Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computational Physics 207 (2005) 173–191

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

Fast algorithms for spectral collocation with non-periodic boundary conditions

W. Lyons [a,*,1], H.D. Ceniceros [a,2], S. Chandrasekaran [b,1], M. Gu [c]

[a] *Department of Mathematics, University of California, Santa Barbara, CA 93106-3080, USA*
[b] *Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106-9560, USA*
[c] *Department of Mathematics, University of California, Berkeley, CA 94720-3840, USA*

Received 24 November 2004; accepted 13 January 2005
Available online 16 February 2005

Abstract

We present a method for the numerical solution of partial differential equations using spectral collocation. By employing a structured representation of linear operators we are able to use fast algorithms without being restricted to periodic boundary conditions. The underlying ideas are introduced and developed in the context of linearly implicit methods for stiff equations. We show how different boundary conditions may be applied and illustrate the technique on the Allen–Cahn equation and the diffusion equation.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Pseudospectral methods; Chebyshev collocation; Fast algorithms; Fast direct solver; Semi-separable; Boundary conditions

1. Introduction

An important factor in the rise in popularity of spectral methods is the availability of fast transform methods. Due to the global nature of the basis functions employed, matrix representations of differential operators in the spatial domain are not sparse and are costly to work with. Working in a transformed domain returns us to a sparse representation.

In a number of situations, staying in the spatial domain may be highly desirable. The most common reason for favoring a transform-free method is to facilitate enforcing boundary conditions. If we wish to work

[*] Corresponding author.
E-mail address: lyons@math.ucsb.edu (W. Lyons).
[1] Partially supported by NSF grant number CCR-0204388.
[2] Partially supported by NSF grant number DMS-0311911.

0021-9991/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2005.01.007

on a non-periodic problem and specify Dirichlet or Neumann boundary conditions, it is natural to work in physical space. However, if we do so our Chebyshev derivative operator will be a full matrix and the size of problem we can tackle will be severely reduced.

In this paper, we introduce a new method for efficiently solving spectral discretizations of partial differential equations (PDEs) while staying in the spatial domain. We will develop our ideas in the context of linearly implicit (LI) methods. This approach is currently the most popular choice for PDEs with low-order non-linear terms and higher-order linear terms.

A recent paper by Kassam and Trefethen [9] includes a survey of methods being applied to treat problems of this kind and indicates that LI methods may not always be the best choice. As such, we also discuss how the ideas we propose here can be used to similar advantage in integrating factor (IF) and exponential time differencing (ETD) methods. These methods are of recent vintage but show great promise.

Linearly implicit (also known as implicit–explicit or IMEX) methods are widely used and have a history going back at least to 1980. In these early papers we find a full description of the method [15] and some results on stability [33]. More recent treatments include [2,3]. The defining feature of LI methods is that they employ an implicit discretization of leading order *linear* terms and an explicit discretization of the remaining, typically *non-linear* terms. Thus, if the inversion of the resulting linear system can be accomplished at a low cost then one obtains an efficient method in which the leading order timestepping stability constraint has been eliminated. Unfortunately, the solution to such linear systems is costly for Chebyshev collocation and quite generally for spectral discretizations when remaining in the spatial domain.

Here, by employing an appropriate representation of the differential operator on the spatial domain, we can use an implicit time discretization of the leading order linear operators and apply direct, fast, non-iterative methods to solve the resulting linear system at each timestep. This allows us to remove the highest order timestepping constraint while retaining nearly linear scaling in the number of collocation points.

We illustrate this strategy in the particular case of functions on a finite interval, discretized by collocation on the Gauss–Lobatto points.

2. Basic strategy

The basis of our approach is a representation of differential operators. In transform methods no explicit representation of the derivative operator is needed, as the coefficients of the derivative are generated through recursion. In the spatial domain, the derivative has a matrix representation and classically this is what is used in physical space numerical methods.

However, the matrix representation of the derivative is not sparse, symmetric or even normal (see [17]). It is computationally expensive to work with such matrices. The matrix of the derivative is also ill-conditioned, so even a brute force solution of a discretized differential equation using *iterative* methods will be very expensive.

In our approach we represent the Chebyshev derivative operator not with a matrix, but with a *hierarchically semi-separable representation* (also known as a *rebus*). This is a representation closely related to the fast multipole method (FMM) in the form developed by Starr and Rokhlin [31] and by Yarvin and Rokhlin [34]. Similar developments along these lines have since been made by Beylkin, Coult and Mohlenkamp in [7] and Beylkin and Sandberg in [10]. The formulation we adopt here was introduced by Chandrasekaran and Gu in [13]. We place this representation in historical context in Section 3 and proceed to review the definition and properties of the representation below in Section 4.

Our approach to solving the PDEs is as follows. Starting from the governing equations, we discretize the time dependence using an implicit discretization of the leading order linear terms. This avoids high stability restrictions on the allowable time step. Then, we introduce the rebus representation of the spectral derivative operator and use fast algorithms to generate the needed higher order linear operator for a timestep.

After this, the linear system that results from the implicit or semi-implicit time discretization and the application of the boundary conditions is solved in rebus form using specialized fast algorithms.

3. Relations to earlier work

As noted by Golub and Van Loan [19] good matrix algorithms rely on exploiting structure. Although the methods being presented here are new, the structure they exploit has been noted before. A number of representations and algorithms have arisen to take advantage of the same properties.

We will briefly compare and contrast the current approach with others which rely on the same principles. We will try to emphasize the differences between each approach and the one advocated here.

3.1. Tree codes

For N-body problems of the type encountered in astrophysics, computational chemistry, plasma physics and other fields relying on particle simulations, the cost of evaluating the mutual interactions grows as N^2 and realistic problems are frequently intractable.

In response to this, a number of procedures were created to reduce the burden of these calculations. Notable amongst these were the Barnes–Hut tree code [5], Appel's method [1] and particle-in-cell methods and their descendants, the particle–particle, particle–mesh methods [22].

The underpinning of these methods was the observation that although short range interactions can be arbitrarily complex, the long range effect of a cluster of particles will be relatively smooth. An alternative viewpoint is that the matrix of the interaction will have low rank away from the diagonal or will contain off-diagonal blocks that can be accurately approximated by low rank matrices.

Structured low rank representations lie behind the tree codes and are rediscovered periodically. They are methods for fast summation and all of the applications rely on the fast evaluation of matrix–vector products. In N-body particle interaction, we are summing the contributions to the potential from N sources and the application is direct.

An extension of fast summation is the solution of linear systems. By using conjugate gradient or other iterative methods, the rate-determining step in the solution of a linear system becomes the rapid evaluation of matrix–vector products. Thus tree codes may be used to solve linear systems, notably those arising from PDEs.

3.2. Fast multipole methods

The FMM was originally introduced to solve integral equations [28]. A different formulation was used as a fast summation method [20] and it is this algorithm that has had the most impact and is generally known as the FMM.

The work of Rokhlin and collaborators on FMM structure resulted in a rich complex of ideas dealing with much more than fast summation. After the initial application to integral equations, it was applied to evaluating conformal mappings [27], two-point boundary value problems [31], ordinary differential equations [30], 2-dimensional integral equations in scattering theory [29], the wave equation [14] and Laplace's equation [23].

Of particular interest to us here is Rokhlin and Starr's work presented in [30,31]. Here, the authors use a recursive partitioning to solve an integral equation. The apparatus developed is analytic in nature and specific to integral equations. The development demonstrates that integral equations can be solved efficiently using a recursive partitioning. The off-diagonal blocks in this partitioning are shown to have low rank. However, unlike the partitioned low rank representations in Section 3.1, the representations in each block are not independent, but are related in a multilevel framework, where information may be projected or interpolated between fine and coarse scales.

The twin ideas of a partitioned SVD representation and a full FMM interaction tree arise again in Rokhlin and Yarvin's work [34].

These ideas are further developed by Beylkin, Coult and Mohlenkamp in [7], where the authors use a recursive block decomposition to enable them to work efficiently with spectral projection operators. This is the same block decomposition used in Rokhlin's work that also arises in rebus methods. They prove spectral projection operators to have a compact representation in terms of low rank blocks. Further, they develop an efficient algorithm to multiply together two matrices stored in this form. The algorithm used is similar to that employed for multiplying in non-standard wavelet form [18].

3.3. Rebus methods

Our motivation in developing rebus-based methods was that all of these applications of the FMM were exploiting the same underlying structure. Although the expansions and representations used in the FMM papers cited above were problem specific, the underlying ideas were consistent. By confining ourselves to a specific recursive block partitioning we further develop the ideas of the FMM to allow an efficient representation of differential and integral operators in general, as opposed to treating specific operators.

By this restriction to a specific FMM tree (which is what a rebus is), we lose the flexibility of the full FMM but it becomes possible to develop a full algebra. Thus we can implement a fast rebus–vector multiplication, rebus–rebus multiplication, perform LU factorization and code direct rebus solvers. The rebus structure is a hybrid, retaining enough FMM structure to design fast algorithms but simplifying matters enough to make algorithms algebraically tractable.

All the algorithms developed to operate on this structure share some desirable features. They are intrinsically parallel since calculations happen locally and are propagated in discrete stages through the tree. The algorithms are also intrinsically multiresolution. All have recognizable "upsweep" and "downsweep" recursions corresponding to interpolating or projecting information to a different scale. All are easily formulated as adaptive algorithms, where refinement is performed only locally and as needed. An example of this in the case of the solution of linear systems is found in Chandrasekaran, Gu and Lyons [12].

4. Rebus representation

The rebus, or hierarchically semi-separable representation of a matrix, was described in [13] and forms the basis for the numerical method being described. We give a brief review here to establish notation.

4.1. Definition

A rebus representation is based on a hierarchical block structure and may be described in terms of block partitioning. Given any dense matrix we consider it as a block 2×2 matrix where the blocks are of arbitrary size and in particular need not all be the same size.

We use the notational device of preceding the usual positional subscripts with the "level" of splitting, so the original dense matrix A can be thought of as $A_{0;11}$. That is, the (1,1) block of the matrix partitioned zero times. Using this notation and block partitioning we get

$$A_{0;11} = \begin{pmatrix} A_{1;11} & A_{1;12} \\ A_{1;21} & A_{1;22} \end{pmatrix}.$$

The off-diagonal blocks are factored and stored as

$$A_{1;ij} = U_{1;i}B_{1;ij}V^{\mathrm{H}}_{1;j} \quad \text{for } (i,j) = (1,2),(2,1),$$

where V^{H} denotes the Hermitian conjugate of V.

The notation here should be reminiscent of that conventionally used for the singular value decomposition (SVD), $A = U\Sigma V^{\mathrm{H}}$. The purpose of the factorization is to allow us to take advantage of rank deficient blocks. However, it is in fact *not* a simple SVD, as the global structure places constraints on which factors are allowable.

We repeat the above process for each of the diagonal blocks. That is, we subdivide $A_{1;11}$ and $A_{1;22}$ to get

$$A_{1;11} = \begin{pmatrix} A_{2;11} & A_{2;12} \\ A_{2;21} & A_{2;22} \end{pmatrix}$$

and

$$A_{1;22} = \begin{pmatrix} A_{2;33} & A_{2;34} \\ A_{2;43} & A_{2;44} \end{pmatrix}.$$

The off-diagonal blocks of these matrices are again factored in the form

$$A_{2;ij} = U_{2;i}B_{2;ij}V^{\mathrm{H}}_{2;j} \quad \text{for } (i,j) = (1,2),(2,1),(3,4),(4,3).$$

The new diagonal blocks are again split, and this process continues down to some lowest level where they are simply stored as dense matrices.

For an appropriate factorization, this will enable us to use a low-rank representation of the off-diagonal blocks. The resulting block structure is illustrated in Fig. 1. The benefit of this structure for representing differential operators is demonstrated in Section 4.2.

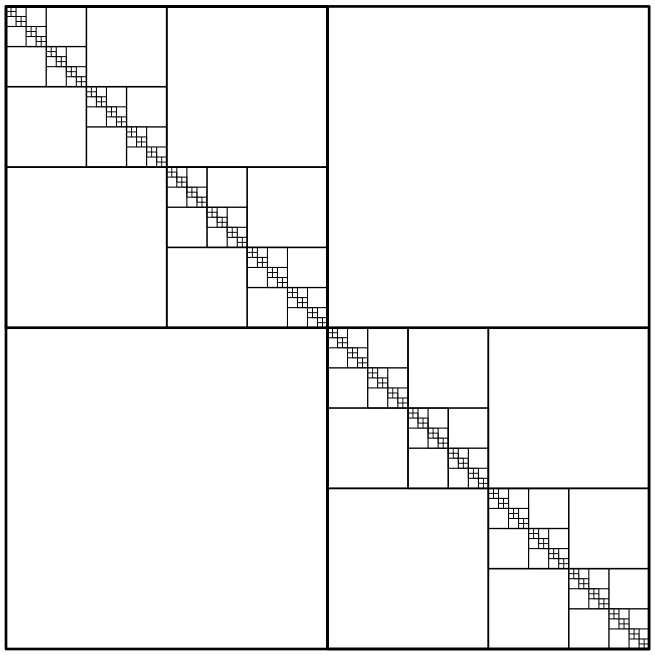

Fig. 1. Block structure used to generate rebus representation.

Together with the (low-rank) representations of the off-diagonal blocks, there is a tree structure as follows: we do not store the factors $U_{k;i}$ and $V_{k;i}$ at every level. Instead, we store them at some lowest level only, and then store intermediate quantities $R_{k,i}$ and $W_{k,i}$ which satisfy the relationships

$$U_{k-1;i} = \begin{pmatrix} U_{k;m}R_{k;m} \\ U_{k;n}R_{k;n} \end{pmatrix}, \qquad V^{\mathrm{H}}_{k-1;i} = \begin{pmatrix} W^{\mathrm{H}}_{k;m}V^{\mathrm{H}}_{k;m} & W^{\mathrm{H}}_{k;n}V^{\mathrm{H}}_{k;n} \end{pmatrix}, \tag{1}$$

where level $k-1$ is the parent of level k. This allows us to store only the "lowest level" factorizations and small transformations which relate them to the factors at the next highest level.

To ensure that this is possible, we must relate the factorizations at each level of the rebus. It is for this reason that we cannot simply take a SVD of each block at each level. The $U_{k;n}$ at the lowest level must not only provide a basis for the column space of $A_{k;nm}$, it must provide a basis for the column space of the whole row n of A_k (excluding the diagonal block).

Once we have performed this recursive splitting and factorization we obtain the components of the rebus: the diagonal blocks D_k, the lowest level factors U_k and V_k and a binary tree of low rank factors R, W and B. This representation allows us to efficiently recover the original matrix, while exposing any low-rank structure.

4.2. Rebus representation of differential operators

The rebus is a hierarchical low-rank representation of a linear operator. It is especially suitable for integral and differential operators.

To show the extent to which we may expect to realize savings in storage and number of operations, consider the hierarchical partitioning of a 1024 by 1024 Chebyshev derivative matrix illustrated in Fig. 2. It is a

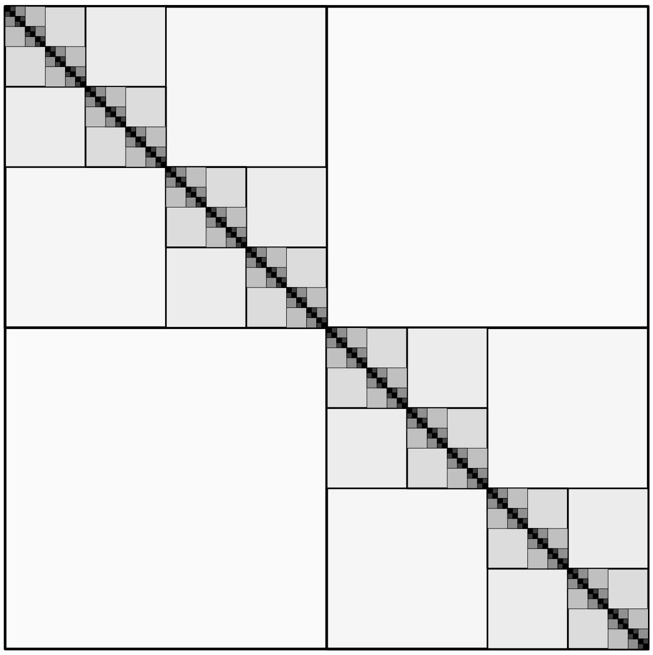

Fig. 2. Rank structure of the Chebyshev derivative matrix with shading proportional to rank.

special case of the partitioning described in Section 4.1. In this example each partition takes exactly half the rows or columns.

The shading of the figure is proportional to the percentage of full rank, with black squares being full rank. Table 1 shows the rank at each hierarchical level.

The goal of the rebus representation is to keep the derivative operator, the timestepping operator and all related quantities in rebus form at all stages of the numerical method. This allows us to use the low-rank representations for these blocks and use fast algorithms designed to take advantage of the binary tree structure to achieve large computational savings.

The details of the algorithms used for constructing the representations and solving the resultant systems are available elsewhere and are not the subject of this article. Details of these algorithms can be found in [12,13].

5. Numerical method

We consider a general PDE that can be written in the form

$$u_t = \mathbf{L}(\mathbf{x}, t, \mathbf{u}) + \mathbf{N}(\mathbf{x}, t, \mathbf{u}), \quad t > 0, \;\; \mathbf{x} \in \Omega, \tag{2}$$

where **L** and **N** represent a linear and a non-linear differential operators, respectively. Inhomogeneous terms, if present, are also included in **N**. The domain Ω is bounded and the equation is supplemented with appropriate initial and boundary conditions. We also assume that the leading order terms at small scales (e.g. terms with the highest order derivatives) are linear and thus contained in **L**.

With a collocation method the computational cost of incorporating boundary conditions of Dirichlet, Neumann or mixed type is low. It is most convenient, in this approach, to use boundary bordering as described by Boyd in [11].

To illustrate ideas, let us consider the particular case where the right hand side of the differential equation can be written as the sum of a linear elliptic operator and a non-linear operator, possibly including an inhomogeneous forcing term:

$$u_t = \nabla \cdot (a\nabla \mathbf{u}) + \mathbf{N}(\mathbf{x}, t, \mathbf{u}), \tag{3}$$

where $a > 0$. This type of equations arises routinely as diffusion-convection equations in computational fluid dynamics or reaction-diffusion problems in chemistry. As explained earlier, a popular approach in this situation is to treat the elliptic part implicitly and the other terms explicitly. The reason for this LI approach is that the elliptic term is the stiffest and gives rise to severe timestep constraints if treated explicitly. The remaining non-linear terms are treated explicitly as their implicit discretization would result in a non-linear system which would be difficult and costly to invert.

Table 1
Rank structure of the 1024×1024 Chebyshev derivative matrix

N	Rank of $N \times N$ block	% of full rank
512	11	2.1
256	10	3.9
128	10	7.8
64	9	14
32	8	25
16	7	44
8	6	75
4	4	100

Terms that we are treating explicitly need to be evaluated before we proceed to solve the system. Since the rebus representation of the derivative is already being computed, it may be applied cheaply to a vector by using a rebus–vector multiply as described in [13] to efficiently calculate any derivatives in the explicit term. This would be needed for example in the common case of an advective term. Here we focus on the problem of integrating implicitly the stiff elliptic term without leaving the physical space.

5.1. Elliptic problem

The timestepping solution of PDEs of this type reduces to solving an elliptic equation at each timestep. Since we are in the spatial domain, imposing the boundary conditions at each timestep introduces no extra complications.

In most spectral methods, a finite difference discretization is used in time. As the most simple example, consider a first-order (backward Euler) discretization. Higher order schemes in time can be easily implemented with exactly the same approach. The time discretization is

$$\frac{1}{\Delta t}(\mathbf{u}(\mathbf{x},t+\Delta t)-\mathbf{u}(\mathbf{x},t)) = \mathbf{L}(\mathbf{x},t+\Delta t,\mathbf{u})+\mathbf{N}(\mathbf{x},t,\mathbf{u}).$$

To step our solution forward in time with given, time-dependent Dirichlet boundary conditions we solve

$$\mathscr{L}_{\Delta t}\mathbf{u}(\mathbf{x},t+\Delta t) = \mathbf{u}(\mathbf{x},t)+\Delta t\mathbf{N}(\mathbf{x},t,\mathbf{u}), \tag{4}$$

where $\mathscr{L}_{\Delta t} = I - \Delta t \cdot \mathbf{L}$.

The solution at each timestep consists of the following steps.

1. Generate the rebus representation of D, the Chebyshev derivative operator.
2. Use the fast rebus–rebus multiplication algorithm to generate the relevant D^n operator.
3. Use scaling and diagonal updates to generate $\mathscr{L}_{\Delta t}$. Algorithms for this are presented in Sections 5.2 and 5.3.
4. Evaluate the non-linear term $\mathbf{N}(\mathbf{x},t,\mathbf{u})$ using $\mathbf{u}$ and possibly the rebus representation of D.
5. Add the right hand side terms to obtain a single, vector-valued, right hand side.
6. Apply boundary conditions via an efficient leaf-update of $\mathscr{L}_{\Delta t}$, described in Section 6.
7. Solve the system using the algorithm presented in [12].

Step 1 may of course be done only once for a given set of nodes and may then be stored. In nearly all problems of interest, step 2 may be done as a preprocessing step and need not be repeated. For a constant coefficient equation with uniform timesteps, step 3 may also be taken out of the loop. In this special case we need only evaluate the non-linear term, update boundary conditions and solve.

5.2. Rebus scaling

Consider the operation $A \to c \cdot A$. For a matrix representation of A we clearly update the matrix elements $a_{i,j} \to c \cdot a_{i,j}$. Now consider the equivalent rebus operation. We will use the notation of Section 4.1.

The interaction of each subdomain with each other subdomain is represented either by $D_{K;i}$ or the product $U_{k;i}B_{k;i,j}V^{\mathrm{H}}_{k;j}$, for some k. U and V themselves satisfy the relations (1).

Thus, the structure can correctly be scaled by

$$D_{K;i} \to c \cdot D_{K;i} \quad \text{for } i = 1,\ldots,2^K$$

and

$$B_{k;i,j} \to c \cdot B_{k;i,j} \quad \text{for } k = 1, \ldots, K \text{ and } (i,j) = (1,2), (2,1).$$

All other factors of the rebus remain unchanged.

5.3. Diagonal updates

The backward Euler timestepping discretization of the PDE is given by

$$(I - \Delta t \cdot \mathbf{L})\mathbf{u}(\mathbf{x}, t + \Delta t) = \mathbf{u}(\mathbf{x}, t) + \Delta t \mathbf{N}(\mathbf{x}, t, \mathbf{u})$$

and in this case the operator we must represent as a rebus is

$$\mathscr{L} = I - \Delta t \cdot \mathbf{L}.$$

Given the rebus representation of $\mathbf{L}$, forming this operator requires a scaling by $-\Delta t$, followed by a diagonal update.

Every off-diagonal block of $I - \Delta t \cdot \mathbf{L}$ is equal to the corresponding block of $-\Delta t \cdot \mathbf{L}$. Thus the update need operate only on the blocks $D_{K;i}$. But these are exactly the blocks which are stored explicitly as normal matrices in our rebus representation.

Thus, the structure can be correctly updated by

$$D_{K;i} \to I - D_{K;i} \quad \text{for } i = 1, \ldots, 2^K,$$

where $D_{K;i}$ are the diagonal blocks of our scaled rebus.

6. Boundary conditions by boundary bordering

The main reason for pursuing these methods is to efficiently treat PDEs with non-periodic boundary conditions. It is appropriate then, to consider how to enforce various boundary conditions in the rebus formulation.

As discussed by Boyd in [11], the most robust and flexible method of imposing physical boundary conditions for a matrix formulation of Chebyshev collocation is boundary bordering. The principle of this method is to allocate m rows of the matrix to explicitly imposing the m boundary conditions, of whatever type. Thus, to impose Dirichlet boundary conditions on a discretization of a second order equation in one dimension we collocate at $N - 2$ interior points of the interval and use two rows of the matrix to impose boundary conditions.

The equivalent process for a rebus proceeds as follows. Instead of using the first two rows of the matrix we need to respect the spatial structure and operate on the first and last rows, corresponding to the boundary positions. We can then accomplish the bordering. However, the rows that need to be replaced are not readily available, as they are split between a number of blocks, which are themselves factored.

The process is less straightforward than bordering a matrix, but is not difficult. The process may be broken down as follows.

- Initialize by setting the m required rows to zero within the rebus.
- Form the required boundary conditions as dense matrix rows.
- Construct a low-rank product expressing the desired boundary conditions in matrix form.
- Use Algorithm 6.4 to add this low rank product to the rebus.

Thus the procedure amounts to zeroing out m rows of the rebus and performing one rank-m addition.

6.1. Initialization

We may modify the first and last diagonal block, $D_{K;1}$ and $D_{K;N}$ directly, as we would the corresponding matrix. However, the remaining factors contributing to the border rows still need to be set to zero. (We tacitly assume here that all the border rows can be contained in the rows covered by these blocks. The procedure below extends to the more general case).

We use the fact that each other element of the first row is the first row of a block $U_{k;i}B_{k;i,j}V^{\mathrm{H}}_{k;j}$, for some k, and that all of the $U_{k;i}$ are generated from the lowest level $U_{K;i}$ via the relation

$$U_{k-1;i} = \begin{pmatrix} U_{k;m}R_{k;m} \\ U_{k;n}R_{k;n} \end{pmatrix}. \tag{5}$$

Our requirement is that

$$\sum_{\mu=1}^{p}\sum_{\nu=1}^{q}(U_{k;i})_{1,\mu}(B_{k;i,j})_{\mu,\nu}(V^{\mathrm{H}}_{k;j})_{\nu,\kappa} = 0$$

for all κ for each block. Thus it is sufficient to require that

$$(U_{k;1})_{1,\mu} = 0 \quad \text{for all } \mu \quad \text{for all } k.$$

And from the recursion relation (5) it is sufficient to enforce

$$(U_{K;1})_{1,\mu} = 0 \quad \text{for all } \mu$$

to achieve boundary bordering of the first row.

Similarly, we impose analogous conditions on all other rows that are to be used for boundary conditions. For example, we would use

$$(U_{K;N})_{N_K,\mu} = 0 \quad \text{for all } \mu$$

for boundary bordering on the final row.

6.2. Boundary conditions in matrix form

We now generate the boundary rows that we would border with in a matrix method.

For Dirichlet boundary conditions or conditions on any linear combination of endpoint or interior values in the form

$$\sum_{j=1}^{N}\omega_j u(x_j) = \alpha,$$

where x_j are our Chebyshev nodes, we simply border with the vector of weights ω. In the Dirichlet case this amounts to a single 1 in the first or last position.

Alternatively, we may be presented with Neumann conditions or conditions on some linear combination of derivatives at the endpoints or interior points. Our boundary condition then would have the form

$$\sum_{i=1}^{N}\omega_i u'(x_i) = \alpha.$$

Let D be the Pseudospectral derivative operator. Using the matrix representation of the derivative, our condition is equivalent to

$$\sum_{j=1}^{N}\left(\sum_{i=1}^{N}\omega_i D_{ij}\right)u(x_j)=\alpha. \tag{6}$$

Thus, we border our matrix with a linear combination of rows from the derivative matrix D as determined by the weights ω. In the case of a simple Neumann condition at $x = -1$, we would border with the first row of D.

For more exotic boundary conditions or constraints the treatment is equally straightforward. An integral condition may be represented by any suitable Chebyshev quadrature scheme, for example the Gaussian or Clenshaw–Curtis scheme. Since quadratures are expressed in terms of weights as

$$\sum_{j=1}^{N} w_j u(x_j)=\alpha,$$

the correct boundary row is again simply the vector of weights w.

6.3. Formulation as a low-rank product

Given our m boundary conditions, we now need to position them appropriately in our rebus. For Dirichlet conditions in one dimension, for example, we should respect the geometry of the problem and border on the first and last rows. Similarly for Neumann conditions. The treatment of more exotic conditions is less obvious but should be guided by attempts to preserve locality in the rebus structure.

Putting our m boundary conditions together into an m by n matrix B, we then write a simple n by m permutation matrix Π so that the product ΠB contains the rows in their chosen position relative to the rows in the rebus.

These are the same rows that were initialized in step 6.1 and should also have the corresponding value of the condition α on this row on the right hand side of our rebus equation.

6.4. Low-rank addition

It remains to combine the initialized rebus with the boundary conditions. We can do this using an efficient algorithm for a low-rank update of a rebus.

Consider the problem of adding the product AC^{T} to a rebus. As in Section 4 we will name the components of the rebus as follows: diagonal blocks D_k, the lowest level factors U_k and V_k and a binary tree of low rank factors R, W and B.

We first partition the columns of A and C commensurately with the rebus.

$$A=A_0=\begin{pmatrix}A_{1;1}\\A_{1;2}\end{pmatrix}=\begin{pmatrix}A_{2;1}\\A_{2;2}\\A_{2;3}\\A_{2;4}\end{pmatrix}=\cdots=\begin{pmatrix}A_{k;1}\\\vdots\\A_{k;2^k}\end{pmatrix}$$

and similarly for C.

At the kth level we need to represent $D_{k;i}+A_{k;i}C_{k;i}^{\mathrm{T}}$ for $j = 1,\ldots,2^k$ and $U_{k;i}B_{k;i,j}V_{k;j}^{\mathrm{T}}+A_{k;i}C_{k;j}^{\mathrm{T}}$ for $(i,j) = (2l,2l - 1),(2l - 1,21)$, for $l = 1,\ldots, 2^{k-1}$.

The off-diagonal blocks can be updated by assigning

$$U_{k;i}\leftarrow\begin{pmatrix}U_{k;i} & A_{k;i}\end{pmatrix},\quad V_{k;j}\leftarrow\begin{pmatrix}V_{k;j} & C_{k;j}\end{pmatrix},\quad B_{k;ij}\leftarrow\begin{pmatrix}B_{k;ij} & 0\\0 & I_m\end{pmatrix},$$

where m is the rank of the product AC^{T}.

It remains to treat the diagonal blocks. These are either dense matrices or each is a rebus with one fewer levels than the case just treated. If a block is a dense matrix (a zero-level rebus) we simply perform the addition

$$D_{k;i} \leftarrow D_{k;i} + A_{k;i}C_{k;i}^{\mathrm{T}}.$$

If the block is a rebus with one fewer levels, we recursively apply the original procedure until all the diagonals have terminated with a dense block.

So using a straightforward recursive algorithm we can efficiently add a low rank matrix to a rebus.

7. Limitations and extensions

7.1. Conditioning of Chebyshev derivative operators

Working with Chebyshev collocation methods with many collocation points, high order derivatives and multiple dimensions is a numerically dangerous proposition. The methods presented in this paper aim to make certain types of calculation much more efficient. However, the numerical considerations of roundoff error and ill conditioning remain.

It is well known that the discretization of the derivative operator on Chebyshev nodes gives rise to an ill-conditioned matrix. As N, the number of discretization points, increases, the condition number of the derivative matrix D grows with $O(N^2)$. Similarly, the condition number of the second derivative operator grows with $O(N^4)$. As the order of the linear term or the dimension increases, this can quickly become unmanageable.

For example in a Kuramoto–Sivashinsky equation [35] we would have an $O(N^8)$ condition number in 1d or $O(N^{16})$ in 2d. If we are working with 16 digits of accuracy then at a grid size around $N = 10^2$ we can expect our answer to contain no meaningful digits. In a two dimensional simulation we cannot expect accuracy on any grid at all.

Dealing with this issue is part of using pseudospectral Chebyshev methods and there is a considerable literature dealing with it.

7.1.1. Classical approaches

In the literature, there are a number of well known strategies to improve the condition number and accuracy of pseudospectral derivative matrices.

Numerical evaluation of the matrix representation of the derivative is subject to serious roundoff errors, especially near the boundary points. Bayliss shows how a simple adjustment to the matrix, chosen to ensure that constant functions lie in the null space of the matrix, can improve this [6]. It is also possible to use alternate formulae to generate the matrix elements, as in Tang and Trummer [32] in order to avoid the cancellations leading to numerical error. Each of these methods can improve the accuracy of the matrix by a few orders of magnitude. The scaling of the condition number is unfortunately not improved.

In a series of papers Heinrichs [21] described a specific basis recombination strategy to improve the condition number of D^k to order N^k. This strategy is also treated by Boyd in [11]. There is also the mapped Chebyshev approach introduced by Tal-Ezer [24] and further studied in [16]. This also reduces the condition number of the kth order derivative to $O(N^k)$.

7.1.2. Extended precision

The most straightforward way to keep ill-conditioning from interfering with calculations which require high precision is to take advantage of extended precision floating point numbers. These are now available on a number of architectures and supported by many compilers [4].

Using quadruple precision allows us to, for example, calculate on a 100 by 100 grid using Chebyshev collocation and still retain 16 digits of accuracy. In a less extreme case, if 8 digits sufficed, we could use 1000 Chebyshev nodes in each direction.

Calculating at extended precision incurs its own costs and does nothing to mitigate the poor numerical behavior of the underlying problem. However, it does keep numerical errors at bay without requiring any extra algorithmic or analytic complexity. Thus for practical problems that are otherwise intractable it offers a realistic approach.

7.1.3. Alternate basis sets

It is well known that other global basis sets, while lacking the optimality properties of the Chebyshev polynomials, lead to much better conditioned derivative operators. Depending on the specifics of the problem it may be advisable to work in a basis of Legendre or Jacobi polynomials.

We note that the methods developed here for Chebyshev collocation methods are equally applicable in any other basis. The low rank structure we are taking advantage of stems from the smoothness of the operator itself, not the basis in which it is represented.

7.1.4. Iterative refinement

If we are only able to obtain relatively low accuracy, but have our system in factored form it may be desirable to employ a few cycles of iterative refinement. By calculating our backward error and performing another fast solve we may extend the accuracy of the solution.

7.1.5. Exploiting limited precision

We can also use the fact that our final accuracy is known to be limited to our advantage. Whether it is due to the order of our timestepping scheme or due to the conditioning of the differential operator, we often know that we are working to less than machine precision.

Similar to a wavelet representation, the rebus is a thresholded representation and captures the operator to arbitrary but finite precision. The thresholding is similar to that of an economy SVD, where basis vectors corresponding to small singular values are discarded.

If we know that our solution will not have more than, say, 10 digits of accuracy we can threshold more aggressively, discarding factors of the off-diagonal blocks corresponding to singular values smaller than 10^{-10}. This results in lower rank representations of the off-diagonal blocks and hence faster computation speed.

By remaining aware of the limitations imposed on the accuracy of our solution in timestepping methods we may at least dispense with unnecessary computations and work with the relevant components of our problem. If only very low accuracy is required, as would be the case for a preconditioner or an iterative refinement step, we may work with a coarse approximation to the operator and proceed through the calculations very rapidly.

7.2. Alternate timestepping schemes for stiff non-linear PDEs

LI methods are a popular choice for solving PDEs with a non-linear contribution and a stiff linear term. However, there are other methods available. Kassam and Trefethen compare the available schemes in [9], and suggests the IF and ETD schemes may be superior choices. Methods of the ETD type appear to have been introduced by Beylkin, Keiser and Vozovoi in [8].

These alternate methods rely on the observation that the linear part of the equation can be solved exactly by a matrix exponentiation. As ever, a suitable explicit step is sought for the non-linear terms so as to avoid an iterative solution of a non-linear system.

To take a timestep using this type of method, the linear term must be inverted, exponentiated or raised to a power. In the scheme favored in [9], (ETDRK4, an exponential time differencing technique based on the fourth order Runge–Kutta scheme) all of these must be done. This poses no particular problem if we have periodic boundary conditions in one dimension. However, if this is not the case and we cannot diagonalize our operator we must be able to efficiently invert, exponentiate and multiply a full matrix.

We will sketch how the rebus methods described in Section 5 extend to timestepping with these modern methods.

7.2.1. Matrix exponentiation

In calculating the timestep in the IF and ETD schemes, matrix exponentiation plays a central role. The IF which multiplies both sides of the PDE is of the form $e^{\mathbf{L}h}$, and needs to be calculated. If the linear operator is constant, it only needs to be calculated once. If we are working in Fourier space, the operator can be rendered diagonal and again, the calculation is straightforward. Note that the exponential will itself not be sparse.

The more difficult case is that of a function $\mathbf{L}$ on a finite domain with physical boundary conditions. If a pseudospectral Chebyshev discretization is used, the matrix exponential becomes very expensive to evaluate. It is an $O(N^3)$ operation and even for relatively modest N may be the rate determining step.

The matrix exponential is most often computed using the scaling and squaring algorithm favored by Golub and Van Loan in [19] and in the review papers [25,26]. The calculation is accomplished with matrix multiplications and a matrix solve and is therefore easily implemented for the rebus structure.

With an $O(N)$ matrix exponentiation, IF and ETD schemes could be applied to problems with a time-varying linear operator and to large problems with non-periodic boundary conditions. A similar observation led to the introduction of ETD schemes in [8], where the authors demonstrated a sparse matrix exponentiation for wavelet representations of strictly elliptic operators.

8. Numerical examples

The following section reports the results of applying these methods to a number of test problems. Since conventional methods find the combination of non-periodic boundary conditions with many collocation points the most problematic, we focus on such problems.

The implicit treatment of the diffusion terms corresponds to the conventional use of LI methods (see [3]) for the convection-diffusion or reaction-diffusion equations arising in chemical simulation. Such simulations typically use a spectral discretization in space and face exactly the problem we address: efficiently solving the equations arising from an implicit discretization in time.

8.1. Time-varying non-homogenous Dirichlet conditions

Consider the test problem of a diffusion equation applied to a Gaussian function.

$$\begin{aligned} &\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \qquad -1 \leqslant x \leqslant 1, \quad t \geqslant 0, \\ &u(x,0) = \exp(x^2) \qquad t = 0, \\ &u(\pm 1, t) = \sqrt{\frac{1}{t+1}} \exp\left(\frac{-1}{4(t+1)}\right) \qquad t > 0. \end{aligned} \tag{7}$$

Note that the Dirichlet boundary conditions are non-homogenous and time varying. This is the same kind of boundary condition we would need to apply for a Dirichlet boundary control problem.

This mixed initial-boundary value problem has the exact solution

$$u(x,t) = \sqrt{\frac{1}{t+1}} \exp\left(\frac{-x^2}{4(t+1)}\right). \tag{8}$$

A second-order Crank–Nicolson scheme was employed for the time integration of the linear term and this example does not have a non-linear term. To demonstrate that the current approach retains the second order convergence expected of a Crank–Nicolson method we first refine our timestep on a fixed grid. Table 2 demonstrates the effect of taking 2^n steps for $n = 3, \ldots, 14$ on a collocation grid of 64 Gauss–Lobatto points.

Assuming the error of the scheme is of the form $\|u(x,1) - \hat{u}(x,1)\|_\infty = k \cdot N^{-\alpha}$, a regression of the data determines $\alpha = 2.0$ with $R^2 = 1.00$. As expected, our implementation of the Crank–Nicolson scheme is second-order.

To demonstrate the scaling properties of the method, now fix Δt and refine our collocation grid. As the spatial accuracy is already exponentially convergent, our accuracy is limited by the fixed timestep size. We are refining the grid in order to show the desirable scaling properties of the method as N grows.

We solve Eq. (7) from $t = 0$ to $t = 1$ for a changing number of collocation points while maintaining our error $\|u(x,1) - \hat{u}(x,1)\|_\infty \leqslant 10^{-5}$. We report timings and uniform errors with $\Delta t = 0.01$.

A regression analysis of the data in Table 3 shows the cost to scale as $N^{1.3}$ with $R^2 = 1.00$. This exponent is not optimal, as linear scaling can theoretically be achieved [13]. The performance is competitive with the $N\log(N)$ currently possible for transform-based, diagonalizable problems. It is clearly superior to the N^3 cost expected for non-diagonalizable problems (see Fig. 3).

Table 2
Convergence after N rebus timesteps on 64 Chebyshev nodes

$\log_2(\Delta t)$	Δt	Error
−3	1.25E−01	3.29E−05
−4	6.25E−02	1.07E−05
−5	3.13E−02	1.93E−06
−6	1.56E−02	4.87E−07
−7	7.81E−03	1.22E−07
−8	3.91E−03	3.04E−08
−9	1.95E−03	7.61E−09
−10	9.77E−04	1.90E−09
−11	4.88E−04	4.75E−10
−12	2.44E−04	1.24E−10
−13	1.22E−04	2.82E−11
−14	6.10E−05	7.05E−12

Table 3
Timings for 100 rebus timesteps on N Chebyshev nodes

N	CPU time (s)	Error (e−6)
200	2.13	1.20
400	5.23	1.19
800	14.1	1.19
1600	38.5	1.20
3200	87.4	1.22
6400	213	1.44

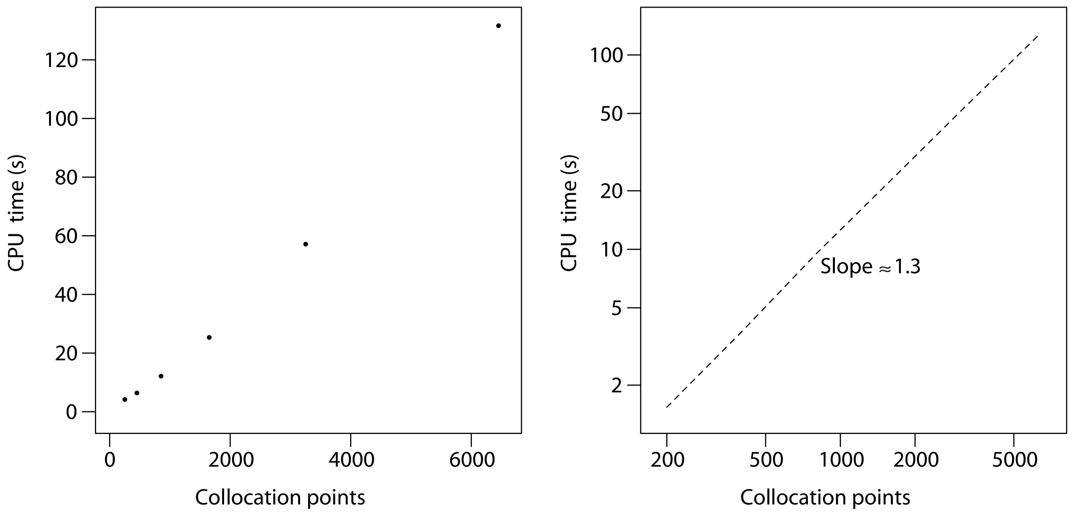

Fig. 3. Timings for 100 rebus timesteps on N Chebyshev nodes with linear and log scales.

8.2. Homogenous Neumann conditions

Having demonstrated the second order convergence of the method and the scaling of computational cost with grid size, we now apply the method to a more challenging simulation. We timestep a reaction-diffusion equation with a non-linear term and Dirichlet boundary conditions.

Consider the initial value problem for an Allen–Cahn equation of the form

$$\begin{aligned} &u_t = \epsilon u_{xx} + u - u^3 \qquad -1 \leqslant x \leqslant 1, \quad t \geqslant 0, \\ &u(x,0) = \sin(5\pi x/2) \qquad t = 0, \\ &u_x(\pm 1, t) = 0 \qquad t > 0. \end{aligned} \tag{9}$$

Eq. (9) has stable equilibria at $u(x) = \pm 1$ and an unstable equilibrium at $u(x) = 0$. It also demonstrates slow dynamics, whereby metastable states may persist for relatively long periods before undergoing a rapid transition to a lower energy state [35].

In this case the linear term was integrated via a Crank–Nicolson scheme, the non-linear term via a second order Adams–Bashforth scheme and the Neumann boundary conditions were enforced as described in Section 6. Table 4 shows the computational time taken for different grid sizes.

Table 4
Timings for 100 rebus timesteps on N Chebyshev nodes

N	CPU time (s)
200	2.06
400	5.12
800	13.9
1600	35.5
3200	86.7
6400	206

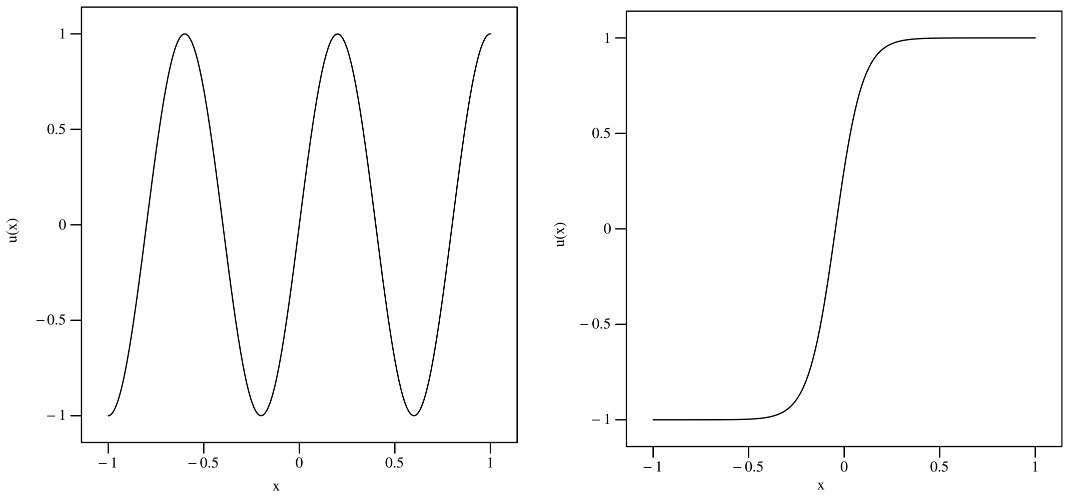

Fig. 4. Initial condition and final (t = 127) state for the Allen–Cahn equation.

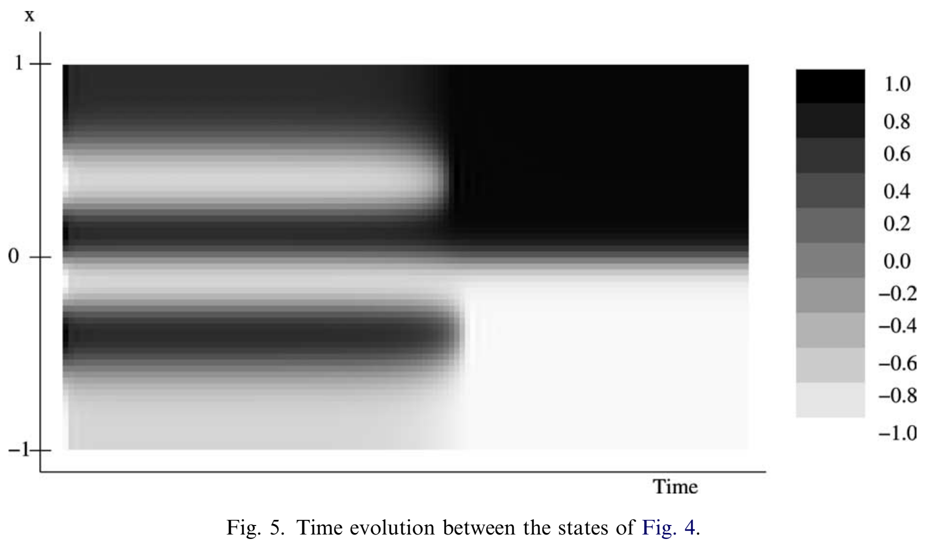

Fig. 5. Time evolution between the states of Fig. 4.

A regression of the data in Table 4 shows that the scaling exponent has remained 1.3, as in Section 8.1, again with $R^2 = 1.00$. The addition of Dirichlet boundary conditions and a non-linear term has not altered the scaling behavior of the method.

The initial and final states are shown in Fig. 4 and the evolution of the state is shown in Fig. 5. Around the midpoint of this evolution we observe the transition out of the metastable state. High order methods are desirable for tracking transitions such as this. As the solution is stable over relatively long periods, an adaptive step size would also be appropriate for this kind of problem. This is straightforward to implement in a rebus scheme. Whatever timestep is needed, the relevant operator is still generated via fast operations from the derivative rebus, which is already known.

All experiments were carried out on a dual 1 GHz PowerPC G4 with 2MB L3 cache per processor, and 1.5 GB RAM, using vendor supplied BLAS in double precision. Only a single CPU was used.

9. Conclusion

In this paper, we demonstrate a new approach for efficient time integration of PDEs via Chebyshev spectral discretization. By using a hierarchically semi-separable representation (a rebus) it becomes possible to use an implicit discretization in time and still directly solve the resulting system of equations in physical space. The cost is shown to scale at a rate which is competitive with the $N \log N$ cost of current fast methods for periodic problems.

The fact that the method allows different types of boundary condition to be combined with a fast algorithm for solution constitutes an advance over current methods. In situations where the problem cannot be diagonalized or periodic boundary conditions imposed, these methods allow us to work efficiently with the resulting non-sparse matrices. Nonperiodic problems arise frequently in practice.

These advantages, combined with the efficient handling of dense matrices which arise in other timestepping methods, such as ETD, give these methods the potential to treat problems that may previously have been intractable.

References

[1] A.W. Appel, An efficient program for many-body simulation, SIAM J. Sci. Stat. Comput. 6 (1) (1985) 85–103.

[2] U.M. Ascher, S.J. Ruuth, R.J. Spiteri, Implicit–explicit Runge–Kutta methods for time-dependent partial differential equations, Appl. Numer. Math. 25 (2–3) (1997) 151–167.

[3] U.M. Ascher, S.J. Ruuth, B. Wetton, Implicit–explicit methods for time-dependent partial differential equations, SIAM J. Num. Anal. 32 (1995) 797–823.

[4] D.H. Bailey, A portable high performance multiprecision package, Technical Report RNR-90-022, Moffett Field, CA 94035, 1992.

[5] J.E. Barnes, P. Hut, A hierarchical O(Nlog N) force-calculation algorithm, Nature 24 (6270) (1986) 446–449.

[6] A. Bayliss, A. Class, B.J. Matkowsky, Roundoff error in computing derivatives using the Chebyshev differentiation matrix, J. Comput. Phys. 116 (1994) 380–383.

[7] G. Beylkin, N. Coult, M.J. Mohlenkamp, Fast spectral projection algorithms for density-matrix computations, J. Comput. Phys. 152 (1) (1999) 32–54.

[8] G. Beylkin, J.M. Keiser, L. Vozovoi, A new class of time discretization schemes for the solution of nonlinear PDEs, J. Comput. Phys. 147 (2) (1998) 362–387.

[9] A.-K. Kassam, L.N. Trefethen, Fourth-order time stepping for stiff PDEs, Technical Report NA-03/14, Oxford University, 2003.

[10] G. Beylkin, K. Sandberg, Wave propagation using bases for bandlimited functions, Technical Report APPM 518, University of Colorado, 2003.

[11] J.P. Boyd, Chebyshev and Fourier Spectral Methods, Springer-Verlag, New York, 1989.

[12] S. Chandrasekaran, M. Gu, W. Lyons, A fast and stable adaptive solver for hierarchically semi-separable representations, Technical Report UCSB Math 2004-20, U.C. Santa Barbara, 2004.

[13] S. Chandrasekaran, M. Gu, T. Pals. Fast and stable algorithms for hierarchically semi-separable representations, submitted for publication.
[14] R. Coifman, V. Rokhlin, S. Wandzura, The fast multipole method for the wave equation: a pedestrian prescription, IEEE Antennas Propag. Mag. 35 (3) (1993) 7–12.
[15] M. Crouzeix, Une mèthode multipas implicite-explicite pour l'approximation des èquations d'èvolution paraboliques, Numer. Math. 35 (1980) 257–276.
[16] W.S. Don, A. Solomonoff, Accuracy enhancement for higher derivatives using Chebyshev collocation and a mapping technique, SIAM J. Sci. Comput. 18 (4) (1997) 1040–1055.
[17] B. Fornberg, A Practical Guide to Pseudospectral Methods, Cambridge University Press, Cambridge, UK, 1996.
[18] D. Gines, G. Beylkin, J. Dunn, LU factorization of non-standard forms and direct multiresolution solvers, Appl. Comput. Harmon. Anal. 5 (2) (1998) 156–201.
[19] G. Golub, C. Van Loan, Matrix Computations, Johns Hopkins University Press, Baltimore, MA, 1996.
[20] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, J. Comput. Phys. 73 (2) (1987) 325–348.
[21] W. Heinrichs, Improved condition number for spectral methods, Math. Comp. 53 (1989) 103–119.
[22] R.W. Hockney, J.W. Eastwood, Computer Simulation Using Particles, Taylor & Francis, Inc, 1988.
[23] P. Jones, J. Ma, V. Rokhlin, A fast direct algorithm for the solution of the laplace equation on regions with fractal boundaries, J. Comput. Phys. 113 (1) (1994) 35–51.
[24] D. Kosloff, H. Tal-Ezer, Modified chebyshev pseudospectral methods with $O(N^{-1})$ time step restriction, J. Comput. Phys. 104 (1993) 457–469.
[25] C.B. Moler, C.F. Van Loan, Nineteen dubious ways to compute the exponential of a matrix, SIAM Rev. 20 (4) (1978) 801–836.
[26] C.B. Moler, C.F. Van Loan, Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later, SIAM Rev. 45 (1) (2003) 3–49.
[27] S.T. O'Donnell, V. Rokhlin, A fast algorithm for the numerical evaluation of conformal mappings, SIAM J. Sci. Stat. Comput. 10 (3) (1989) 475–487.
[28] V. Rokhlin, Rapid solution of integral equations of classical potential theory, J. Comput. Phys. 60 (1985) 187–207.
[29] V. Rokhlin, Rapid solution of integral equations of scattering theory in two dimensions, J. Comput. Phys. 86 (2) (1990) 414–439.
[30] P. Starr, On the numerical solution of one-dimensional integral and differential equations, Technical Report YALEU/DCS/RR-888, Yale University, 1991.
[31] P. Starr, V. Rokhlin, On the numerical solution of two-point boundary value problems II, Technical Report YALEU/DCS/RR-802, Yale University, 1990.
[32] T. Tang, M.R. Trummer, Boundary layer resolving pseudospectral methods for singular perturbation problems, SIAM J. Sci. Comput. 17 (1996) 430–438.
[33] J.M. Varah, Stability restrictions on second order, three level finite difference schemes for parabolic equations, SIAM J. Numer. Anal. 17 (1980) 300–309.
[34] N. Yarvin, V. Rokhlin, A generalized one-dimensional fast multipole method with application to filtering of spherical harmonics, J. Comput. Phys. 147 (2) (1998) 594–609.
[35] D. Zwillinger (Ed.), Handbook of Differential Equations, Third ed., Academic Press, Boston MA, 1997.