su)(x) = 0, \quad 0 < x < \pi, \tag{11}$$

describes a traveling corner wave for families of nonlinear dispersive wave equations [6]. The linear operator S determines the nature of the dispersion; for example, the choice

$$(Su)(x) = \int_0^\pi [\cos(x-y) + \cos(x+y)]u(y)dy \tag{12}$$

leads to a model studied by Gabov [16] and Shefter and Rosales [17] and is the one considered here. The construction of a spectral collocation approximation to S is done via

```
d = domain (0,pi);
F = fred (@(x,y) cos (x-y)+cos (x + y),d);
```

The phase speed c in (11) is considered an unknown, and thus there are three boundary conditions as well:

$$u(0) = c, \tag{13a}$$
$$(u'(0))^2 - (Su)(0) = 0, \tag{13b}$$
$$u'(\pi) = 0. \tag{13c}$$

In addition, the fact that the kernel of the Fredholm operator (12) has zero mean implies that the solution is defined only up to a uniform translation; to make the solution unique, we also impose the zero-mean condition

$$\int_0^\pi u(x)dx = 0. \tag{13d}$$

Solving for the function $u(x)$ and scalar c simultaneously leads to a parting of ways between traditionally discretized methods and the function/operator point of view. In a classical numerical approach, the function u is discretized into n degrees of freedom, and c simply becomes an additional discrete unknown. Statements about the derivative of u at a boundary, for example, must be restated as linear combinations of elements of a discretized u. If however we want to view u as a function and state nothing explicitly about its discretization, then we need to acknowledge that u and c are fundamentally different classes of objects. We shall proceed by eliminating c to produce

$$G(u) := (u - E_0u)u'' + (u')^2 - Su = 0, \tag{14}$$

for $x \in [0,\pi]$, where E_0 is a left-evaluation functional; i.e., $E_0u = u(0)$. We find it convenient to view E_0 as a linear operator that produces a constant-valued function, in which case its implementation is

```
mat = @(n) ones (n,1)*[1 zeros (1,n-1)];          % outer product matrix
op = @(u) chebfun (u (0),d);                       % constant-valued function
E0 = linop (mat,op,d);
```

The integro-differential equation (14) is subject to the two boundary conditions (13b) and (13c) plus the zero mean condition (13d). In order to solve this nonlinear problem, we can apply Newton's iteration. If u represents a guess to the solution, we can linearize (14) and (13b) about u to find that the Newton correction $\bar{u}$ satisfies

[3] All running times in this paper are wall-clock times for a dual-core 2.66 GHz PC running MATLAB under Windows XP.

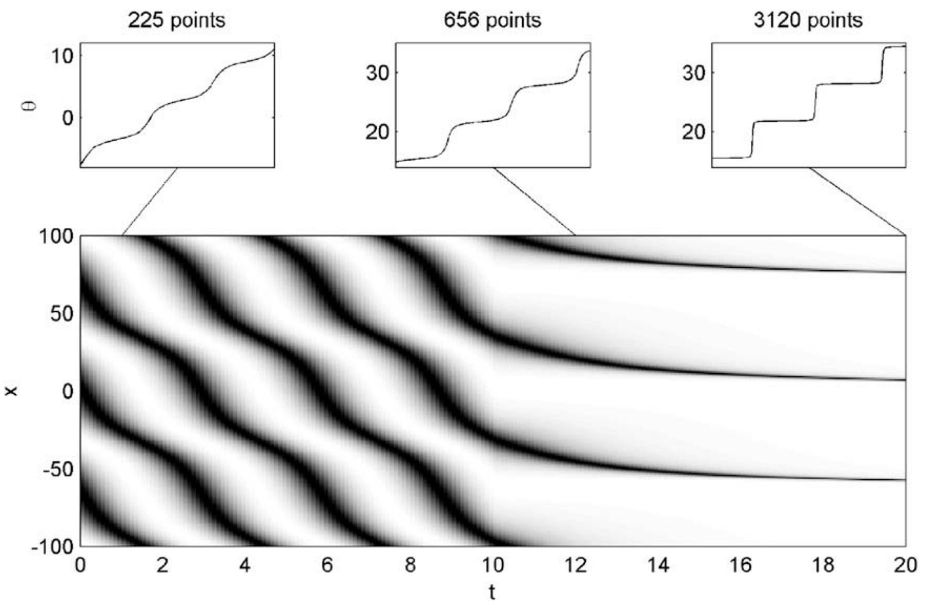

Fig. 3. Solution of neural transmission problem (9) using chebfun. The gray levels show values of cosθ, from -1 (white) to 1 (black), over all x and t. At selected times the function $\theta(x)$ is shown. The solution at $t = 20$ takes 3120 points to represent accurately, as determined by the chebfun constructor.

$$(u(x)-u(0))\bar{u}''(x)+u''(x)(1-E_0)\bar{u}(x)+2u'(x)\bar{u}(x)-(S\bar{u})(x)=-G(u), \tag{15a}$$

$$2u'(0)\bar{u}'(0)-(S\bar{u})(0)=0, \tag{15b}$$

$$\bar{u}'(\pi)=0, \tag{15c}$$

$$\int_0^\pi \bar{u}(x)dx=0. \tag{15d}$$

This is a linear boundary-value problem that can be solved using \. A full implementation of the Newton iteration is given in Appendix B. (Note that we present the details of linearization here for completeness, but in a forthcoming work we describe how automatic differentiation techniques can be used to avoid needing to find or set up the linearization manually [18].) Starting from a quadratic initial guess, six Newton iterations converge to the known exact solution $u=\frac{32}{3\pi}-\frac{16}{3}\sin(x/2)$ to a 2-norm error of less than 3×10^{-15}, taking about 3.1 s altogether.

Different types of dispersion can be studied simply by changing the definition of S from that in (12). For example, the Ostrovsky–Hunter equation [19,20] results if $Su\equiv u$, in which case the code for S is simply `S = eye(d)`. (We can also remove the zero-mean enforcement for the Newton updates, but doing so is not necessary in practice.) A more interesting example is the Whitham–Zaitsev equation given by [21]

$$(Su)(x)=pb^2\left[u(x)-\frac{b}{2\sinh(b\pi)}\int_0^\pi[\cosh(b(|x-y|-\pi))+\cosh(b(x+y-\pi))]u(y)dy\right], \tag{16}$$

for parameters p and b. The absolute value in (16) causes a slope discontinuity in the Fredholm kernel along the line $x = y$. While the chebfun constructor can detect and cope with such discontinuities automatically [4], doing so in this context slows it down dramatically; more importantly, the discrete forms of the Fredholm operator will not be cognizant of the discontinuity and will give greatly reduced accuracy when linearized problems are solved. However, we can break the troublesome integral into smooth pieces as

$$\int_0^\pi \cosh(b(|x-y|-\pi))u(y)dy=\int_0^x \cosh(b(x-y-\pi))u(y)dy+\int_0^{\pi-x}\cosh(b(|x-\pi+s|-\pi))u(\pi-s)ds,$$

where we have used $s = \pi - y$. The first integral on the right is clearly a Volterra operator with a smooth kernel. Suppose we let $v(x)$ be the result of the second integral. If we define the "flip operator" Q by $(Qu)(x) = u(\pi - x)$, then we can write

$$(Qv)(x)=\int_0^x \cosh(b(|x-s|-\pi))(Qu)(s)ds,$$

and we recognize this integral as the same Volterra operator! Finally, using $Q^2 = I$, we have the succinct code

```
[d,x]=domain (0,pi); b = 1; p = 1;
Q = linop (@(n) fliplr (eye (n)),@(u) flipud (u),d);        % flip operator
V = volt (@(x,y) cosh (b*(x-y-pi)),d);
F = fred (@(x,y) cosh (b*(x + y-pi)),d);
S = p * b^2*(1 - b/(2 * sinh (b * pi))*(F + V + Q * V*Q));      % full dispersive operator
```

Everything else about the Newton code in Appendix B remains unchanged. After nine Newton iterations starting from the same quadratic initial guess as before, the 2-norm error of the solution is less than 2×10^{-14}.

The examples above illustrate that mathematical expressions involving operators on functions can be implemented as code in a literal way. The manipulations appear symbolic but are really creating spectral collocation discretizations of the continuous objects. One need not repeat the discretization steps for each new problem involving Fredholm and Volterra operators, and discretization sizes are chosen automatically to create accuracy very close to machine precision. However, as Example 3 shows, one must remain aware of how the underlying methods behave in order to write operators that make sense numerically. More details on the usage of `fred` and `volt`, which are included in chebfun, are available in the online documentation, and a new section on integral equations is in preparation for the user guide that is distributed with the software.

3. High-order scalar equations in integral form

A well-known problem in using Chebyshev spectral methods for differential equations is a loss of accuracy due to ill-conditioning of differentiation matrices [7]. Greengard [1] proposed circumventing this problem by rewriting a second-order, constant-coefficient differential equation as an integral equation for the second derivative of the solution. Coutsias and coauthors [8,9] modified Greengard's techniques somewhat, interpreting the idea as a spectral (as opposed to spectral collocation or pseudospectral) method appropriate for problems with band-limited coefficient functions. In this section Greengard's original approach is extended to linear mth order boundary-value and generalized eigenvalue problems. The process is described in terms of operators that are easily coded using chebfun into adaptive spectral collocation methods without explicit references to discretization.

We begin with the mth order linear boundary-value problem

$$p_m(x)u^{(m)}(x) + p_{m-1}(x)u^{(m-1)}(x) + \cdots + p_0(x)u(x) = f(x), \quad a < x < b, \tag{17a}$$

$$\sum_{j=0}^{m-1} w_{ij}^{(a)} u^{(j)}(a) + w_{ij}^{(b)} u^{(j)}(b) = r_i, \quad i = 1, \dots, m. \tag{17b}$$

We do not consider singular problems, so we assume that $p_m(x)$ is bounded away from zero. In operator form, we have

$$(P_m D^m + P_{m-1} D^{m-1} + \cdots + P_0)u = Au = f, \tag{18}$$

where D is the differentiation operator and P_j is the "diagonal" operator representing pointwise multiplication by p_j. For the boundary conditions (17b) we write

$$\mathbf{r} = \sum_{j=0}^{m-1} \left(\mathbf{w}_j^{(a)} E_a + \mathbf{w}_j^{(b)} E_b \right) D^j u = Wu, \tag{19}$$

where each $\mathbf{w}_j^{(a,b)}$ is an $m \times 1$ vector, and E_a and E_b represent point evaluation functionals at the ends of the interval. The interpretation of W is as a functional mapping the solution into $\mathbf{C}^m$, or, informally, as an operator of dimensions $m \times \infty$. For functions discretized as n-vectors, it becomes an $m \times n$ matrix.

Instead of the usual differential approach in which we solve directly for u, we solve for the highest derivative $u^{(m)}$. Thus, $v = D^m u$ will be a primary unknown. To recover u from v, we will need to introduce constants of integration, later determined by boundary conditions. Define T to be an $\infty \times m$ quasimatrix (that is, an object with finitely many functions as columns [22]) whose columns span the vector space of polynomials of degree less than m. Then $u = C^m v + T\mathbf{k}$, where C is the indefinite integration (`cumsum`) operator and $\mathbf{k}$ is an undetermined m-vector of integration constants. More generally,

$$D^j u = C^{m-j} v + D^j T\mathbf{k}, \quad j = 0, \dots, m. \tag{20}$$

The case $j = 0$ may be considered a definition, with the other cases following as consequences because DC is an identity operator. Note that $D^m T = 0$, so indeed $v = D^m u$.

Formally, we can substitute directly for u in (18) and (19) and solve the block system

$$\begin{bmatrix} AC^m & AT \\ WC^m & WT \end{bmatrix} \begin{bmatrix} v \\ \mathbf{k} \end{bmatrix} = \begin{bmatrix} f \\ \mathbf{r} \end{bmatrix} \tag{21}$$

for the function v and the vector $\mathbf{k}$. The blocks in this "matrix" are an operator, a quasimatrix, a functional, and an ordinary $m \times m$ matrix. A traditional algorithm replaces v by its n-point discretization to obtain an $mn \times mn$ linear system that can be solved simultaneously for $\mathbf{k}$ and the discretized v. To work within the chebfun operator framework, however, we proceed by elimination (much as in Example 3) of the constants $\mathbf{k}$ through Schur complementation:

$$\mathbf{k} = (WT)^{-1}\mathbf{r} - (WT)^{-1}WC^m v. \tag{22}$$

This process requires that the $m \times m$ matrix WT, which consists of boundary condition evaluations of the columns of T, be nonsingular – that is, that there be no nontrivial polynomial of degree less than m that can satisfy the boundary conditions. Proceeding with this assumption, we remove $\mathbf{k}$ from (21) to obtain

$$\left[AC^m - (AT)(WT)^{-1}(WC^m)\right] v = f - (AT)(WT)^{-1}\mathbf{r}, \tag{23}$$

which is a linear integral equation for the function v. In chebfun, we will solve this equation using the \ construct, as explained in the next subsection. Once (23) is solved, we can reconstruct the original u as

$$u = C^m v + Tc = \left[C^m - T(WT)^{-1}(WC^m)\right] v + T(WT)^{-1}\mathbf{r}. \tag{24}$$

We can proceed similarly to reformulate the eigenvalue problem

$$\begin{gathered}(P_m D^m + P_{m-1}D^{m-1} + \cdots + P_0)u = \lambda(Q_m D^m + Q_{m-1}D^{m-1} + \cdots + Q_0)u \\ Au = \lambda Bu,\end{gathered} \tag{25}$$

with the homogeneous boundary conditions $Wu = \mathbf{0}$, where W is as in (19). Again we substitute $u = C^m v + T\mathbf{k}$ and, after the elimination of $\mathbf{k}$ by the boundary constraint, we arrive at

$$\left[AC^m - (AT)(WT)^{-1}(WC^m)\right] v = \lambda\left[BC^m - BT(WT)^{-1}(WC^m)\right] v, \tag{26}$$

which is a generalized eigenvalue problem for v. We want to solve this for λ and v with the chebfun `eigs` command, as explained below. This solution can be followed by the reconstruction of u as in (24), with $\mathbf{r} = \mathbf{0}$.

3.1. Chebfun implementation

The integral equations derived above need to be posed and solved without use of the differentiation operator D, whose large norm and poor conditioning typically degrade the available accuracy. As a consequence, one cannot specify the interior operator A in (18) or the boundary operator W in (19) through a direct differential representation, even though such representations are readily available in the software [3]. Instead, we will represent A through the coefficient functions $p_0,\ldots,p_m$, given as columns of a quasimatrix. The boundary conditions are given through an m-vector $\mathbf{r}$ and the $m \times m$ matrices W_a and W_b, whose entries appear in (17b); that is, the (i,j) entries give in the ith boundary condition the coefficients of $u^{(j-1)}$ evaluated at either endpoint.

The formulas (23) and (26) for the derivative v, and the reconstruction of the original u through (24), require the construction of a number of operators and quasimatrices. We start with the quasimatrix T whose range is the nullspace of D^m:

$$T = [T_0(x)\ T_1(x)\ \cdots\ T_{m-1}(x)], \quad T_0(x) \equiv 1, \quad T_k(x) = \frac{1}{k!}(x-a)^k. \tag{27}$$

Note that $T_k = CT_{k-1}$, because the integration operator C is defined to give a result that is zero at the left endpoint $x = a$. As a result, we have

$$D^j T = [0\ \ldots 0\ T_0\ T_1\ \ldots\ T_{m-j-1}], \quad j = 0,\ldots,m-1, \tag{28}$$

and $D^m T = 0$. Hence terms that include the operator-quasimatrix product AT can be computed stably without numerically computed derivatives as

$$AT = P_1(D^{m-1}T) + \cdots P_{m-1}(DT) + P_m T, \tag{29}$$

applying (28) and a form of Horner's method. Keeping in mind that MATLAB indexes always start at one, not zero, the code becomes

```
AT = 0; DT = T; zero = chebfun (0,d);
for j = 1:m
  AT = AT + diag(P (:,j))*DT;
  DT = [ zero, DT (:,1:m-1) ];
end
```

We also must apply the boundary operator W to T in order to get the $m \times m$ matrix WT that appears often in the formulas. Using (28) and the fact that $T_j(a) = \delta_{j0}$ in (19), it turns out that

$$WT = W_a + W_b \begin{bmatrix} T_0(b) & T_1(b) & \cdots & T_{m-1}(b) \\ & T_0(b) & \cdots & T_{m-2}(b) \\ & & \ddots & \vdots \\ & & & T_0(b) \end{bmatrix}. \tag{30}$$

We should expect WT to be no worse conditioned, typically, than are W_a and W_b, unless perhaps the size of the domain is quite large. In such a case the independent variable or the T_k could be rescaled; we do not pursue this idea further here. In code we may use

```
WT = Wa + Wb * toeplitz([1zeros(1,m - 1)],T(b,:))
```

Our formulas also call for the operator AC^m and the functional WC^m. In the first case we have

$$AC^m = P_0 + \sum_{j=1}^{m} P_j C^j, \tag{31}$$

which can be implemented using Horner's rule:

```
ACm = diag(P (:,1));
for j = 1:m-1
  ACm = ACm * C + diag(P (:,j + 1));
end
ACm = ACm + diag(P (:,m + 1));
```

For the boundary operator W, we obtain a slight simplification by recalling that, by the definition of C, $E_aC = 0$. Thus

$$WC^m = \sum_{j=0}^{m-1} w_j^{(b)} E_b C^{m-j}. \tag{32}$$

The functional E_b itself is quite easy to represent, as seen in Example 3. In sum,

```
Eb = linop(@(n) [zeros (1,n-1) 1],@(u) feval (u,b), d);
WCm = Wb(:,1)*Eb;
for j = 1:m-1
  WCm = WCm * C + Wb(:,j + 1)*Eb;
end
Wm = WCm * C;
```

We now have stable and concise expressions for most of our formulas, but one more matter remains to be explained. The Schur complementation used to eliminate integration constants leads to an operator of the form $(AT)(WT)^{-1}(WC^m)$. Each individual element of this expression has been identified already, but currently chebfun syntax does not allow the expression of the "outer product" or dyad between the $\infty \times m$ quasimatrix AT and the $m \times \infty$ functional $(WT)^{-1}(WC^m)$. This is easily remedied with an explicit construction:

```
mat = @(n) AT(chebpts(d,n),:) * (WT\WCm (n));           % n-by-n realization
op = @(v) AT * (WT\(WCm * v));                          % operational expression
H = linop (mat,op,d);
```

Recall that `chebpts` returns Chebyshev nodes in the domain `d`. An identical construction may be used to create the term $BT(WT)^{-1}WC^m$ appearing in the "mass matrix" of (26).

The operators, matrices, and quasimatrices needed in the formulas for all three problems have been encoded as suggested above in an auxiliary function called `intdata`, given in its entirety in Appendix C. With these elements in place, the solutions of the BVP (18), (19) and the homogeneous eigenvalue problem (26) are straightforward and are given as functions `intbvp` and `inteigs` in Appendix C. The key lines are:

```
v = (ACm - H) \ (f - AT*(WT\r));
k = WT \ (r-WCm * v);
```

for the boundary value problem, and

```
[v,L] = eigs(ACm-H,C^m-R);
lam = diag(L);
k = -WT \ (WCm * v);
```

for the eigenvalue problem. In each case, recall that the solution for the unknown v implies a chebfun construction in which finite linear systems or eigensystems are solved at increasing sizes until the solution is judged to be well resolved.

3.2. Numerical experiments

Example 4. The boundary value problem

$$u'''' - 2\cos(2x)u''' + [48\cos^2(2x)(1+\sin(2x)) - 16\sin(2x)(1-3\sin(2x))]u = 0, \quad x \in [0, 2\pi],$$

with boundary conditions

$$u(0) = 1, \quad u'(0) = 2, \quad u'(2\pi) = 2, \quad u''(2\pi) = 4.$$

has exact solution $u = \exp(\sin(2x))$, which takes 92 points to represent as a chebfun directly from its formula. This problem was solved both in the differential formulation [3] and in the integral formulation given in Appendix C. The condition numbers of the matrices in the two formulations at various values of the discretization parameter N are given in Table 2. These results clearly show the mesh-independence of the condition number in the integral case, as opposed to the $O(N^8)$ growth in the differential case. Accordingly, the integral solution, found in 0.37 s, has an error with 2-norm 4.6×10^{-13}, while the differential solution has an error of norm 4.7×10^{-6}.

These results are typical for problems having operator of type $D^m + \gamma I$ with $|\gamma|$ not too large, because it converts to the integral form $I + \gamma C^m$, which is well conditioned by the compactness of C. For problems of type $\epsilon D^m + I$, however, the integral form is $\epsilon I + C^m$, which may be poorly conditioned when $|\epsilon| \ll 1$. In such cases the differential form may be preferable at realistic values of N, despite the rapid growth as $N \to \infty$.

Example 5. The following sixth-order generalized eigenvalue problem arises for the modes of vibration of a circular ring with rectangular cross section and symmetric but nonuniform thickness [23]:

$$\begin{aligned} &\frac{\phi}{\pi^4} w^{(6)} + \frac{3\phi'}{\pi^4} w^{(5)} + \left(\frac{2\phi}{\pi^2} + \frac{3\phi''}{\pi^4}\right) w^{(4)} + \left(\frac{4\phi'}{\pi^2} + \frac{\phi'''}{\pi^4}\right) w''' + \left(\phi + \frac{3\phi''}{\pi^2}\right) w'' + \left(\phi' + \frac{\phi'''}{\pi^2}\right) w' \\ &\quad = \omega^2 \left(f w'' + f' w' - \pi^2 f w\right), \end{aligned} \tag{33}$$

for $0 < x < 1$, where $f(x) = -4(r-1)x^2 + 4(r-1)x + 1$ for parameter r represents the thickness as a function of angle, and $\phi = f^3$. In what follows we use $r = 1$. The displacement w is subject to the conditions $w = w' = w''' = 0$ at both $x = 0$ and $x = 1$.

This problem is in a form directly amenable to the codes from Appendix C. The first four modes and associated frequencies $\sqrt{\omega}$ are shown in Fig. 4. Estimating the error in these solutions is not straightforward. Simply substituting a computed mode and eigenvalue into (33) will not suffice, because the computation of high-order derivatives will introduce significant roundoff. Instead, the numerical solution of the integral formulation for $w^{(6)}$ can be integrated up one order at a time to give accurate representations of all of the lower derivatives. Using this process to substitute for the solution and its derivatives in (33), and computing the (continuous) 2-norm of the residual of the equation over $x \in [0,1]$, leads to values ranging from about 1.3×10^{-11} to 5.8×10^{-6} for the four modes shown. Modes computed by a direct differential discretization of (33) agree with those computed by integration to about four digits, but the residuals of the differentially computed modes are as much as seven orders of magnitude larger near the boundaries.

Example 6. Finding the eigenvalues of the Orr–Sommerfeld operator for plane Poiseuille flow is a classic problem in hydrodynamic stability, one long associated with spectral methods [25]. The operator can be written as the generalized fourth-order eigenvalue problem

$$u'''' - [2\alpha^2 + i\alpha R(1-x^2)]u'' + [\alpha^4 - 2i\alpha R + i\alpha^3 R(1-x^2)]u = \lambda R(u'' - \alpha^2 u),$$

Table 2
Condition numbers of matrices in the linear systems arising in the integral and differential formulations of Example 4.

	$n = 9$	$n = 17$	$n = 33$	$n = 65$	$N = 128$
Integral formulation	9.87×10^3	3.13×10^4	4.80×10^4	4.70×10^4	4.67×10^4
Differential formulation	7.73×10^1	2.27×10^4	9.85×10^6	4.21×10^9	1.61×10^{12}

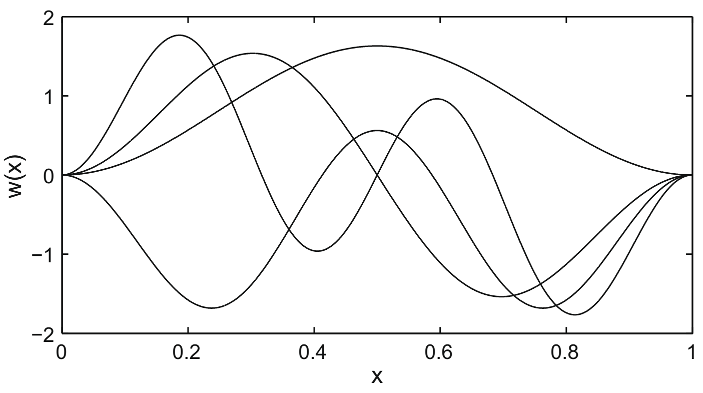

Fig. 4. First four eigenmodes of the nonuniform circular ring vibrations described by (33). The computed frequency $\sqrt{\omega}$ in each case is 2.266742077, 6.923297244, 13.97766883, and 22.81956261. The first value agrees with a result previously reported to five digits [24].

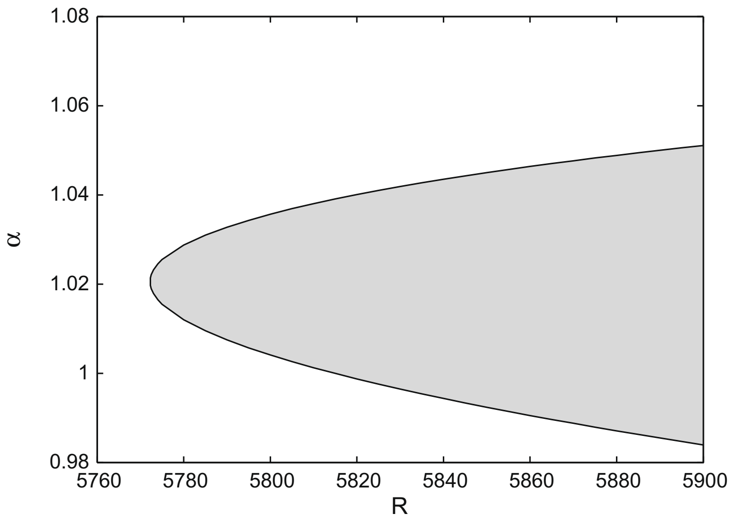

Fig. 5. Unstable region for the Orr–Sommerfeld operator in the (R, α) plane.

for $x \in [-1, 1]$, with $u(\pm 1) = u'(\pm 1) = 0$. Here R is the Reynolds number of the flow and α is the streamwise wavenumber of the flow disturbance. We can embed eigenvalue calculations inside a loop that constructs, for various fixed values of R, a chebfun approximation of $M = \max_j\{\text{Re}\lambda_j\}$ over a range of α. By finding the roots of these chebfuns, we can draw the boundary of the unstable region in (R, α) space. The resulting region is shown in Fig. 5. The code used for the calculation is given in Appendix D. The entire calculation for Fig. 5 took 756 s.

4. Iterative methods and nonsmooth problems

The reader can be forgiven at this point for wondering whether the operator manipulations of the previous section are needlessly cumbersome. After all, if one discretizes from the beginning then one has more or less the same operations to go through with matrices, without the (small) additional syntactic overhead of the chebfun references. However, in this section we can begin to exploit the advantages of the simultaneous matrix/operator definitions maintained by the chebfun operator objects.

Both `intbvp` and `inteigs` from Appendix C use dense linear algebra methods on matrix discretizations of the corresponding integral equations. When the matrices are very large, of course, the $O(n^3)$ work requirements of dense linear algebra become troublesome. A more subtle drawback of the matrix realization of these problems is the underlying global Chebyshev discretization of the interval on which they are posed. In the integral formulations, slope or jump discontinuities in the coefficient functions present no difficulties in principle; if one allows a piecewise representation of $v = D^m u$, integrations of it are spectrally accurate and straightforward, and no matching or patching conditions are necessary as in the differential formulation. However, the matrices corresponding to piecewise discretizations are available but much less straightforward to express and construct than in the global case.

The chebfun operator objects allow us to sidestep both limitations of matrices. The chebfun package includes a `gmres` method for solving linear equations in the form $Au = f$, where A is represented as an operator and f is a chebfun. The `gmres` method does *not* call the built-in MATLAB routine for solving finite-dimensional systems of equations at increasing values of n. Rather, it implements a functional form of the GMRES iteration in which the vectors used to span the Krylov subspace are not of fixed, identical finite dimension but are notionally members of an infinite-dimensional function space. The iteration refers to the operator A only through applications `A * u` for iteratively generated chebfuns `u`. Thus, the functional applications of A are used exclusively and the `gmres` iteration works entirely with a matrix-free representation of the operator A. The connections between integral equations, mesh independence, and GMRES have been noted previously [26] and are fairly clear, while the application of GMRES for differential operators is less mature [27] and the right method for incorporation of boundary conditions and discontinuities remains unclear.

In terms of the `intbvp` code in Appendix C, we need to change only one line to use GMRES, replacing the construction of `v` using backslash with the line

```
v = gmres(ACm − H, f − AT * (WT \ r))
```

Example 7. Consider the nonlinear boundary-value problem $y'' + |y| = 0$, for $0 \leqslant x \leqslant 4$, with $y(0) = 0$ and $y(4) = -2$. (This problem is shipped with standard MATLAB as the file `twobvp`.) This problem has two solutions, one of which is smooth and the other having a corner in y'' due to a change of sign in y. The location of this corner is not known in advance. If we perturb a function y satisfying the end conditions to $y + u$, then a naive linearization suggests that u should solve the BVP

$$u'' + \operatorname{sign}(y)u = 0, \quad u(0) = u(4) = 0. \tag{34}$$

The lack of smoothness in $|y|$ (technically, it lacks a Fréchet derivative) makes the validity of (34) suspect at best. Nevertheless, we can build a simple Newton iteration based on it.

```
[d,x] = domain (0,4);  % initial guess satisfying the BCs
y = x.*(4-x) - x/2;
D2y = diff (y,2);
Wa=[1 0;0 0]; Wb=[0 0;1 0]; r=[0;0]; % homogeneous BCs for u
resid = 1; u = 1; % first iteration placeholders
while (norm (resid) > 1e-11) && (norm (u) > 1e-11)
resid = D2y + abs (y); % residual of the nonlinear ODE
  p = [sign (y),0,1]; % coefficients in the linearization ODE
  [u,uder] = intbvp (p,-resid,Wa,Wb,r);
  y = y + u; D2y = D2y + uder (:,1); % Newton correction
end
```

Because the linearized problems have discontinuous coefficients, `intbvp` has to be modified here to use `gmres` as described above. Note that to maintain accuracy, the second derivative y'' is not computed directly within each iteration but is updated using the accurate u'' returned by `intbvp`. The Newton iteration converges in three steps, producing corrections of norm 1.63, 0.284, and 3.14×10^{-12}, and a final residual that is numerically exactly zero. The solution and its second derivative are shown in Fig. 6.

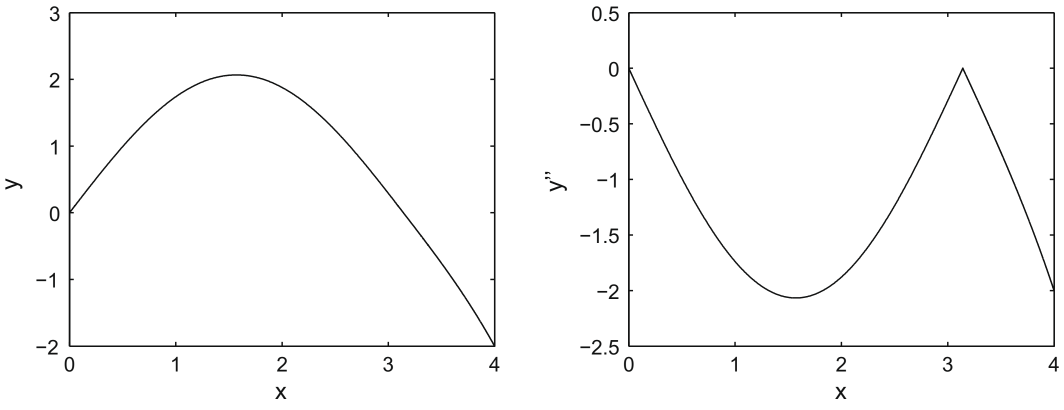

Fig. 6. Solution of $y'' + |y| = 0$ using a Newton iteration with integral-formulation solutions of the linearization. Despite the lack of smoothness in the linearization, the residual of the solution shown is numerically zero, and the discontinuity location is discovered automatically as part of the iteration.

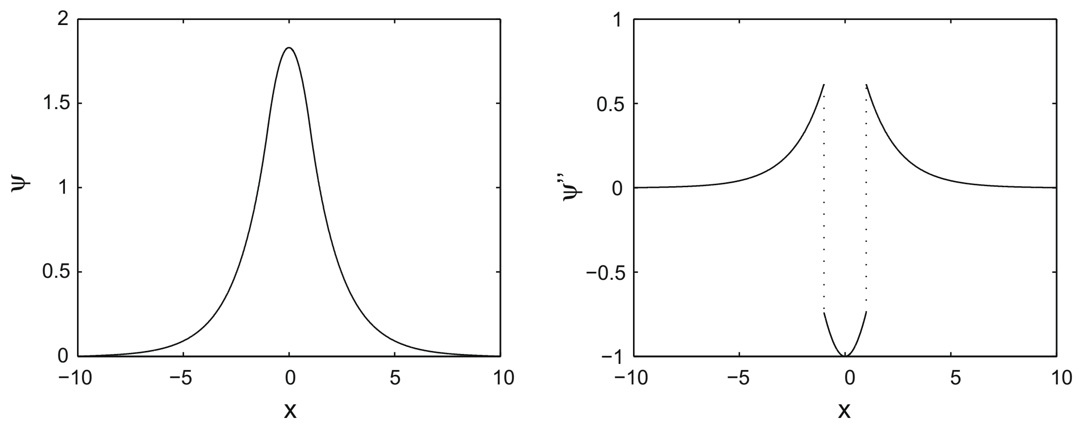

Fig. 7. Eigenfunction of Schrödinger's equation with a step potential. A generalized eigenproblem is set up for the discontinuous ψ'' and solved by iteration on integration and pointwise multiplication and addition of functions. The entire solution, accurate to within one or two digits of double precision, is ultimately represented using a vector of 149 sample values and exact knowledge of the jump locations in the potential.

Solving eigenvalue operator problems with iterative linear algebra is not as straightforward as for linear systems, as there is currently no chebfun implementation of, say, a functional Arnoldi iteration. (The built-in MATLAB `eigs` command, which uses the implicitly restarted Arnoldi iteration of ARPACK [28], accepts a functional form of the operator but still works only on vectors of fixed finite length.) Here we settle for a demonstration of a less powerful, but still useful, functional inverse iteration.

Example 8. Here we find the eigenvalues of the Schrödinger equation $-\psi'' + V\psi = \lambda\psi$, for $-10 \leqslant x \leqslant 10$, with homogeneous Dirichlet conditions on ψ, and the step potential

$$V(x) = \begin{cases} 0, & |x| < 1 \\ 1, & |x| > 1. \end{cases}$$

Once we define a chebfun representation for V, we can use the `intdata` code of Appendix C to create operators A and B for a generalized eigenproblem $Av = \lambda Bv$, where $v = \psi''$. Then the Rayleigh quotient iteration $(A - \sigma_n B)v^{n+1} = Bv^n$, where GMRES is used to solve the linear system for each iteration, takes the form

```
v = chebfun (1,d); lam = 0;
for n = 1:6
  v = gmres(A-lam * B,B * v);
  v = v/norm (v,inf); v.scl = 1;
  lam = v' *(A * v)/ (v' *(B * v))
end
```

(Note that the residual tolerances on the inner GMRES iterations, which we have left fixed here, can be relaxed somewhat without much harm to the convergence of the outer iterations [29].) The eigenvalue estimates converge quadratically to 0.546250046356179, which further iterations do not change except in the last two digits. The resulting eigenfunction ψ is shown in Fig. 7.

5. First-order systems in integral form

We conclude with a discussion of generalizing Greengard-style integral formulations to the first-order system BVP

$$B(x)\mathbf{u}'(x) - A(x)\mathbf{u}(x) = \mathbf{f}(x), \quad a < x < b, \tag{35a}$$
$$W_a\mathbf{u}(a) + W_b\mathbf{u}(b) = \mathbf{r}, \tag{35b}$$

in which $\mathbf{u}$, $\mathbf{f}$, and $\mathbf{r}$ are all in $\mathbf{C}^m$, and $A(x)$ and $B(x)$ are in $\mathbf{C}^{m\times m}$. Informal experiments suggest that the case for an integral formulation of (35) does not seem to be compelling on the grounds of condition numbers, presumably because the differential form does not have explicit high-order derivatives. However, while high-order problems can always be cast in first-order form, doing so increases the discretization size, which increases linear algebra costs significantly, and the integral form does offer clear advantages for iterative linear algebra and nonsmooth problems, as described in Section 4. An extension of the expressions in this section to systems of higher-order equations should be conceptually straightforward, though notationally awkward.

In operator terms, we write (35a) as

$$\mathsf{B}\mathsf{D}\mathbf{u} - \mathsf{A}\mathbf{u} = \mathbf{f}, \tag{36}$$

where we use the sans-serif letters to denote block operators on vectors of functions:

$$\mathsf{A} = \begin{bmatrix} \operatorname{diag}(a_{11}(x)) & \cdots & \operatorname{diag}(a_{1m}(x)) \\ \vdots & & \vdots \\ \operatorname{diag}(a_{m1}(x)) & \cdots & \operatorname{diag}(a_{mm}(x)) \end{bmatrix}, \quad \mathsf{B} = \begin{bmatrix} \operatorname{diag}(b_{11}(x)) & \cdots & \operatorname{diag}(b_{1m}(x)) \\ \vdots & & \vdots \\ \operatorname{diag}(b_{m1}(x)) & \cdots & \operatorname{diag}(b_{mm}(x)) \end{bmatrix},$$

$$\mathsf{C} = I_m \otimes C = \begin{bmatrix} C & & 0 \\ & \ddots & \\ 0 & & C \end{bmatrix}, \quad \mathsf{D} = I_m \otimes D. \tag{37}$$

Following the style of (19), we also condense the boundary conditions (35b) into the linear relation

$$\mathsf{W}\mathbf{u} = \mathbf{r}, \tag{38}$$

in which W maps a vector of m functions into $\mathbf{C}^m$. Using Kronecker product notation, we can write

$$\mathsf{W} = (W_a \otimes E_a) + (W_b \otimes E_b). \tag{39}$$

To understand this expression for W, it may help to think in terms of discretization. The matrices W_a and W_b are each simply $m \times m$ collections of coefficients describing the boundary conditions of the problem. If the solution $\mathbf{u}(x)$ is replaced by its discretization, it becomes a block vector of total length mn. We would then replace the endpoint evaluation functionals E_a and E_b with the first and last rows respectively of an $n \times n$ identity matrix, leaving the discrete W as an $m \times mn$ matrix ready to operate on the discretized $\mathbf{u}(x)$. All of the formulas that follow have correspondingly concrete interpretations when discretization is introduced.

Instead of solving for u in (35), we will solve for $\mathbf{v} = \mathsf{D}\mathbf{u}$, so that

$$\mathbf{u} = \mathsf{C}\mathbf{v} + \mathbf{k}, \tag{40}$$

for an unknown constant vector $\mathbf{k} \in \mathbf{C}^m$. Note that we can interpret $\mathbf{k}$ as a vector of constant functions. When (40) is substituted into (36), for instance, applying A to $\mathbf{k}$ simply results in $A\mathbf{k}$, that is, linear combinations of the entries of the coefficient matrix A in the original Eq. (35a). Similarly, applying W to $\mathbf{k}$ via (39) results in $(W_a + W_b)\mathbf{k}$, because evaluation of $\mathbf{k}$ at either endpoint leaves it unchanged. Hence the ODE and boundary conditions become

$$\begin{bmatrix} \mathsf{B} - \mathsf{A}\mathsf{C} & A \\ \mathsf{W}\mathsf{C} & (W_a + W_b) \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{k} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{r} \end{bmatrix}. \tag{41}$$

We can similarly reformulate the generalized eigenproblem

$$B(x)\mathbf{u}'(x) = \lambda A(x)\mathbf{u}(x), \quad a < x < b, \tag{42a}$$

$$W_a\mathbf{u}(a) + W_b\mathbf{u}(b) = \lambda(Y_a\mathbf{u}(a) + Y_b\mathbf{u}(b)), \tag{42b}$$

which allows the eigenvalue to be part of the boundary conditions, as required in some problems (such as the instability of shear flows over porous layers [30]), into

$$\begin{bmatrix} \mathsf{B} & 0 \\ \mathsf{W}\mathsf{C} & (W_a + W_b) \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{k} \end{bmatrix} = \lambda \begin{bmatrix} \mathsf{A}\mathsf{C} & A \\ \mathsf{Y}\mathsf{C} & (Y_a + Y_b) \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{k} \end{bmatrix}. \tag{43}$$

Both (41) and (43) require the simultaneous solution for a function $\mathbf{v}$ and a constant vector $\mathbf{k}$. In Example 3 and Section 3, we were able to eliminate the auxiliary constants by hand to obtain a purely functional problem. In principle, we can proceed the same way here and eliminate the integration constants $\mathbf{k}$ to get a block system for functions only. However, doing so requires that the matrix $W_a + W_b$ describing boundary conditions be nonsingular, and there are some rather ordinary problems for which this is not true. For example, if there are $m = 2$ variables and both boundary conditions are imposed only on the first component u_1, then

$$W_a = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad W_b = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix},$$

and $W_a + W_b$ is singular. This does not imply that the full operator in (41) is singular, just that $\mathbf{k}$ cannot be independently eliminated.

Currently, chebfun does support block linear operators, but not more general block constructs such as those in (41) and (43) that combine operators, quasimatrices, functionals, and matrices. These objects make sense as matrices after discretization, however. Given the difficulty with elimination of the constant term $\mathbf{k}$ in the continuous formulation, a better method of attack appears to be direct assembly and solution of a discrete form of the full system like (35a), within an iteration on the discretization size *n*. In principle, this could be accomplished within a generalized form of \ and expanded object syntax, but code for this capability requires changes at a deeper level, and development has not been undertaken to date.

Acknowledgments

The author thanks Folkmar Bornemann for his insights about the efficiency of the Fredholm operator construction and kernel function evaluation, and for his contributions to the released versions of the code. The author also thanks Nick Trefethen for many useful comments about drafts of the paper.

Support for the development of the chebfun software system, including for the author, was provided by the UK EPSRC Grant EP/E045847.

Appendix A. Code for neuronal transmission problem

This code was used for Example 2 in Section 2.

```
function theta = neuron(mu,sigma)
chebfunpref('eps',1e-10) % desired accuracy
cheboppref('maxstorage',300e6); % cache LU factors
cheboppref('maxdegree',5000) % allow degree 5000 polynomials
splitting off
L = 100; [d,x] = domain(-L-8 * sigma,L + 8 * sigma);
% Problem definition: A*(d theta/dt) = f (theta,t)
ker = @(x,y) exp (-(x-y).^2/sigma^2)/ (sigma * sqrt (pi));
F = fred ( @kernel, d,1);
A = 1-mu * F;
f = @(theta,t) cos (theta) + 1 + double (t<=10);
t = 0; dt = 0.2;
theta = 3 * pi * x/L;  % initial condition
% RK4 stepping
for kt = 1:20/dt
  s1 = dt*(A\f (theta,t));
  s2 = dt*(A\f (theta + 0.5 * s1,t + 0.5 * dt));
  s3 = dt*(A\f (theta + 0.5 * s2,t + 0.5 * dt));
  s4 = dt*(A\f (theta + s3,t + dt));
  theta = theta + (s1 + s4 + 2*(s2 + s3))/6; t = t + dt;
end
function K = kernel (x,y)
  if nargin==2     % direct evaluation
    K = ker (x,y);
  else       % sparse evaluation for speed
    n = length (x);
    d = find ( abs (x-x (1)) < 6 * sigma ) - 1;  % estimate the bandwidth
    d = union (d,-d);
    B = zeros (n,length (d));
    for k = 1:length (d)
      if d (k) >= 0
        B (1 + d (k):n,k) = ker (x (1:n-d (k)),x (1 + d (k):n));
      else
        B (1:n + d (k),k) = ker (x (-d (k)+1:n),x (1:n + d (k)));
      end
    end
    K = spdiags (B,d,n,n);
  end
end
end
```

Appendix B. Code for corner wave problem

This code was used for Example 3 in Section 2.

```
chebfunpref factory
[d,x] = domain (0,pi);
S = fred (@(x,y) cos (x-y)+cos (x + y),d);   % dispersion operator
G = @(u) (u - u (0)).*diff (u,2) + diff (u).^2- S * u;  % nonlinear IDE
g1 = @(u) feval ( diff (u).^2-S * u, 0);  % left BC
g2 = @(u) feval ( diff (u), pi);        % right BC
% Define the left-evaluation operator E0.
mat = @(n) ones (n,1)*[1 zeros (1,n-1)];
op = @(u) chebfun (u (0),d);
```

```
EO = linop (mat,op,d);
% Define the linearization (Jacobian operator plus boundary conditions).
D = diff (d);
jac = @(u) diag (diff (u,2))*(1-EO) + diag (u-u (0))*D^2 + 2 * diag (diff (u))*D - S;
bcl = @(u) {2 * feval (diff (u),0)*D - S, -g1(u)};
bcr = @(u) {D, -g2(u)};
u = (x-pi).^2; u = u-sum (u)/pi;    % initial guess with zero mean
du = 1;                             % force first iteration
while norm (du) > 1e-10
  A = -jac (u);
  A.lbc = bcl (u); A.rbc = bcr (u);    % BC's of the linear IDE
  A.rbc (2) = cumsum (d);              % add the zero-mean condition
  A.scale = u (0);                    % prevent over-resolution
  du = A\G (u);                       % Newton correction
  u = u + du;
end
```

Appendix C. Codes for integral formulations of BVPs and eigenvalue differential equations

Presently, the codes in this section are available from the author. They may be distributed as part of future chebfun releases, along with updated information and instruction on syntax and usage. The function `intdata` here is used by both `intbvp` and `inteigs` below.

```
function [AT,ACm,WT,WCm,C,T,H] = intdata (P,Wa,Wb)
m = size (P,2)-1;    % order of ODE
d = domain (P (:,1));
a = d (1); b = d (end);
C = cumsum (d);
% Formulate quasimatrix for null space of D^m.
T = chebfun (1,d);
for k = 2:m, T (:,k) = C * T (:,k-1); end
% Interior operator A applied to T, by Horner's method.
AT = 0; DT = T;
for j = 1:m
  AT = AT + diag (P (:,j))*DT;
  DT = [chebfun (0,d), DT (:,1:m-1) ];
end
% Boundary operator W applied to T.
WT = Wa + Wb * toeplitz ( [1 zeros (1,m-1)], T (b,:) );
% Operators applied to C^m integration operator, again using Horner.
ACm = diag (P (:,1));
Eb = linop (@(n) [zeros (1,n-1) 1],@(u) feval (u,b), d);
WCm = Wb (:,1)*Eb;
for j = 1:m-1
  ACm = ACm * C + diag (P (:,j + 1));
  WCm = WCm * C + Wb (:,j + 1)*Eb;
end
ACm = ACm * C + diag (P (:,m + 1));
WCm  = WCm * C;
% Chebop form of ``outer products".
mat = @(n) AT(chebpts(d,n),:) * (WT\WCm (n));   % n-by-n realization
op = @(v) AT * (WT\(WCm * v));   % functional expression
H = linop (mat,op,d);
end
```

This code solves a scalar differential boundary-value problem posed as described in Section 3.

```
function [u,uder] = intbvp (P,f,Wa,Wb,r)
m = size (P,2)-1;  % order of ODE
[C,T,AT,WT,ACm,WCm,H] = intdata (P,Wa,Wb);
% Construct chebfun solution for v = D^m * u, and integration constants.
v = (ACm - H) \ (f - AT*(WT\r));
k = WT \ (r-WCm * v);
% Reconstruct all derivatives of the solution accurately.
Cjv = v; uder = v;
for j = 1:m
  Cjv = C * Cjv;
  uder (:,j + 1) = Cjv + T (:,1:j)*k (m-j + 1:m);
end
u = uder (:,end);  % same as u = C^m * v + T * k
```

This code solves a scalar differential eigenvalue problem posed as described in Section 3.

```
function [u,lam,uder] = inteigs (P,Q,Wa,Wb)
m = size (P,2)-1; % order of ODE
[C,T,AT,WT,ACm,WCm,H,R] = intdata (P,Wa,Wb);
[t1,t2,BT,t3,BCm] = intdata (Q,Wa,Wb);
% Construct chebfun solution for v = D^m * u, and integration constants.
[v,L] = eigs (ACm-H,C^m-R);
lam = diag (L);
k = -WT \ (WCm * v);
u = C^m * v + T * k;
end
```

Appendix D. Code for Orr–Sommerfeld stability region calculation

The following function uses code from Appendix C to compute Orr–Sommerfeld eigenvalues as a function of R and α.

```
function lam = OSeig (R,a)
[d,x] = domain (-1,1);
Wa = zeros (4); Wa (1,1) = 1; Wa (2,2) = 1;
Wb = zeros (4); Wb (3,1) = 1; Wb (4,2) = 1;
p = [ a^4−2i * a*R + 1i * a^3 * R*(1-x.^2), 0, ...
        −2 * a^2− 1i * a*R*(1-x.^2), 0, 1 ];
q = [-R * a^2, 0 * x, R, 0, 0];
[u,lam] = inteigs (p,q,Wa,Wb);
```

This is embedded in a loop as follows.

```
R = [5772.3 5772.5 5773 5774 5775:5:5900]
chebfunpref ('eps',1e-8)   % relax the accuracy
b = {}; M = chebfun;
for j = 1:length (R)
  M (:,j) = chebfun ( @(a) max (real (OSeig (R (j),a))), [0.98 1.08]);
  b{j} = roots (M (:,j));
end
```

Each branch of the resulting `b` is one side of the boundary of the region in Fig. 5.

Appendix E. Supplementary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.jcp.2010.04.029.

References

[1] L. Greengard, Spectral integration and two-point boundary value problems, SIAM J. Numer. Anal. 28 (1991) 1071–1080.
[2] L.N. Trefethen, Z. Battles, An extension of MATLAB to continuous functions and operators, SIAM J. Sci. Comput. 25 (2004) 1743–1770.
[3] T.A. Driscoll, F. Bornemann, L.N. Trefethen, The chebop system for automatic solution of differential equations, BIT 48 (2008) 701–723.
[4] R. Pachón, R. Pachon, R.B. Platte, L.N. Trefethen, Piecewise-smooth chebfuns, IMA J. Numer. Anal. Doi: 10.1093/imanum/drp008.
[5] Z. Jackiewicz, M. Rahman, Z. Jackiewicz, M. Rahman, B.D. Welfert, Numerical solution of a Fredholm integro-differential equation modelling \dot{θ}-neural networks, Appl. Math. Comput. 195 (2) (2008) 523–536.
[6] J.P. Boyd, A Legendre-pseudospectral method for computing travelling waves with corners (slope discontinuities) in one space dimension with application to Whitham's equation family, J. Comput. Phys. 189 (1) (2003) 98–110.
[7] W. Heinrichs, Improved condition number for spectral methods, Math. Comput. 53 (187) (1989) 103–119.
[8] E.A. Coutsias, T. Hagstrom, D. Torres, An efficient spectral method for ordinary differential equations with rational function coefficients, Math. Comput. 65 (1996) 611–635.
[9] E.A. Coutsias, T. Hagstrom, J.S. Hesthaven, D. Torres, Integration preconditioners for differential operators in spectral τ-methods, in: A.V. Ilin, L.R. Scott (Eds.), Proceedings of the Third International Conference on Spectral and High Order Methods, Houston, TX, 1996, pp. 21–38.
[10] N. Mai-Duy, An effective spectral collocation method for the direct solution of high-order ODEs, Commun. Numer. Method. Eng. 22 (6) (2006) 627–642.
[11] N. Mai-Duy, R.I. Tanner, A spectral collocation method based on integrated Chebyshev polynomials for two-dimensional biharmonic boundary-value problems, J. Comp. Appl. Math. 201 (2007) 30–47.
[12] B. Fornberg, A Practical Guide to Pseudospectral Methods, Cambridge Universtiy Press, Cambridge, 1996.
[13] L.N. Trefethen, Spectral Methods in MATLAB, SIAM, Philadelphia, 2000.
[14] K. Atkinson, The Numerical Solution of Integral Equations of the Second Kind, Cambridge Univ. Press, 1997.
[15] H. Brunner, High-order methods for the numerical solution of Volterra integro-differential equations, J. Comp. Appl. Math. 15 (1986) 301–309.
[16] S. Gabov, On Whitham's equation, Sov. Math. Dokl. 19 (1978) 1225–1229.
[17] M. Shefter, R.R. Rosales, Quasiperiodic solutions in weakly nonlinear gas dynamics. Part I. Numerical results in the inviscid case, Stud. Appl. Math. 103 (1999) 279–337.
[18] A. Birkisson, T.A. Driscoll, Automatic Fréchet differentiation for the numerical solution of boundary value problems, in preparation.
[19] R.H.J. Grimshaw, J.-M. He, L.A. Ostrovsky, Terminal damping of a solitary wave due to radiation in rotational systems, Stud. Appl. Math. 101 (2) (1998) 197–210.
[20] J.K. Hunter, Numerical solutions of some nonlinear dispersive wave equations, Lectures Appl. Math. 26 (1990) 301–316.
[21] G.B. Whitham, Linear and Nonlinear Waves, Wiley, New York, 1974.

[22] G.W. Stewart, Afternotes Goes to Graduate School, Society for Industrial and Applied Mathematics, Philadelphia, 1998.
[23] R.H. Gutierrez, P.A.A. Laura, Vibrations of non-uniform rings studied by means of the differential quadrature method, J. Sound Vibr. 185 (3) (1995) 507–513.
[24] Y. Wang, Y.B. Zhao, G.W. Wei, A note on the numerical solution of high-order differential equations, J. Comp. Appl. Math. 159 (2) (2003) 387–398.
[25] S.A. Orszag, Accurate solution of the OrrSommerfeld stability equation, J. Fluid Mech. 50 (04) (1971) 689–703.
[26] C.T. Kelley, Z.Q. Xue, GMRES and integral operators, SIAM J. Sci. Comput. 17 (1996) 217–226.
[27] S. Olver, GMRES for the differentiation operator, SIAM J. Numer. Anal. 47 (2009) 3359–3373.
[28] R.B. Lehoucq, D.C. Sorensen, C. Yang, ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods, SIAM, 1998.
[29] J. Berns-Müller, A. Spence, Inexact inverse iteration with variable shift for nonsymmetric generalized eigenvalue problems, SIAM J. Matrix Anal. Appl. 28 (2006) 1069–1082.
[30] M.-H. Chang, F. Chen, B. Straughan, Instability of Poiseuille flow in a fluid overlying a porous layer, J. Fluid Mech. 564 (2006) 287–303.