Chebfun: A New Kind of Numerical Computing

R.B. Platte[1] and L.N. Trefethen[2]

[1] School of Mathematical and Statistical Sciences, Arizona State University, Tempe, AZ 85287-1804, USA, platte@math.asu.edu
[2] Oxford University Mathematical Institute, Oxford OX1 3LB, UK trefethen@maths.ox.ac.uk

Summary. The functionalities of the chebfun and chebop systems are surveyed. The chebfun system is a collection of MATLAB codes to manipulate functions in a manner that resembles symbolic computing. The operations, however, are performed numerically using polynomial representations. Chebops are built with the aid of chebfuns to represent linear operators and allow chebfun solutions of differential equations. In this article we present examples to illustrate the simplicity and effectiveness of the software. Among other problems, we consider edge detection in logistic map functions and the solution of linear and nonlinear differential equations.

1 Introduction

For a long time there have been two kinds of mathematical computation: symbolic and numerical. Symbolic computing manipulates algebraic expressions exactly, but it is unworkable for many applications since the space and time requirements tend to grow combinatorially. Numerical computing avoids the combinatorial explosion by rounding to 16 digits at each step, but it works just with individual numbers, not algebraic expressions.

The chebfun system introduced in 2004 by Battles and Trefethen [1] aims to combine the feel of symbolics with the speed of numerics. The idea is to represent functions by Chebyshev expansions whose length is determined adaptively to maintain an accuracy of close to machine precision. The system has been significantly extended since its introduction. Among other developments, it now handles piecewise smooth functions on arbitrary intervals [2] and linear operators [3]. The latter extension was made possible by T. A. Driscoll who implemented the *chebop* class. In this article, we review the main features of the software and demonstrate its effectiveness through many examples, including solution of differential equations.

A.D. Fitt et al. (eds.), *Progress in Industrial Mathematics at ECMI 2008*, Mathematics in Industry 15, DOI 10.1007/978-3-642-12110-4_5,

The chebfun system is implemented in object-oriented MATLAB. One of the guiding principles in its design is the analogy of commands available for vectors and those implemented in the chebfun package for functions. Once a chebfun object has been created, commands like `diff`, `sum` and `norm` can be used to compute its derivative, definite integral, and norm, respectively.

The simplicity of its use is illustrated in the following example, where the number of roots, maximum and L_1-norm of the function $f(x) = J_0(x)\sin x$ are computed in the interval $[0, 100]$.

```
>> f = chebfun(@(x) besselj(0,x).*sin(x), [0 100]);
>> length(roots(f))
ans = 64
>> max(f)
ans = 0.644562281456927
>> norm(f,1)
ans = 6.295294435933753
```

Similarly, the chebop extension to linear operators relies on underlying polynomial-based spectral methods. The analogy here, to some extent, is between linear operators and matrices. With chebops, commands such as `diff` and `sum` are used to define differential and integral operators, while "`*`" and "\" are used to apply operators in forward and inverse modes. The following commands, for example, can be used to differentiate $f(x) = \sin(x)$ in $[-\pi, \pi]$ using chebop notation.

```
[d,x] = domain([-pi,pi]);
D = diff(d);
df = D*sin(x);
```

One of the main strengths of chebops is how simple the syntax is for solving differential equations. To solve the boundary value problem

$$u''(x) + u'(x) + u(x) = \sin(x), \quad x \in (-\pi, \pi), \quad u(\pm\pi) = 0,$$

for instance, one only needs to define the operator and appropriate boundary conditions and type \,

```
L = D^2 + D + eye(d) & 'dirichlet';
sol = L\sin(x);
```

This article is organized in two main sections. In Sect. 2 we review basic aspects of the chebfun system, including piecewise polynomial representations and apply the chebfun edge detector to locate break points of piecewise constant functions that are limits of logistic map sequences. In Sect. 3 we briefly describe the syntax of the chebop system and give examples to illustrate how simple and effective it is.

2 Chebfuns

In this section we give some insight into the underlying theory and implementation of the system. More detailed information can be found in [1] and [4].

2.1 Funs: Smooth Representations on $[-1, 1]$

The original chebfun class implemented by Battles in 2004 for smooth functions on $[-1, 1]$ is now called *fun*. A chebfun object consists of one or more funs. Each smooth piece is mapped to the interval $[-1, 1]$ and represented by an expansion in Chebyshev polynomials of the form

$$f_N(x) = \sum_{j=0}^{N} \lambda_j T_j(x), \qquad x \in [-1, 1], \tag{1}$$

where $T_j(x) = \cos(j \arccos(x))$. When constructing a fun object, the system computes the coefficients λ_j by interpolating the target function f at $N + 1$ Chebyshev extreme points,

$$x_j = \cos \frac{\pi j}{N}, \qquad j = 0, \ldots, N.$$

The polynomial degree N is automatically determined so that the representation is as accurate as possible in double precision arithmetic.

Polynomial interpolation in Chebyshev nodes is known to be near-optimal for approximating functions that are smooth, converging geometrically for analytic functions [1]. Fast Fourier transforms (FFTs) can be used to map function values $f(x_j)$ to coefficients λ_j, and vice versa, in $O(N \log N)$ operations. Figure 1 presents the polynomial representation of the Bessel function J_0 and its corresponding Chebyshev coefficients. The construction process begins by sampling the target function at $2^n + 1$ points, with $n = 3, 4, \ldots$. The optimal degree N is then determined such that $|\lambda_j|$ is close to zero, relative to the coefficient of largest magnitude, for all $j > N$.

The left graph of Fig. 1 was obtained with the following commands:

```
f = chebfun(@(x) besselj(0,x),[0 100]); plot(f,'.-')
```

and the coefficients were plotted using

```
c = chebpoly(f); semilogy(flipud(abs(c)),'.')
```

The execution of the first command constructs the chebfun object from an anonymous function evaluated in the specified interval. Once a chebfun object has been created, there are a number of methods that can be used to operate on it. The list of methods can be obtained by typing `methods chebfun`. The syntax is, in most cases, the same as the usual MATLAB calls for vectors. The integral of f from 0 to 100, for instance, is obtained with the command `sum`.

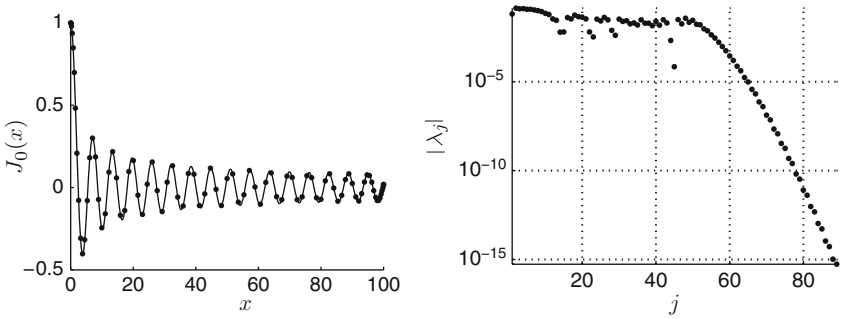

Fig. 1. *Left*: Chebfun representation by a polynomial of degree 88 of the Bessel function J_0 on the interval $[0, 100]$. The dots mark the 89 Chebyshev interpolation points. *Right*: semilog plot of the magnitude of the corresponding Chebyshev coefficients

```
>> sum(f)
ans = 0.922662556960163
```

All digits in this answer are correct except the last one. Integrals are computed efficiently by Clenshaw–Curtis quadrature in $O(N)$ operations once the coefficients are obtained with the aid of the FFT. Similarly, `cumsum(f)` returns the indefinite integral of the chebfun `f`.

Rootfinding plays a key role in the chebfun system. The method we use makes use of a recursion proposed by Boyd [5]. The main idea behind this approach is that the roots of a polynomial of the form (1) are the eigenvalues of an $N \times N$ *colleague matrix* [6]. To avoid the cubic growth of the number of operations required by eigenvalue computations, the algorithm uses recursive subdivision of intervals to bring the degree of the polynomial representation to at most 100, improving the overall operation count to $O(N^2)$.

Here is an example where rootfinding is used to obtain all local maxima of `f`.

```
df = diff(f);
xcrit = roots(df);
ddf = diff(df);
xmax = xcrit(ddf(xcrit)<0);
plot(f), hold on, plot(xmax,f(xmax),'o')
```

The result is displayed in Fig. 2. Also shown in this figure is the global minimum, which is computed in a similar way with just one function call: `[ymin,xmin]=min(f); plot(xmin,ymin,'*')`.

The evaluation of a chebfun at arbitrary points is carried out using the barycentric formula introduced by Salzer [7, 8]. The formula has been proved to be stable by Higham in [9] and requires $O(MN)$ operations to evaluate a chebfun at M points. The `plot` command, for instance, relies on evaluations at thousands of points.

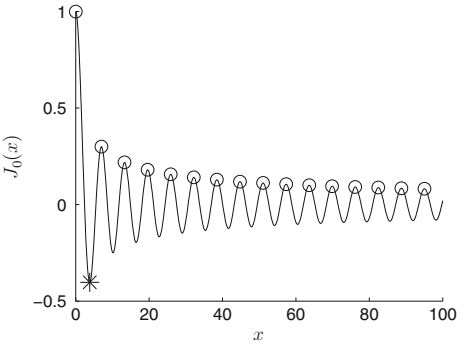

Fig. 2. Local maxima (*opencircle*) and global minimum (*asterisk*) of J_0 in [0,100]

2.2 Piecewise Representations

The chebfun system also handles piecewise smooth functions [2]. Piecewise representations can result from certain operations on smooth functions such as

```
abs, sign, floor, ceil, round, fix, min, max
```

among others. They may also be defined using the chebfun constructor. In the construction precess, each smooth piece may be explicitly defined or obtained through an edge detection procedure.

The main components of a chebfun with several pieces are the endpoints of the interval, the breakpoints, and the corresponding *funs*, which are objects representing each smooth piece. When breakpoints are introduced by operations on chebfuns, they are, in most cases, obtained by rootfinding. In the following code segment, for instance,

```
>> x = chebfun(@(x) x);
>> f = sin(4*x.^2).*floor(1.5*sin(5*x));
>> norm(f,1)
ans = 0.936713707137759
```

zerofinding is used twice. To find the breakpoints of the piecewise constant chebfun `floor(1.5*sin(5*x))`, the system finds all values of x that satisfy $1.5\sin(5x) - n = 0$ for $n = -1, 0, 1$. To compute the L_1 norm of f, it first obtains a piecewise representation of $|f|$, which also requires rootfinding.

Chebfun also comes with an efficient edge detector, since in many situations, one may want to construct a representation from samples of a function. To this end, the constructor works in two splitting modes that may be selected by the user: `splitting on` and `splitting off` – the current default is `on`. The following example illustrates the edge detector in action:

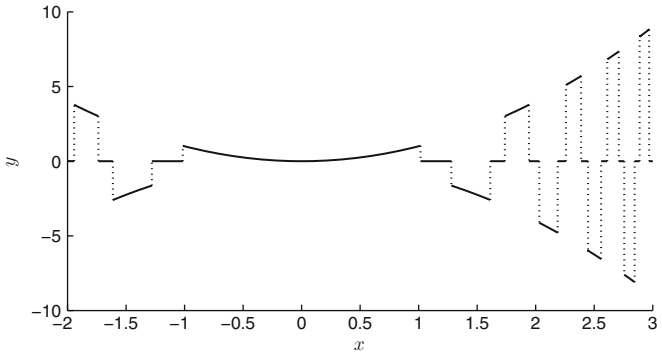

Fig. 3. Plot of the chebfun corresponding to $f(x) = x^2 \,\mathrm{round}(\cos x^3)$

```
splitting on
f = chebfun(@(x) x.^2.*round(cos(x.^3)),[-2 3]);
plot(f)
```

The result is shown in Fig. 3. The breakpoints are stored in the field `f.ends`. The edge detection algorithm uses bisection and finite differences to locate jumps in function values accurately to machine precision, as well as jumps in first, second and third derivatives [2].

In `splitting off` mode, the system disables the splitting algorithm. This mode is recommended when the target functions are smooth since in such cases manipulating global approximations is often more efficient. Most operations in the chebop system are restricted to this mode.

The Logistic Map

Simple examples of piecewise smooth functions arise throughout applied mathematics and are easily manipulated in the chebfun system. For one set of examples, see the online chebfun guide [10]. Here, we shall push the system harder with a more challenging example. Many chebfun computations finish in a fraction of a second; the results we shall show have taken minutes.

We use the logistic map to illustrate some strengths and limitations of piecewise polynomial representations. The map is given by the recurrence formula

$$x_{k+1} = r x_k (1 - x_k), \tag{2}$$

with $x_k \in [0, 1]$ and $r \in [0, 4]$, and is often used to model simple population dynamics and to illustrate key properties of dynamical systems such as chaos. The bifurcation diagram for the logistic map is shown in Fig. 4.

Suppose we are interested in representing the map functions, $g_r^k : x_0 \mapsto x_k$, and studying their convergence. For $r = 4$, it is possible to derive a simple polynomial representation [11],

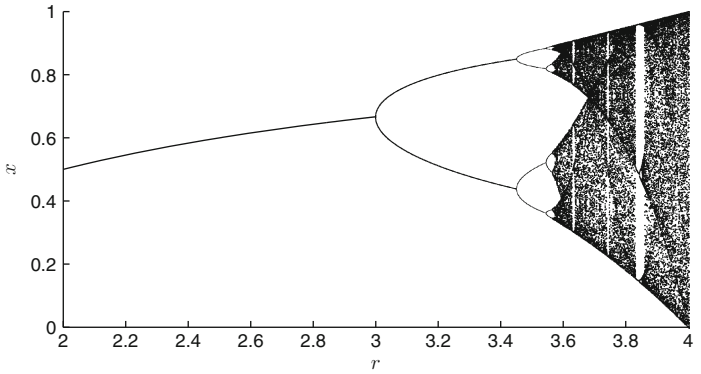

Fig. 4. The famous bifurcation diagram for the logistic map, showing period doubling as a route to chaos

$$g_4^k(x_0) = \frac{1 - \cos(2^k \arccos(1 - 2x_0))}{2},$$

but in general, nonrecursive expressions are not available. The maps g_r^k are polynomials of order 2^k, but their chebfun representations often have smaller degrees. For $1 < r < 3$, the functions g_r^k converge to a constant, $1 - 1/r$, if we exclude the singular endpoints: $x = 0$ and $x = 1$. Here are the degrees of chebfuns for $g_{2.5}^k$:

```
>> xk = chebfun(@(x) x, [0.001 .999]);
>> for k = 1:51
       xk = 2.5*xk.*(1-xk); deg(k) = length(xk)-1;
   end
>> deg
deg =
  Columns 1 through 9
     2     4     8    16    32    64   112   178   284
  Columns 10 through 18
   434   574   544   554   522   522   522   496   488
  Columns 19 through 27
   488   474   470   390   390   388   388   354   352
  Columns 28 through 36
   352   352   338   330   330   258   258   256   256
  Columns 37 through 45
   158   158   158   158   158   106   106   106   106
  Columns 46 through 51
   106    72    72     0     0     0
```

Despite the initial exponential growth in degree, the length of the chebfuns reaches a maximum of 574, and for $k \geq 49$, the chebfun representations of $g_{2.5}^k$

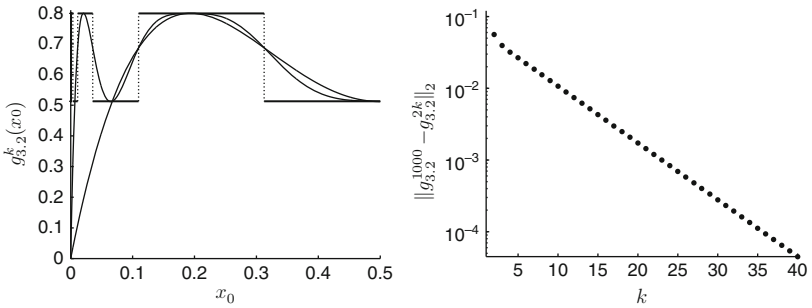

Fig. 5. *Left*: piecewise chebfun representation of $g^k_{3.2}$, $k = 2, 4$ and 1,000. *Right*: convergence plot of $g^{2k}_{3.2}$ in the L_2-norm

are constant functions with the correct value 0.6. In the exact arithmetic of symbolic computing, for $k = 49$ the degree would be 562,949,953,421,312.

It is also interesting to look at the convergence of these functions in the two-cycle region, $3 < r < 3.44\ldots$, where the subsequences $\{g_r^{2k-1}\}$ and $\{g_r^{2k}\}$ converge to piecewise constant functions. With the aid of the chebfun automatic edge detection algorithm, we can represent these limiting functions and compute the rates for convergence as follows for $r = 3.2$:

```
g1000 = chebfun(@(x) logistic(3.2,1000,x), [0 0.5]);
xk = chebfun(@(x) x, domain(g1000));
delta = zeros(40,1);
for k = 1:80
   xk = 3.2*xk.*(1-xk);
   if mod(k,2)==0, delta(k/2) = norm(xk-g1000); end
end
plot(g1000), figure, semilogy(delta,'.')
```

The result is shown in Fig. 5 together with the graphs of $g^2_{3.2}$ and $g^4_{3.2}$. The first line of the execution above requires the function `logistic.m`:

```
function x = logistic(r,n,x)
     for k=1:n, x = r*x.*(1-x); end
```

Notice that the functions g_r^k are symmetric about $x = 0.5$, so in the example above we only considered the interval $[0, .5]$. The subsequences $\{g_r^{2k-1}\}$ and $\{g_r^{2k}\}$ cannot converge uniformly because of the jumps in the limit. In the (default) L_2-norm, on the other hand, they converge very fast. The right plot in Fig. 5 indicates exponential convergence. We point out that the chebfun representation of $g^{1000}_{3.2}$ has 31 break points, most of them near $x = 0$, with the spaces between them decaying exponentially.

A similar cascade of break points can be observed in the 4-cycle region. In fact, as the parameter r is increased, the number of jump locations also

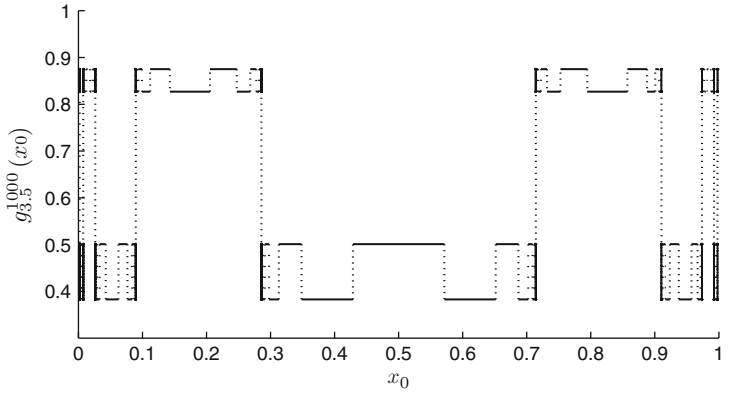

Fig. 6. Piecewise chebfun representation of $g_{3.5}^{1000}$. Now there are four constant values instead of two

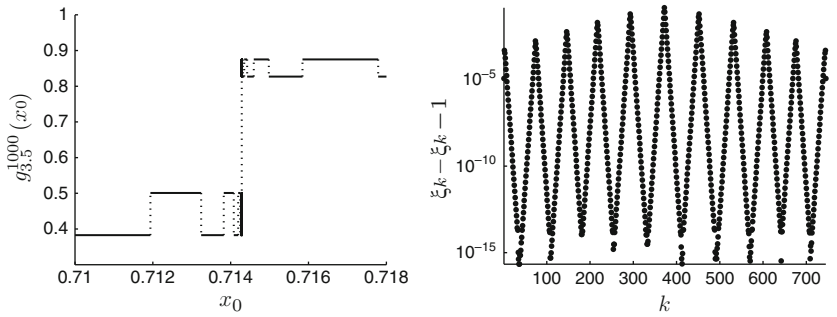

Fig. 7. *Left*: plot of $g_{3.5}^{1000}$ near the unstable fixed point $x = 1 - 1/3.5$. *Right*: semilog plot of the distance between breakpoints of $g_{3.5}^{1000}$

increases. Figure 6 shows the chebfun representation for $r = 3.5$ and $k = 1{,}000$, `g1000 = chebfun(@(x) logistic(3.5,1000,x), [0.001 0.999])`. Notice that now there are several clusters of break points. A detailed plot of $g_{3.5}^{1000}$ around the unstable fixed point $x = 1 - 1/3.5 = 0.714285\ldots$ is presented in Fig. 7. The semilog plot on the right of this figure shows that the distance between neighboring break points decreases exponentially near some critical values. In this plot, ξ_k denotes jump locations which were recovered from the field `g1000.ends`. This graph was generated with `semilogy(diff(g1000.ends))`.

Because these subsequences of polynomials are converging to piecewise constant functions, the pointwise convergence is slower near the location of a

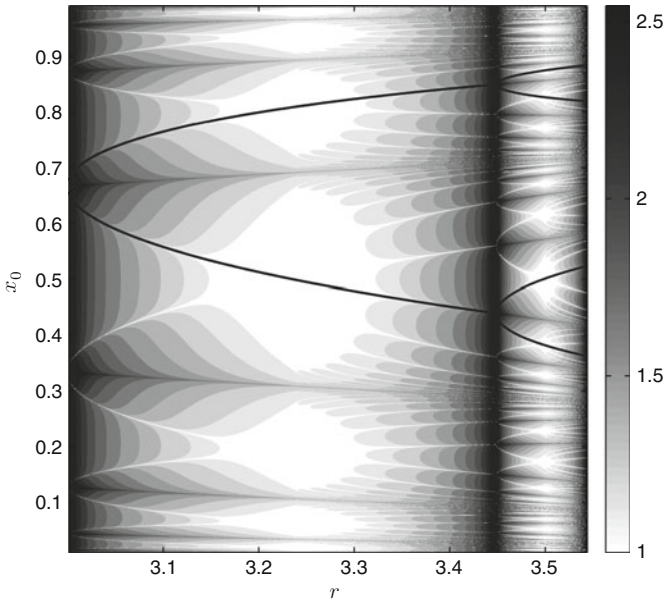

Fig. 8. Number of iterations needed to converge (to a tolerance of 10^{-5}) to the 2-cycle and 4-cycle limits as a function of x_0 and r. The grayscale map shows the $\log_{10}$ of the number of iterations. The bifurcation diagram is superimposed (*solid lines*). Figures 5 and 6 correspond to vertical sections through this plot at $r = 3.2$ and $r = 3.5$, respectively

jump in the limit function. Figure 8 shows a grayscale map of the logarithm of the number of iterations required for a subsequence $\{g_r^k(x_0)\}$ to converge to its limit, to a tolerance of 10^{-5}. This figure is not the result of a chebfun computation; it is provided to give insight into the convergence of chebfun computations. Notice that near bifurcation points, convergence is very slow, regardless of the starting value. Away from these regions, convergence is fast almost everywhere. The locations of slow convergence in this case seem to coincide with the jump locations in the limiting function. Similar convergence maps have been presented in [12].

Finally, the logistic map can also be used to illustrate some limitations of piecewise polynomial representations. Near bifurcation points, for instance, chebfun representations of the maps g_r^k can only be obtained for very small values of k, since the degree of the representations grows exponentially with k and the limit is not achieved in thousands of iterations. Similarly, near or

at the chaotic regimes, the maps are impossible to represent for large k due to the complexity of these functions.

3 Chebops

The chebop system developed by Driscoll et al. [3] is an extension of the chebfun system to handle linear operators. Here, the analogy is between matrices and continuous operators rather than vectors and functions.

A chebop object is defined by a domain, a chebfun, or another chebop. Identity, differentiation and integration operators, for instance, are defined using the domain class:

```
[d,x] = domain(0,1);
D = diff(d)     % differentiation
I = eye(d)      % identity
S = cumsum(d)   % integration
```

We point out that `domain` returns a domain object and a chebfun, in this case `x`. The multiplication operator, on the other hand, is defined by a chebfun and the exponential operator by a chebop. These operators can then be combined to generate other chebops. For example, `L = D^2 + 5*I` defines the operator $L : u \mapsto \partial^2 u/\partial x^2 + 5u$.

In chebops, multiplication has been overloaded to apply operators to chebfuns and other chebops. This can be illustrated as follows:

```
u = sin(3*pi*x)
f = L*u
```

Now, suppose that we would like to solve the differential equation $Lu = f$ for u. Of course, the backward operation requires boundary conditions for uniqueness. For example, if the desired boundary conditions are homogeneous Dirichlet at $x = 0$ and Neumann at $x = 1$, we augment L with

```
L.lbc = 'dirichlet'  % left boundary condition
L.rbc = 'neumann'    % right boundary condition
```

and the solution of the differential equation can then be obtained using the backslash command, which has been overloaded to invert chebops:

```
sol = L\f
```

The algorithms used in the chebop system are described in [3]. When inverting these operators, as in the solution of differential equations, chebops rely on adaptive spectral collocation methods that are also based on Chebyshev polynomials [13,14]. Lazy evaluations of the associated spectral discretization matrices are performed to compute the solution. As in the chebfun system, the polynomial degree of the solution of a differential equation is determined

by the relative magnitudes of Chebyshev coefficients. In the present implementation, most chebops operations are restricted to global representations, i.e., to `splitting off` mode.

We give a number of examples that illustrate the use of chebops in the solution of linear and nonlinear ODEs, PDEs, and eigenvalue problems. The codes used to solve each problem are provided here, and more examples can be downloaded from the chebfun website [10].

3.1 Linear Differential Equations with Variable Coefficients

Consider the hypergeometric equation

$$xy'' + (5 - x)y' + y = \sin(5x), \quad x \in (1, 6), \tag{3}$$

subject to homogeneous Neumann boundary conditions. The chebop syntax to obtain a solution is

```
[d, x] = domain([1 6]);
D = diff(d);
L = diag(x)*D^2 + diag(5-x)*D + eye(d) & 'neumann';
u = L\sin(5*x);
plot(u)
```

Here `diag` is used to define the multiplication operator and `&` to define the boundary conditions. When this code is executed, the system adaptively determines that the desired solution can be represented to approximately machine precision by a polynomial of degree 47. The plot is shown in the left of Fig. 9. The maximum value of the residual in this calculation is

```
>> norm(L*u-sin(5*x),inf)
ans = 3.925115787950517e-11
```

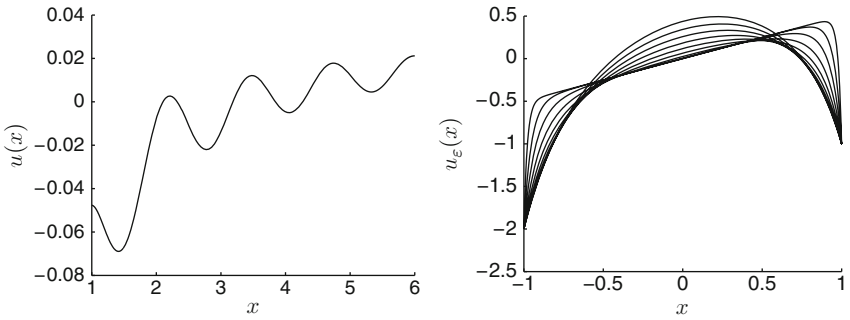

Fig. 9. Chebop solution of two boundary value problems. *Left*: the hypergeometric equation (3). *Right*: the boundary layer problem (4) with $\varepsilon = 0.02, 0.04, \ldots, 0.2$

Our next example is the singularly perturbed problem [15],

$$\varepsilon y'' - xy' + y = 0, \quad x \in (-1, 1), \quad y(-1) = -2, \quad y(1) = -1. \tag{4}$$

Chebops handle boundary layers well, as the clustering of Chebyshev nodes provide good resolution near the endpoints of the interval. The following commands generate plots for several values of ε:

```
figure, hold on
[d,x] = domain(-1,1);
D = diff(d);
for ep = 0.02:0.02:0.2
    L = ep*D^2-diag(x)*D+eye(d);
    L.lbc = -2; L.rbc = -1;
    plot(L\0)
end
```

The solutions correspond to polynomials of degrees 64, 50, 42, 38, 36, 34, 34, 28, 28, 28, and are presented on the right of Fig. 9.

3.2 The Orr–Sommerfeld Eigenvalue Problem

The chebop system also overloads the command `eigs` to solve eigenvalue problems. The eigenvalues of the 1D Laplacian operator on $[0, \pi]$, for instance, can be easily computed with

```
>> d = domain(0,pi);
>> L = -diff(d,2) & 'dirichlet';
>> eigs(L,6)
ans =
   0.999999999999991
   3.999999999999823
   8.999999999999659
  15.999999999999831
  25.000000000000089
  35.999999999999893
```

The command `eigs` has been overloaded instead of `eig` because, in MATLAB, the latter is used to return all eigenvalues of a matrix, which is not possible for differential operators. The details of which eigenvalues are returned by `eigs` can be found in [3].

Our next example is an Orr–Sommerfeld generalized eigenvalue problem arising in the eigenvalue stability analysis of plane Poiseuille fluid flow. The Orr–Sommerfeld equation is given by

$$\frac{d^4u}{dx^4} - 2\alpha^2\frac{d^2u}{dx^2} + \alpha^4 u - i\alpha R\left[(1-x^2)\left(\frac{d^2u}{dx^2} - \alpha^2 u\right) - 2u\right] = \lambda\left(\frac{d^2u}{dx^2} - \alpha^2 u\right)$$

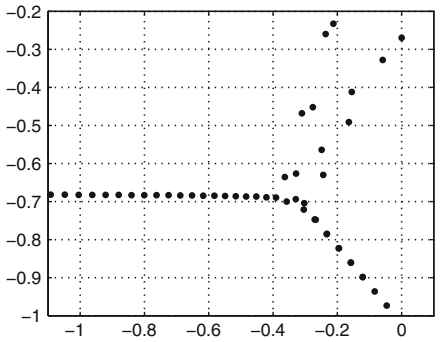

Fig. 10. Rightmost eigenvalues of the Orr–Sommerfeld operator in the complex plane for $R = 5772.22$ and $\alpha = 1.02056$

where R is the Reynolds number and α a wave number. Orszag showed in [16] that $R = 5772.22$, $\alpha = 1.02056$ are critical values, with one of the eigenvalues crossing to the right half of the complex plane. We repeat his eigenvalue computation using chebops.

```
[d,x] = domain(-1,1);
I = eye(d); D = diff(d);
R = 5772.22; alpha = 1.02056;
B = D^2 - alpha^2;
A = B^2/R - 1i*alpha*(2+diag(1-x.^2)*B);
A.lbc(1) = I; A.lbc(2) = D;
A.rbc(1) = I; A.rbc(2) = D;
e = eigs(A,B,50,'LR');
```

We confirm Orszag's result by showing these eigenvalues in Fig. 10 and computing their largest real part:

```
>> max(real(e))
ans = 6.129513257887425e-09
```

3.3 Linear Partial Differential Equations

Certain linear partial differential equations can also be handled by chebops. In the following example, we use the exponential operator `expm` to advance in time. Writing a linear partial differential equation in the form $u_t = Lu$, we have $u(t + \Delta t, x) = \exp(\Delta t L)u(t, x)$, assuming that $\exp(\Delta t L)$ is well defined. The following code solves the convection-diffusion equation,

$$u_t = 0.05u_{xx} - xu_x, \qquad x \in (-2, 2), \tag{5}$$

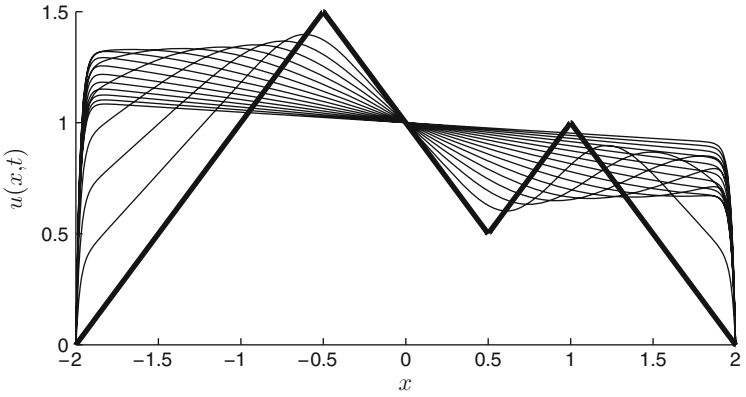

Fig. 11. Solution to the PDE (5) at several times t. The initial condition is shown by a *thick line*

with homogeneous Dirichlet boundary conditions and initial condition

$u(0, x) = -|x + 0.5| + |x - 0.5| - |x - 1| + 2.$

```
[d,x] = domain(-2,2);
splitting on
u = chebfun(@(x) -abs(x+0.5)+abs(x-0.5)-abs(x-1)+2, d);
splitting off
L = 0.05*diff(d,2)-diag(x)*diff(d);
dt = 0.2; expmL = expm(dt*L & 'dirichlet');
plot(u,'k', 'linewidth',4), hold on
for t = 0:dt:3
    u = expmL*u;
    plot(u,'k')
end
```

The result of this execution is presented in Fig. 11. Notice that despite the lack of smoothness in the initial condition, chebops can be used in the solution of this problem as u is smooth for all $t > 0$.

Chebops can also be used to solve nonlinear PDEs with implicit or semi-implicit time-stepping schemes. One example, involving the nonlinear cubic Schrödinger equation, is presented in [3].

3.4 Nonlinear Boundary-Value Problems

While linear equations can be solved with "\", nonlinear problems require iterative algorithms.[1] In our next example we use Newton's method together with chebop technology to solve the boundary-value problem

[1]Solutions to nonlinear boundary value problems have been automated in Chebfun Version 3 via automatic differentiation. The example in this section can now be solved with "\".

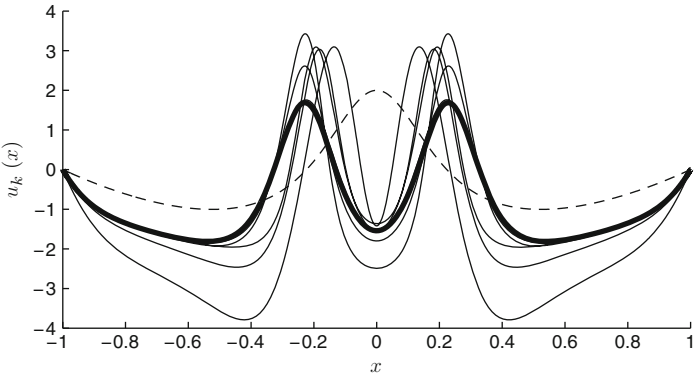

Fig. 12. Newton's method solution of (6). Intermediate iterates u_k are shown together with the initial guess (*dashed*) and the final solution (*thick line*) – cf. Fig. 9.26 in [17]

$$\varepsilon u'' + 2(1-x^2)u + u^2 = 1, \quad x \in (-1,1), \tag{6}$$

with homogeneous Dirichlet boundary conditions. This equation, due to Carrier, is discussed at length by Bender and Orszag [17]. The problem has many solutions, some of which can be approximated by boundary-layer theory. The following code was used to generate the solution plotted in Fig. 12. The figure also shows the intermediate Newton method iterates.

```
[d,x] = domain(-1,1);
D2 = diff(d,2); F = diag(2*(1-x.^2));
u = 2*(x.^2-1).*(1-2./(1+20*x.^2));
eps = 0.01; nrmdu = Inf;
plot(u,'--k'), hold on
while nrmdu > 1e-10
    r = eps*D2*u + F*u + u.^2 - 1;
    A = eps*D2 + F + diag(2*u) & 'dirichlet';
    A.scale = norm(u); delta = -(A\r);
    u = u+delta;  nrmdu = norm(delta)
    plot(u,'k')
end
plot(u,'k', 'linewidth',4)
```

3.5 Ground State Solution of the 3D Cubic Schrödinger Equation

Our final example, which comes to us from Roudenko and Holmer [20], is related to radial solutions of the cubic Schrödinger equation in $\mathbb{R}^3$,

$$iu_t + \Delta u + |u|^2 u = 0.$$

Using the separation of variables $u(x,t) = e^{it}v(x)$, we obtain a nonlinear equation for v,

$$-v + \Delta v + |v|^2 v = 0. \tag{7}$$

This equation has an infinite number of solutions in $H^1(\mathbb{R}^3)$. The solution of minimal mass is positive, radial, and exponentially decaying and is called *the ground state* [18].

We shall seek a positive radial solution to (7) with exponential decay. Because the current implementation of the chebfun and chebop systems is restricted to bounded domains, we perform the change of variables $r = \tilde{r}/(1-\tilde{r})$, $\tilde{r} \in [0,1]$, and $Q(\tilde{r}) = v(\tilde{r}/(1-\tilde{r}))$. An equation for Q can then be written as

$$\tilde{r}\left[-Q + (1-\tilde{r})^4 Q_{\tilde{r}\tilde{r}} + Q^3\right] + 2(1-\tilde{r})^4 Q_{\tilde{r}} = 0, \qquad \tilde{r} \in (0,1), \tag{8}$$

with boundary conditions $Q_{\tilde{r}}(0) = Q(1) = 0$. As in the previous example, we use Newton's method to find a solution.

```
[d,r] = domain(0,1); D = diff(d); D2 = D^2;
Q = chebfun(@(r) 4*sech(2*r./(1-r+eps)), d);
nrmdu = Inf;
while nrmdu > 1e-13
   res = r.*(Q.^3-Q+(1-r).^4.*(D2*Q)) + 2*(1-r).^4.*(D*Q);
   A = diag(r)*(diag(3*Q.^2)-eye(d)+diag((1-r).^4)*D2)+ ...
       2*diag((1-r).^4)*D;
   A.rbc = 'dirichlet' ; A.lbc = 'neumann';
   A.scale = norm(Q);  delta = -(A\res);
   Q = Q+delta;  nrmdu = norm(delta)
end
plot(Q)
```

The resulting plot is shown in Fig. 13.

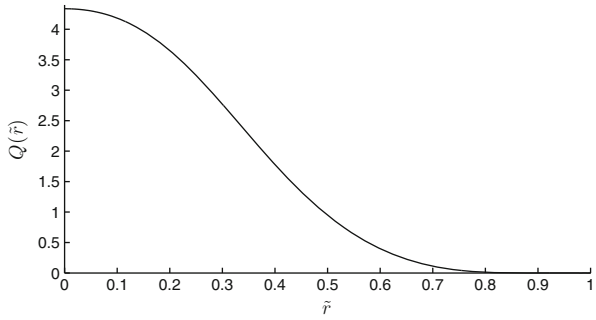

Fig. 13. The solution of (8) computed with Newton's method

4 Concluding Remarks

A brief review of the chebfun and chebop systems has been presented and several examples provided to demonstrate how simple and effective the system is. Some capabilities of the software have not been mentioned here, such as quasimatrices [19]. The system is evolving and efforts are currently being made to extend it to handle unbounded domains via mapped polynomial representations. We hope that the change of variables performed in the final example may be handled automatically in future releases.

The computations presented in this paper were carried out with the October 2008 release of chebfun Version 2. The code is freely available under a BSD-type software license, and can be found together with a user's guide and other information at http://www.maths.ox.ac.uk/chebfun/.

Acknowledgments

The chebfun system is currently a joint project with Ricardo Pachón and Toby Driscoll. Toby Driscoll is the principal author of the chebop system, to which another key contributor was Folkmar Bornemann.

References

1. Battles, Z., Trefethen, L.N.: An extension of MATLAB to continuous functions and operators. SIAM J. Sci. Comput. **25**(5), 1743–1770 (2004)
2. Pachón, R., Platte, R.B., Trefethen, L.N.: Piecewise-smooth chebfuns. IMA J. Numer. Anal. doi:10.1093/imanum/drp008 (2009)
3. Driscoll, T.A., Bornemann, F., Trefethen, L.N.: The chebop system for automatic solution of differential equations. BIT Numer. Math. **48**(4), 701–723 (2008)
4. Trefethen, L.N.: Computing numerically with functions instead of numbers. Math. Comput. Sci. **1**(1), 9–19 (2007)
5. Boyd, J.P.: Computing zeros on a real interval through Chebyshev expansion and polynomial rootfinding. SIAM J. Numer. Anal. **40**(5), 1666–1682 (2002)
6. Good, I.J.: The colleague matrix, a Chebyshev analogue of the companion matrix. Quart. J. Math. **12**, 61–68 (1961)
7. Berrut, J.-P., Trefethen, L.N.: Barycentric Lagrange interpolation. SIAM Rev. **46**(3), 501–517 (2004)
8. Salzer, H.E.: Lagrangian interpolation at the Chebyshev points $X_{n,\nu} \equiv \cos(\nu\pi/n), \nu = 0(1)n$; some unnoted advantages. Comput. J. **15**, 156–159 (1972)
9. Higham, N.J.: The numerical stability of barycentric Lagrange interpolation. IMA J. Numer. Anal. **24**(4), 547–556 (2004)
10. Trefethen, L.N., Pachón, R., Platte, R.B., Driscoll, T.A.: Chebfun version 2. http://www.maths.ox.ac.uk/chebfun/ (2008)
11. Sprott, J.C.: Chaos and Time-Series Analysis. Oxford University Press, New York (2003)

12. Bresten, C.L., Jung, J.-H.: A study on the numerical convergence of the discrete logistic map. Commun. Nonlinear Sci. **14**(7), 3076–3088 (2009)
13. Fornberg, B.: A Practical Guide to Pseudospectral Methods. Cambridge University Press, Cambridge (1996)
14. Trefethen, L.N.: Spectral Methods in MATLAB. SIAM, Philadelphia, PA (2000)
15. O'Malley, R. Jr.: Singularly perturbed linear two-point boundary value problems. SIAM Rev. **50**(3), 459–482 (2008)
16. Orszag, S. A.: Accurate solution of the Orr-Sommerfeld stability equation. J. Fluid Mech. **50**, 689–703 (1971)
17. Bender, C.M., Orszag, S.A.: Advanced Mathematical Methods for Scientists and Engineers. I. Springer, New York (1999). Reprint of the 1978 original
18. Weinstein, M.I.: Nonlinear Schrödinger equations and sharp interpolation estimates. Comm. Math. Phys. **87**(4), 567–576 (1982/1983).
19. Trefethen, L.N.: Householder triangularization of a quasimatrix. IMA J. Numer. Anal. doi:10.1093/imanum/drp018 (2009)
20. Holmer, J., Roudenko, S.: A sharp condition for scattering of the radial 3D cubic nonlinear Schrödinger equation. Commun. Math. Phys. **282**(2), 435–467 (2008)