View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by IntechOpen

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

3

Matrix Based Operatorial Approach to Differential and Integral Problems

Damian Trif
Babes-Bolyai University of Cluj-Napoca
Romania

1. Introduction

Many problems of the real life lead us to linear differential equations of the form

$$\sum_{k=0}^{m} P_k(x)\frac{d^k y}{dx^k} = f(x), \quad x \in [a,b] \tag{1}$$

with the general conditions

$$\sum_{j=1}^{m} \alpha_{ij}^{(1)} y^{(j-1)}(x_{ij}^{(1)}) + \cdots + \sum_{j=1}^{m} \alpha_{ij}^{(m)} y^{(j-1)}(x_{ij}^{(m)}) = \beta_i, \quad i = 1,...,m \tag{2}$$

where $x_{ij}^{(k)} \in [a,b]$, $\forall i,j,k = 1,...,m$. These multipoint conditions include (for $m = 2$, for example)
- initial value conditions, $y(a) = \beta_1$, $y'(a) = \beta_2$,
- boundary value conditions $\alpha_{11}y(a) + \alpha_{12}y'(a) = \beta_1$, $\alpha_{21}y(b) + \alpha_{22}y'(b) = \beta_2$,
- periodic conditions $y(a) - y(b) = 0$, $y'(a) - y'(b) = 0$.

Eigenvalue problems for linear differential operators

$$\sum_{k=1}^{m} P_k(x)\frac{d^k y}{dx^k} + (P_0(x) - \lambda w(x))\, y = 0, \quad x \in [a,b]$$

$$\sum_{j=1}^{m} \alpha_{ij}^{(1)} y^{(j-1)}(x_{ij}^{(1)}) + \cdots + \sum_{j=1}^{m} \alpha_{ij}^{(m)} y^{(j-1)}(x_{ij}^{(m)}) = 0, \quad i = 1,...,m$$

are also included in this general form. Moreover, nonlinear problems where the r.h.s. $f(x)$ is replaced by $f(x, y(x), y'(x), ..., y^{(m-1)}(x))$ can be solved using Newton's method in the functional space $C^m[a,b]$ by solving a sequence of linear problems (1)+(2).
MATLAB uses different methods to solve initial condition problems (`ode` family) or boundary value problems (`bvp4c` or `bvp5c`) based on Runge-Kutta, Adams-Bashforth-Moulton, BDF algorithms, etc.
One of the most effective methods for solving (1)+(2) is to shift the problem to the interval $[-1,1]$ and then to use the *Chebyshev spectral methods*, i.e. to approximate the solution y by a finite sum of the Chebyshev series

$$y(x) = \frac{1}{2}c_0 T_0(x) + c_1 T_1(x) + c_2 T_2(x).... \tag{3}$$

Here $T_k(x) = \cos(k \cos^{-1}(x)), k = 0, 1, ...$ are the Chebyshev polynomials of the first kind and the coefficients c_k, $k = 0, 1, ...$are unknown. A spectral method is characterized by a specific way to determine these coefficients. The Chebyshev spectral methods could be implemented as
- Galerkin and tau methods, where we work in the spectral space of the coefficients $\mathbf{c} = c_0, c_1, c_2, ...$ of y or as
- spectral collocation (or pseudospectral) methods, where we work in the physical space of the values of y at a specific grid $\mathbf{x} = x_1, x_2, ... \in [-1, 1]$.
The well known MATLAB packages which use spectral methods, *MATLAB Differentiation Matrix Suite* (*DMS*) (Weideman & Reddy, 2000) and *Chebfun* (Trefethen et al., 2011), are based on the pseudospectral methods. Usually, these methods are implemented in an operatorial form: a differentiation matrix D (or linear operator) is generated so that $Y' = DY$ where the vector Y' contains the values of the derivative y' at the specific grid while Y contains the values of y at the same grid. The equation (1) becomes

$$\left(\sum_{k=0}^{m} diag(P_k(\mathbf{x}))D^k \right) Y = f(\mathbf{x}) \text{ i.e. } AY = F$$

and the conditions (2) are enclosed in the matrix A and in the vector F. The numerical solution of the differential problem is now $Y = A^{-1}F$. We note that MATLAB capabilities of working with matrices make it an ideal tool for matrix based operatorial approach.
There is a price to pay for using pseudospectral methods: the differentiation matrix D is full (while for finite differences or finite element methods it is sparse) and, more importantly, D is very sensitive to rounding errors. We give here a comparison between *DMS*, *Chebfun* and our proposed package *Chebpack* (based on the tau spectral method) for an eigenvalue problem suggested by Solomonoff and Turkel. Let us consider the evolution problem

$$u_t = -xu_x,\ u(x,0) = f(x),\ x \in [-1,1]\,,$$

with the exact solution $u(x,t) = f(xe^{-t})$. Here $x = \pm 1$ are outflow boundaries so that no boundary conditions are required. Using a Chebyshev spectral method to discretize the spatial part of the equation, the stability of time integration depends on the eigenvalues of that spatial part

$$-xu_x = \lambda u,\ x \in [-1,1]\,. \tag{4}$$

The exact (polynomial) eigenvectors are the monomials x^n and the corresponding eigenvalues are $\lambda_n = -n, n = 0, 1, ...$.
The commands for the *DMS* package

```
[x,D]=chebdif(64,1);L=eig(-diag(x)*D);
```

for *Chebfun*

```
N=chebop(@(x,u) -x.*diff(u),[-1,1]);L=eigs(N(66),64,'LR');
```

and for *Chebpack*

```
X=mult(64,[-1,1]);D=deriv(64,[-1,1]);L=eig(full(-X*D));
```

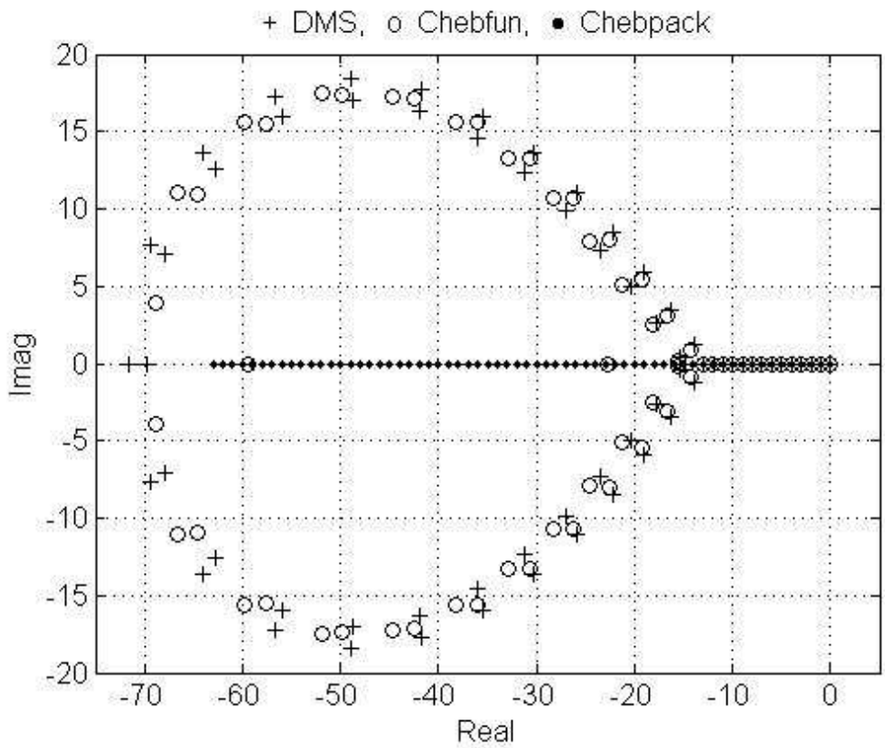

Fig. 1. Eigenvalues for the problem (4)

give the 64 approximated eigenvalues successively in the vector L and they are represented in Fig. 1. We see that *DMS* and *Chebfun* calculate accurately only a small number of eigenvalues, while *Chebpack* gives exactly all 64 eigenvalues.
The proposed package *Chebpack*, which is described in this chapter, is based on the representation (3) of the unknown functions and uses the *tau method* for linear operators (such as differentiation, integration, product with the independent variable,...) and the pseudospectral method for nonlinear operators – nonlinear part of the equations.
The tau method was invented by Lanczos (1938, 1956) and later developed in an operatorial approach by Ortiz and Samara (Ortiz & Samara, 1981). In the past years considerable work has been done both in the theoretical analysis and numerical applications.
Chebpack is freely accessible at `https://sites.google.com/site/dvtrif/` and at (Trif, 2011). All the running times in this chapter are the elapsed times for a 1.73GHz laptop and for MATLAB 2010b under Windows XP.

2. Chebpack, basic module

The package contains, at the basic module – level0, the tools which will be used in the next modules. Let us start with the Chebyshev series expansion of a given function y (Boyd, 2000):
THEOREM 1. *If y is Lipschitz continuous on [-1, 1], it has a unique representation as an absolutely and uniformly convergent series*

$$y(x) = \frac{c_0}{2}T_0(x) + c_1T_1(x) + c_2T_2(x) + ...,$$

and the coefficients are given by the formula

$$c_k = \frac{2}{\pi}\int_{-1}^{1} \frac{y(x)T_k(x)}{\sqrt{1-x^2}}dx,\ k = 0, 1, 2, ...$$

Let y_{N-1} be the truncation of the above Chebyshev series

$$y_{N-1}(x) = \frac{c_0}{2}T_0(x) + c_1T_1(x) + c_2T_2(x) + \cdots + c_{N-1}T_{N-1}(x) \tag{5}$$

and $dom = [-1, 1]$ be the working interval. Any interval $[a, b]$ can be shifted and scaled to $[-1, 1]$ by using the shifted Chebyshev polynomials

$$T_k^*(x) = T_k(\alpha x + \beta),\ \alpha = \frac{2}{b-a},\ \beta = -\frac{b+a}{b-a},\ x \in [a, b]\,.$$

First of all, we need a set of N collocation points $x_1, ..., x_N \in dom$ in order to find a good transformation between the above spectral approximation (5) of y and its physical representation $y(x_1), y(x_2), ..., y(x_N)$. Let

$$p_{N-1}(x) = \frac{a_0}{2}T_0(x) + a_1T_1(x) + ... + a_{N-1}T_{N-1}(x) \tag{6}$$

be the unique polynomial obtained by interpolating $y(x)$ through the points $x_1, ..., x_N$, see (Trefethen, 2000) for more details on the coefficients a_k versus c_k. The next theorem of (Boyd, 2000) estimates the error:

THEOREM 2. *Let y have at least N derivatives on dom. Then*

$$y(x) - p_{N-1}(x) = \frac{1}{N!}y^{(N)}(\xi)\prod_{k=1}^{N}(x - x_k)$$

for some ξ on the interval spanned by x and the interpolation points. The point ξ depends on the function y, upon N, upon x and upon the location of the interpolation points.

Consequently, the optimal interpolation points are the roots of the Chebyshev polynomial $T_N(x)$, (**Chebyshev points of the first kind**)

$$x_k = -\cos\frac{(2k-1)\pi}{2N},\ k = 1, ..., N. \tag{7}$$

For these points **x**, the polynomials $\{p_{N-1}\}$ are generally nearly as good approximations to y as the polynomials $\{y_{N-1}\}$ and if y is *analytic* on *dom*, then both $\|y - y_{N-1}\|$ and $\|y - p_{N-1}\|$ decrease geometrically as $N \to \infty$. This is the *spectral convergence property*, i.e. the convergence of $\|y - y_{N-1}\|$ and $\|y - p_{N-1}\|$ towards zero is faster than any power of $\frac{1}{N}$ as $N \to \infty$.

Numerical integration and Lagrangian interpolation are very closely related. The standard formulas for a continuous function f on $[-1, 1]$ are of the type

$$\int_{-1}^{1} f(x)dx \approx \sum_{k=1}^{N} w_k f(x_k), \tag{8}$$

where w_k are the quadrature weights

$$w_k = \int_{-1}^{1}\prod_{j=1, j\neq k}^{N}\frac{x - x_j}{x_k - x_j}dx,\ k = 1, 2, ..., N.$$

The *Gauss quadrature formulas* are based on the optimal Legendre points $x_k,\ k = 1, ..., N$ and these formulas are exact for polynomials f up to degree $2N-1$. The idea of *Clenshaw-Curtis quadrature* is to use Chebyshev points $\mathbf{x}$ instead of the above optimal nodes. By using the Chebyshev points of the first kind (7) one obtains the "classical" Clenshaw-Curtis formula while by using the zeros of the first derivative of a Chebyshev polynomial plus the endpoints ± 1, i.e. the Chebyshev extrema

$$x_k = -\cos\frac{(k-1)\pi}{N-1}, k = 1, ..., N \tag{9}$$

in $[-1, 1]$ (the so called **Chebyshev points of the second kind**) one obtains the "practical" Clenshaw-Curtis formula. Both formulas have all the good properties of the Gaussian quadrature, see (Trefethen, 2008) for more details.
Consequently, we may use Chebyshev points of the first kind or of the second kind both for quadrature formulas and for physical representation of a function y on $[-1, 1]$. Any interval $[a, b]$ may be scaled to $[-1, 1]$ and we obtain the corresponding formulas. Moreover, by using the mapping $x = \cos\theta$ and $T_k(x) = \cos k\theta$ we see that the following two series

$$y(x) = \frac{c_0}{2}T_0(x) + c_1T_1(x) + c_2T_2(x) + ...$$
$$y(\cos\theta) = \frac{c_0}{2} + c_1\cos\theta + c_2\cos 2\theta + ...$$

are equivalent. A Chebyshev series is in fact a Fourier cosine series so that the FFT and iFFT may be used to transform the spectral representation of y into the physical one and conversely, the physical representation into the spectral representation. The quadrature weights $\mathbf{w}$ could also be calculated by a fast algorithm given in (Waldvogel, 2006).
The first code of level0, inspired from `chebpts.m` of *chebfun* (Trefethen et al., 2011) is

```
[x,w] = pd(N,dom,kind)
```

(`pd` means "**p**hysical **d**omain"). It calculates the grid $\mathbf{x} = [x_1, ..., x_N]^T$ (column vector) and the quadrature weights $\mathbf{w} = [w_1, ..., w_N]$ (row vector) for the quadrature formula

$$\int_a^b f(x)dx \approx \sum_{j=1}^{N} w_j f(x_j) \equiv \mathbf{w}\cdot f(\mathbf{x}).$$

The input parameters are N – the dimension of the vectors $\mathbf{x}$ and $\mathbf{w}$, *dom* – the computational domain $[a, b]$ and *kind* which can be 1 or 2 in order to calculate $\mathbf{x}$ as the Chebyshev points of the first or of the second kind.
Some short tests show the performances of this code. Let us approximate

$$\int_0^1 x\sin\frac{1}{x}dx = \frac{cos(1) + sin(1) + Si(1)}{2} - \frac{\pi}{4} \approx 0.37853001712416130988...$$

for $N = 10, 10^2, ..., 10^6$. Here we use Chebyshev points of the first kind and hence we have no problems with the singularity at the origin. We have instead problems with the highly oscillatory behavior of the integrand near the origin. The code is `Chebpack\examples\ex_level0\quad_ex1.m` and the result for $N = 10^6$ is

```
Elapsed time = 0.938258074699438 seconds
err = 1.9568e-11.
```

A more efficient code is

```
[int,gridpts] = quadcheb(myfun,n,dom,kind,tol,gridpts,I)
```

in the folder `Chebpack\examples\ex_level0` which uses `pd` into a recursive procedure. Precisely, starting from the initial interval $dom = [a,b]$, `pd` is used with n points in $[a,b]$ and on two subintervals $[a,c]$ and $[c,b]$ where $c = (a+b)/2$. If the results differe by more than a tolerance ε, the interval $[a,b]$ is divided to that subintervals . Now `quadcheb` is called again for each subinterval and at each step we sum the results. For $N = 128$ we obtain

```
Elapsed time = 0.013672 seconds.
err = 4.907093620332148e-010
```

Of course, this non-optimized quadrature calculation is only a collateral facility in *Chebpack* and it does not work better than the basic quadrature command `quadgk` from MATLAB, which is designated for highly oscillatory integrals.
The next codes of level0

```
v = t2x(c,kind) and c = x2t(v,kind)
```

are inspired by `chebpolyval.m` and `chebpoly.m` from *chebfun* (Trefethen et al., 2011). These codes perform the correspondence between the spectral representation $\mathbf{c}$ of a function f and its physical representation $\mathbf{v} = f(\mathbf{x})$ on Chebyshev points of the first or second kind. It is important to remark that linear operators are better represented in the spectral space, while the nonlinear operators are easily handled in the physical space.
In `t2x` and `x2t`, $\mathbf{c}$ and $\mathbf{v}$ are matrices of the same dimension, each column represents the coefficients or the values for some other function, the number of rows is the above dimension N, while *kind* specifies the type of Chebyshev points used in $\mathbf{v}$. For example, the code

```
n=16;dom=[0,1];kind=2;x=pd(n,dom,kind);
vs=sin(x);vc=cos(x);ve=exp(x);c=x2t([vs,vc,ve],kind);
```

gives in the columns of $\mathbf{c}$ the coefficients of the Chebyshev series (6) of $\sin(x)$, $\cos(x)$ and $\exp(x)$ calculated by using the values of these functions on Chebyshev points of the second kind on $[0,1]$. We remark here that, taking into account the term $c_0 T_0/2$, the coefficient c_0 is doubled.
Another code from level0, inspired from `bary.m` of *chebfun* (Trefethen et al., 2011) and useful for graphical representation of the functions is

```
fxn = barycheb(xn,fk,xk,kind)
```

It interpolates the values $\mathbf{fk}$ of a function f at the Chebyshev nodes $\mathbf{xk}$ of the first or second kind in *dom* by calculating the values $\mathbf{fxn}$ at the new (arbitrary) nodes $\mathbf{xn}$ in *dom*. The barycentric weights are calculated depending on kind.
Precisely, cf. (Berrut & Trefethen, 2004), the barycentric formula is

$$f(x) = \frac{\sum_{k=1}^{N} \frac{w_k}{x-x_k} f_k}{\sum_{k=1}^{N} \frac{w_k}{x-x_k}}, \quad w_k = \frac{1}{\prod_{j \neq k} (x_k - x_j)}, \; k = 1, ..., N.$$

For Chebyshev points one can give explicit formula for barycentric weights $\mathbf{w}$. For the Chebyshev points of the first kind we have

$$x_k = -\cos\frac{(2k-1)\pi}{2N}, \; w_k = (-1)^k \sin\frac{(2k-1)\,\pi}{2N}, \; k = 1, ..., N$$

and for the Chebyshev points of the second kind we have

$$x_k = -\cos\frac{(k-1)\pi}{N-1},\ w_k = (-1)^k\delta_k,\ \delta_k = \begin{cases} \frac{1}{2}, & k = 1, N \\ 1, & \text{otherwise} \end{cases},\ k = 1, ..., N.$$

We remark that for a general interval $dom = [a, b]$ and if the sign changes for all x_k and w_k the weights must be multiplicated by $\pm 2^{N-1}(b-a)^{1-N}$. This factor cancels out in the barycentric formula so that it is no need to include it.
Let us calculate now the differentiation matrix D such that if $\mathbf{f}$ is the column of the Chebyshev coefficients of a function f, then $D\mathbf{f}$ is the column of the Chebyshev coefficients of the derivative function $\frac{df}{dx}$. On $[-1, 1]$ the derivatives of T_i satisfy

$$T_0 = T_1',\ T_1 = \frac{T_2'}{4}, ..., T_i = \frac{T_{i+1}'}{2(i+1)} - \frac{T_{i-1}'}{2(i-1)},\quad i \geq 2$$

from where

$$\frac{T_0'}{2} = 0,\ T_i' = 2i\left(T_{i-1} + T_{i-3} + ... + T_1\right),\quad i\ even$$
$$T_i' = 2i\left(T_{i-1} + T_{i-3} + ... + 0.5T_0\right),\quad i\ odd.$$

Consequently, D is a sparse upper triangular matrix with

$$D_{ii} = 0,\ D_{ij} = 0 \text{ for } (j-i) \text{ even and } D_{ij} = 2j \text{ otherwise.}$$

Of course, the differentiation could be iterated, i.e. the coefficients of $f^{(p)}$ are $D^p\mathbf{f}$. The corresponding code from level0 is

```
D=deriv(n,dom)
```

where n is the dimension of the matrix D. For $dom = [a, b]$ the above matrix D is multiplied by $2/(b-a)$.
Similarly, the code

```
[J,J0]=prim(n,dom)
```

calculates the sparse integration matrix J such that the coefficients of $\int^x f(t)dt$ are $J\mathbf{f}$. *Here the first coefficient of the result* $J\mathbf{f}$ *may be changed in order to obtain the coefficients for a specific primitive of* f. For example, the coefficients of the primitive which vanishes at $a = dom(1)$ are obtained by using $J_0\mathbf{f}$.
The basic formulas for $dom = [-1, 1]$ are

$$\int \frac{T_0}{2}dx = \frac{T_1}{2},\ \int T_1 dx = \frac{T_0/2}{2} + \frac{T_2}{4},\ \int T_k dx = \frac{1}{2}\left(\frac{T_{k+1}}{k+1} - \frac{T_{k-1}}{k-1}\right),\ k \geq 2$$

from where

$$J_{k,k} = 0,\ J_{0,1} = \frac{1}{2},\ J_{k,k-1} = \frac{1}{2k} = -J_{k,k+1},\ k = 1, 2, ...$$

For a general $dom = [a, b]$ the above matrix J is multiplied by $(b-a)/2$.

As an important example, let us calculate the coeficients of a specific primitive $F(x)$ of the function $f(x)$. We must then solve the initial-value problem

$$\frac{dF}{dx} = f(x),\ y(-1) = \alpha,\ x \in [-1,1].$$

If $\mathbf{c}$ are the Chebyshev coefficients of F and $\mathbf{f}$ are the coefficients of f, the equation is discretized in spectral space as $D\mathbf{c} = \mathbf{f}$. In order to implement the initial condition, we remark that

$$y(-1) = c_0\frac{T_0}{2} + c_1T_1(-1) + c_2T_2(-1) + ... + c_{N-1}T_{N-1}(-1) = \alpha$$

can be written as $T\mathbf{c} = \alpha$ where

$$T = \left[\frac{T_0}{2}, T_1(-1), T_2(-1), ..., T_{N-1}(-1)\right].$$

This means that we can replace the last row of D by T and the last entry of $\mathbf{f}$ by α, thus obtaining a new matrix $\widetilde{D}$ and a new vector $\widetilde{f}$. Finally, $\mathbf{c} = \widetilde{D}^{-1}\widetilde{f}$ are the coefficients of the specific primitive.
The following code from level0

```
T=cpv(n,xc,dom)
```

(**c**hebyshev **p**olynomial **v**alues) implements such conditions. Here $\mathbf{x}_c$ is an arbitrary vector in $dom = [a,b]$ and `cpv` calculates the values of the Chebyshev polynomials T_k, $k = 0,1,..,n-1$ at the column of nodes ξ

$$T = [T_0/2, T_1(\xi), T_2(\xi), ..., T_{N-1}(\xi)],\ \xi = \frac{2\mathbf{x}_c}{b-a} - \frac{b+a}{b-a},\ 1 \le \xi \le 1.$$

The code is based on the recurrence formulas of Chebyshev polynomials on $[-1,1]$

$$T_0(x) = 1,\ T_1(x) = x,\ T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x),\ k \ge 2.$$

The test code `Chebpack\examples\ex_level0\quad_ex3` performs these calculations for the special case $y' = \cos x$, $y(0) = 0$, with the solution $y = \sin(x)$. The coefficients $\mathbf{c}$ are obtained by using the differentiation matrix, **cc** are the coefficients of the exact solution, **ccc** are obtained by using the integration matrix J and **cccc** are obtained by using the integration matrix J_0.
We also remark that if `T=cpv(n,x,dom)`,

$$f(x) = 0.5c_0T_0^*(x) + c_1T_1^*(x) + ... + c_{N-1}T_{N-1}^*(x) \tag{10}$$

and $\mathbf{c} = (c_0, ..., c_{N-1})^T$, we have $f(\mathbf{x}) = T\mathbf{c}$, for $\mathbf{x} \in [a,b]$. The code `cpv` could be used to transform between the spectral representation $\mathbf{f}$ of the function f and the physical representation $\mathbf{v} = f(\mathbf{x})$ of values at the Chebyshev grid $\mathbf{x}$,

$$\mathbf{v} = T\mathbf{f},\ \mathbf{f} = T^{-1}\mathbf{v}.$$

These transforms are performed by FFT in the codes `x2t` and `t2x`, but for a small dimension N we may use this direct matrix multiplication.

As another example, let us calculate the values at the grid points $\mathbf{x}$ of the specific primitive which vanishes at $a = dom(1)$

$$F(x_i) = \int_a^{x_i} f(t)dt, \ \ i = 1, ..., n$$

Starting with the values $\mathbf{f} = f(\mathbf{x})$ we have the Chebyshev coefficients $T^{-1}\mathbf{f}$, then $J_0T^{-1}\mathbf{f}$ are the Chebyshev coefficients of the specific primitive on $[a, b]$ and finally,

$$F(\mathbf{x}) = TJ_0T^{-1}\mathbf{f}. \tag{11}$$

Another code from level0

```
X=mult(n,dom)
```

calculates the sparse multiplication matrix X such that if $\mathbf{f}$ is the column vector of the Chebyshev coefficients of a function $f(x)$, then $X\mathbf{f}$ is the column vector of the coefficients of the function $xf(x)$. The code is based on the formulas

$$xT_0 = T_1, \ \ xT_1(x) = \frac{T_0}{2} + \frac{T_2}{2}, ..., \ xT_k(x) = \frac{T_{k-1}(x)}{2} + \frac{T_{k+1}(x)}{2}, \ \ k \geq 2$$

for $x \in [-1, 1]$. Consequently,

$$X_{k,k} = 0, \ X_{k,k-1} = X_{k,k+1} = \frac{1}{2}, \ k = 2, 3, ..., N-1,$$
$$X_{1,1} = 0, \ X_{1,2} = 1, \ X_{N,N-1} = \frac{1}{2}.$$

Then, in general, the coefficients of $x^p f(x)$ are given by $X^p\mathbf{f}$ and the coefficients of $a(x)f(x)$ are given by $a(X)\mathbf{f}$ for analytical functions $a(x)$, where $a(X)$ is the matricial version of the function a. Moreover, if $\frac{f(x)}{x^p}$ has no singularity at the origin, then its coefficients are $X^{-p}\mathbf{f}$. Of course, X is a tri-diagonal matrix, X^2 is a penta-diagonal matrix and so on but, generally, the matrix version `funm(full(X))` of the scalar function $a(x)$ or $X^{-p} = [inv(X)]^p$ are not sparse matrices. For a general interval $dom = [a, b]$, X is replaced by $\frac{b-a}{2}X + \frac{b+a}{2}I_N$ where I_N is the sparse unit matrix `speye(N)`.
Another method to calculate $a(X)$ is to pass from the values $a(\mathbf{x})$ at the Chebyshev grid $\mathbf{x}$ to the Chebyshev coefficients $\mathbf{a}$ using `x2t` and to approximate

$$a(x) \approx .\frac{a_0}{2} + \sum_{k=1}^{m-1} a_k T_k(x). \tag{12}$$

Here m must be chosen sufficiently large, but $m \leq N$ so that the known function $a(x)$ is correctly represented by $a_0, a_1, ..., a_{m-1}$.
In order to calculate the coefficients of the product

$$a(x)f(x) = \left(\frac{a_0}{2} + \sum_{k=1}^{m-1} a_k T_k(x)\right)\left(\frac{f_0}{2} + \sum_{j=1}^{n-1} f_j T_j(x)\right)$$

we may use the formula

$$T_j(x)T_k(x) = \frac{T_{j+k}(x) + T_{|j-k|}(x)}{2}, \ \forall j,k.$$

The needed coefficients are given by $A\mathbf{f}$ where the matrix $A \approx a(X)$ is given by the code

```
A=fact(a,m)
```

from level0.
The test code `Chebpack\examples\ex_level0\fact_ex1.m` calculates `cosh(X)` using `fact` and `funm` from Matlab. We remark that X is a sparse matrix, so that `funm` must be applied to `full(X)`.

3. Chebpack, linear module

At the first level – level1, the package contains subroutines to solve
- initial and boundary value problems for high order linear differential equations
- initial value problems for first order linear differential systems
- linear integral equations
- eigenvalues and eigenfunctions for differential problems.

The main method used is the so called tau-method, see (Mason & Handscomb, 2003) or (Canuto et al., 1988) for more theoretical details and the implementation. It is based on
- discretization using the differentiation matrix D
- discretization using the integration matrix J
- splitting the interval $dom = [d_1, d_2, ..., d_p]$.

3.1 Discretization using the differentiation matrix D

The corresponding code is `Chebpack\level1\ibvp_ode.m`

```
[x,solnum]=ibvp_ode(n,dom,kind)
```

where *n*, *dom*, *kind* have the same significance as above, **x** is the Chebyshev grid and **solnum** is the numerical solution in the physical space calculated at the grid **x**.
The structure of `ivp_ode` is

```
function [x,solnum]=ibvp_ode(n,dom,kind)
x=pd(n,dom,kind);X=mult(n,dom);D=deriv(n,dom);
myDE;myBC;sol=A\b;solnum = t2x(sol,kind);myOUT;
end
```

where `myDE`, `myBC` and `myOUT` must be written by the user and describe the differential equation, the boundary conditions and the output of the program.
For example, for the problem

$$y''' - xy = (x^3 - 2x^2 - 5x - 3)e^x, \ x \in [0,1]$$
$$y(0) = 0, \ y'(0) = 1, \ y(1) = 0$$

we have

```
function myDE
A=D^3-X;b=x2t((x.^3-2*x.^2-5*x-3).*exp(x),kind);end
function myBC
T=cpv(n,dom,dom);A(n-2,:)=T(1,:);b(n-2)=0;
```

`A(n-1,:)=T(1,:)*D;b(n-1)=1;A(n,:)=T(2,:);b(n)=0;end`
The program is called by
`[x,solnum]=ibvp_ode_test(32,[0 1],2);`
Other examples are coded in `Chebpack\examples\ex_level1\ibvp_ode_ex*.m`
The general form for initial/boundary value problems for high order linear differential equations is (1) and its discrete form is

$$A\mathbf{c} \equiv \left(\sum_{k=0}^{m} P_k(X) D^k \right) \mathbf{c} = \mathbf{b}$$

where the unknown y is represented in the spectral space by its Chebyshev coefficients $\mathbf{c}$, while $\mathbf{b}$ are the Chebyshev coefficients of the r.h.s. $f(x)$.
We remark that the coefficients $P_k(X)$ of the equation can be defined in `myDE`
- directly, for example $P_k(x) = -x$ gives $P_k(X) = -X$
- using $funm$, for example $P_k(x) = \sin x$ gives $P_k(X) = funm(full(X), @sin)$, i.e. using the Taylor series of $\sin X$
- using $fact$, for example $P_k(x) = \cos x$ gives $P_k(X) = fact(x2t(cos(x), kind), m)$, i.e. using the Chebyshev series of $\cos X$
- if $P_k(x)$ is a constant, say P_k, then $P_k(X) = P_k \cdot speye(N)$.
The boundary conditions of the general type (2) are implemented using `cpv`. For example, for $y(x_{1c}) - y'(x_{2c}) + 2y''(x_{3c}) = y_c$ we calculate `T=cpv(N,[x1c,x2c,x3c],dom)`. One of the last rows of A is replaced by `T(1,:)-T(2,:)*D+2*T(3,:)*D^2` and the corresponding entry of the vector $\mathbf{b}$ is replaced by y_c.

3.2 Discretization using the integration matrix J

The corresponding code is `Chebpack\level1\ibvp_ode_int.m`

```
[x,solnum]=ibvp_ode_int(n,dom,kind)
```

We remark that the discretization using the differentiation matrix D does not work well for large N. For example, this type of discretization for the problem

$$\varepsilon y'' + xy\prime = -\varepsilon\pi^2 cos(\pi x) - \pi\, x\, sin(\pi x),\ x \in [-1,1]\,, \varepsilon = 10^{-5}, \qquad (13)$$
$$y(-1) = -2,\ y(1) = 0$$

with $N = 2048$ and an error about 5.46×10^{-11} leads us to a sparse system $A\mathbf{c} = \mathbf{b}$ but with a sparsity factor about 25% that increases the computational time to 6.4 sec, see the example `ibvp_ode_ex.m` in the folder `Chebpack\examples\ex_level1`. A better idea is to integrate two times the above equation using the much more sparse integration matrix J. This integration make the coefficients c_0 and c_1 arbitrary and we may fix their values by using the boundary conditions, this time at the first two rows of A and $\mathbf{b}$.
Precisely, the first and the second integration of the equation (13) gives

$$\varepsilon y' + xy - \int^x y = \int^x \left[-\varepsilon\pi^2 cos(\pi x) - \pi\, x\, sin(\pi x) \right]$$
$$\varepsilon y + \int^x xy - \int^x \int^x y = \int^x \int^x \left[-\varepsilon\pi^2 cos(\pi x) - \pi\, x\, sin(\pi x) \right].$$

The discrete form is $\left(\varepsilon I_N + JX - J^2\right)\mathbf{c} = J^2\mathbf{f}$ where $\mathbf{c}$ are the Chebyshev coefficients of the solution y and $\mathbf{f}$ are the Chebyshev coefficients of the r.h.s. The new code

`ibvp_ode_int_ex.m` in the same folder as above gives the same accuracy for the same $N = 2048$, but needing only 0.12 sec. The new matrix A has now a sparsity factor of about 0.2439% for the dimension 2048.
This higher sparsity qualifies the integration method to be used for splitting the interval $dom = [a, b]$ into $dom = [a, d_1] \cup [d_1, d_2] \cup ... \cup [d_{m-1}, b]$ as well as for differential systems, where the dimension N of matrices is multiplied by the number of subintervals or by the number of differential equations in the system.
We give the formulas for first order equations and a general formula. For the first order we have in `myDE`

$$P_1(x)y' + P_0(x)y = F \implies P_1(x)y - \int P_1'(x)y + \int P_0(x)y = \int F,$$

$$A\mathbf{c} \equiv \left[P_1(X) + J\left[P_0(X) - P_1'(X)\right]\right]\mathbf{c} = J\mathbf{f} \equiv \mathbf{b}.$$

Generally, if we denote the differentiation operator on functions P by dP, the identity operator by I and the formal k power of the operator $I - Jd$ by $[\ ]^{(k)}$, we obtain, after m integrations, $A\mathbf{c} = \mathbf{b}$ where

$$A = \sum_{k=0}^{m} J^{m-k}\left[I - Jd\right]^{(k)} P_k(X),\ \mathbf{b} = J^m\mathbf{f}. \tag{14}$$

For example, for $m = 3$ we have

$$\iiint P_0(x)y(x) \to J^3P_0(X)\mathbf{c},\ \iiint P_1(x)y'(x) \to J^2\left[P_1(X) - JP_1'(X)\right]\mathbf{c},$$

$$\iiint P_2(x)y''(x) \to J\left[P_2(X) - 2JP_2'(X) + J^2P''(X)\right]\mathbf{c},$$

$$\iiint P_3(x)y'''(x) \to \left[P_3(X) - 3JP_3'(X) + 3J^2P_3''(X) - J^3P_3'''(X)\right]\mathbf{c}.$$

It is important to remark that the absolute value of the Chebyshev coefficients gives us some information about the necessary dimension N of the discretized problem in order to capture the correct behavior of the solution. For example, let us consider the problem `ibvp_ode_int_ex2.m` in the folder `Chebpack\examples\ex_level1`

$$\varepsilon y'' + (x^2 - w^2)y = 0,\ y(-1) = 1, y(1) = 2,\ x \in [-1, 1]$$

for $w = 0.5$, $\varepsilon = 1.e - 6$. The command
`[x,solnum]=ibvp_ode_int_ex2(1024,[-1 1],2);`
gives the results in Fig. 2. We see that up to a dimension about $N = 400$, the algorithm cannot resolve y accurately, due to its highly oscillatory behavior. After that, the Chebyshev series begins to converge rapidly. For $N = 1024$ the elapsed time is about 0.05 sec.

3.3 Discretization using the integration matrix J and splitting the interval

The corresponding code is `Chebpack\level1\ibvp_ode_split.m`

```
[x,solnum]=ibvp_ode_split(n,dom,kind)
```

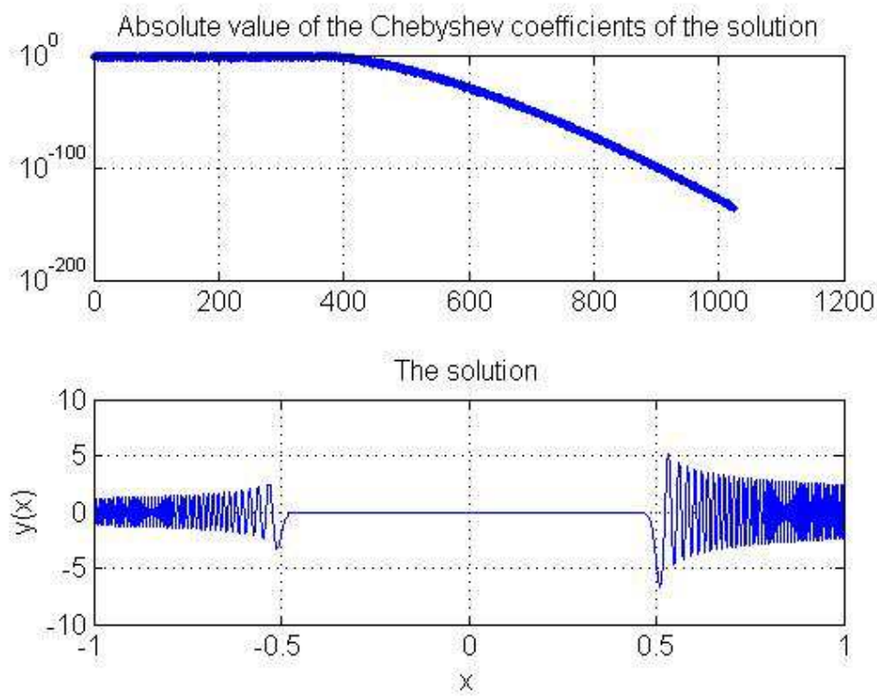

Fig. 2. The Chebyshev coefficients and the numerical solution for ex2

Sometimes, the solution of a differential problem has a different behavior in different subintervals. For example, for small ε the solution of the problem (13) has a shock near the origin and we need a very large N in order to capture its correct behavior. In these cases it is better to split the working interval $dom[a,b]$ into disjoint subintervals $[d_1,d_2]\cup[d_2,d_3]\cup ...\cup [d_{p-1},d_p]=[a,b]$ adapted to the behavior of the solution. The great advantage is to use a small N for each subinterval. The partial solutions on each subinterval are connected by some level of smoothness.

Precisely, let us consider for example a second order differential problem on $[a,b]$ and let us split the interval as $[a,b]=[d_1,d_2]\cup[d_2,d_3]\cup[d_3,d_4]$. This splitting is given by $dom=[d_1,d_2,d_3,d_4]$ on input. If we calculate the basic ingredients **xs**, Xs, Ds, Js for the standard interval $[-1,1]$, then for each subinterval $[d_i,d_{i+1}]$, $i=1,2,3$ the corresponding ingredients become

$$\mathbf{x}=len*\mathbf{xs}+med,\ X=len*Xs+med*I_N,\ D=\frac{Ds}{len},\ J=len*Js$$

where $len=\frac{d_{i+1}-d_i}{2}$, $med=\frac{d_{i+1}+d_i}{2}$.

Using the matrix J for the discretization we obtain on each subinterval $i=1,2,3$ the discretized form $A^{(i)}c^{(i)}=b^{(i)}$ where the matrix $A^{(i)}$ and the vector $\mathbf{b}^{(i)}$ are given by (14) as above, while $\mathbf{c}^{(i)}$ are the Chebyshev coefficients of the solution $y^{(i)}$ on that subinterval i. Now, using the Kronecker product and the `reshape` command of Matlab, we build a large (but very sparse) system $A\mathbf{c}=\mathbf{b}$

$$\begin{pmatrix} A^{(1)} & O & O \\ O & A^{(2)} & O \\ O & O & A^{(3)} \end{pmatrix}\begin{pmatrix} \mathbf{c}^{(1)} \\ \mathbf{c}^{(2)} \\ \mathbf{c}^{(3)} \end{pmatrix}=\begin{pmatrix} \mathbf{b}^{(1)} \\ \mathbf{b}^{(2)} \\ \mathbf{b}^{(3)} \end{pmatrix}.$$

The boundary conditions are now the given boundary conditions say at d_1 and d_4 supplemented by smoothness conditions at d_2, d_3

$$y^{(1)}(d_2-0)=y^{(2)}(d_2+0),\ \frac{dy^{(1)}}{dx}(d_2-0)=\frac{dy^{(2)}}{dx}(d_2+0), \tag{15}$$
$$y^{(2)}(d_3-0)=y^{(3)}(d_3+0),\ \frac{dy^{(2)}}{dx}(d_3-0)=\frac{dy^{(3)}}{dx}(d_3+0).$$

Of course, for a higher order equation (say m) we have coincidence conditions (15) until the derivatives of order $m-1$. The given boundary conditions are implemented in the first m rows of the first block-row of the matrix A and in the first entries of the first block of the vector $\mathbf{b}$, while the coincidence conditions are implemented in the first m rows of each of the following block-rows of A and in the first m entries of each following block of $\mathbf{b}$. The sparsity structure of A with 4 subintervals is given in Fig. 3. Here we have 16 blocks of size 64×64, the

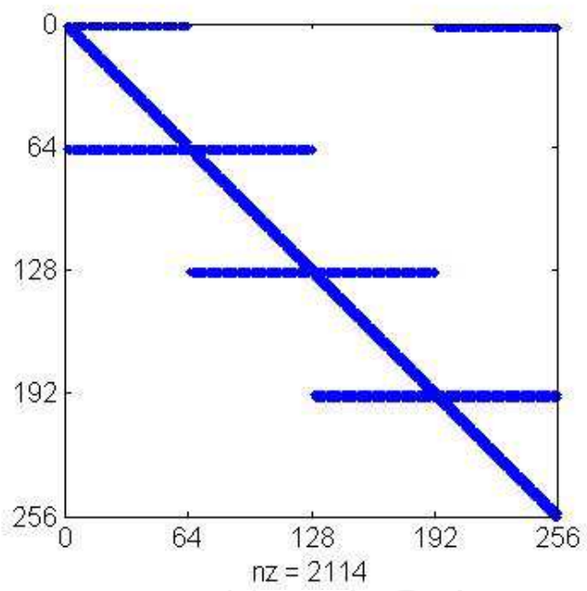

Fig. 3. The sparsity structure of the matrix A with boundary conditions implemented for 4 subintervals

4 diagonal segments come from the matrices $A^{(i)}$, $i=1,2,3,4$, the first horizontal segments come from the given boundary conditions while the next 3 pairs of horizontal segments come from connectivity conditions. Each block acts on the corresponding block coefficients $\mathbf{c}^{(i)}$, $i=1,2,3,4$.

Using this technique for the problem (13) for example, the command

```
[x,solnum]=ibvp_ode_split_ex(64,[-1 -0.05 0 0.05 1],2);
```

from the folder `Chebpack\examples\ex_level1` gives the numerical solution with an accuracy of about 6×10^{-15} with four subintervals with $N=64$ in only 0.014 sec. The new matrix A has now a sparsity factor of about 3.2257% and the dimension 256.

3.4 Linear first-order systems

Let us consider a first order linear differential system for $x \in [a,b]$

$$P_1(x)y_1' + P_{11}(x)y_1 + ... + P_{1m}(x)y_m = f_1(x),\ y_1(a) = y_{1a},$$
$$P_2(x)y_2' + P_{21}(x)y_1 + ... + P_{2m}(x)y_m = f_2(x)\ ,\ y_2(a) = y_{2a},$$
$$.....$$
$$P_m(x)y_m' + P_{m1}(x)y_1 + ... + P_{mm}(x)y_m = f_m(x),\ y_m(a) = y_{ma}.$$

If we denote by $\mathbf{c} = (c^{(1)},...,c^{(m)})^T$ the Chebyshev coefficients of $y_1(x),...,y_m(x)$ and by $\mathbf{f}^{(1)},...,\mathbf{f}^{(m)}$ the coefficients of $f_1,...,f_m$ then the discretized version of the system is $\mathbf{Ac} = \mathbf{b}$ where

$$\mathbf{A} = \begin{pmatrix} P_1(X)D + P_{11}(X) & \cdots & P_{1m}(X) \\ \vdots & \ddots & \vdots \\ P_{m1}(X) & \cdots & P_m(X)D + P_{mm}(X) \end{pmatrix},\ \mathbf{b} = \begin{pmatrix} \mathbf{f}^{(1)} \\ \vdots \\ \mathbf{f}^{(m)} \end{pmatrix}.$$

The initial conditions are implemented like in (15): the last row of each block in the above matrix is replaced by T or zeros for the corresponding columns and the last entry of each block in the r.h.s. is replaced by y_{ka} such that $[O,...,T,...,O]\mathbf{c} = y_{ka}$ for each k. The corresponding code from level1 is

```
[x,solnum]=ibvp_sys(n,dom,kind,y0)
```

where **y0** is the column vector of the initial values $[y_{1a},...,y_{ma}]^T$. Of course, we may use the integration matrix J instead of D for discretization, obtaining again a system $\mathbf{Ac} = \mathbf{b}$ where

$$\mathbf{A} = \begin{pmatrix} P_1(X) - JP_1'(X) + JP_{11}(X) & \cdots & JP_{1m}(X) \\ \vdots & \ddots & \vdots \\ JP_{m1}(X) & \cdots & P_m(X) - JP_m'(X) + JP_{mm}(X) \end{pmatrix},\ \mathbf{b} = \begin{pmatrix} J\mathbf{f}^{(1)} \\ \vdots \\ J\mathbf{f}^{(m)} \end{pmatrix}$$

with the implementation of the initial conditions on the first row of each block, see `ibvp_sys_ex3_int.m`, or we may consider systems of higher order, see `ibvp_sys_ex2x2.m` from the folder `Chebpack\examples\ex_level1`.

3.5 Linear integral equations

Let us consider a Fredholm integral equation

$$y(x) = \int_a^b K(x,t)y(t)dt + f(x) \equiv A(y)(x) + f(x),\ x \in [a,b]. \tag{16}$$

The Fredholm integral operator $A(y)$ becomes after discretization with shifted Chebyshev polynomials

$$A(y)(x) = \int_a^b K(x,t) \sum_{k=0}^{N-1} c_k T_k^*(t)dt = \sum_{k=0}^{N-1} c_k \int_a^b K(x,t)T_k^*(t)dt =$$
$$\sum_{k=0}^{N-1} c_k \cdot I_k(x) = \sum_{k=0}^{N-1} c_k \sum_{j=0}^{N-1} k_{jk}T_j^*(x) = \sum_{j=0}^{N-1} \left[\sum_{k=0}^{N-1} k_{jk}c_k\right] T_j^*(x).$$

Consequently, if $\mathbf{c} = (c_0, ..., c_{N-1})^T$ are the coefficients of y, then $K\mathbf{c}$ are the coefficients of $A(y)$, given by the matrix $K = (k_{jk})_{j,k=0,...,N-1}$.
The matrix K can be calculated starting from the physical values

$$I_k(x_s) = \int_a^b K(x_s, t) T_k^*(t) dt = \sum_{i=0}^{N-1} w_i K(x_s, x_i) T_k^*(x_i), \quad s, k = 0, ..., N-1.$$

In matrix form, this means

$$(I_k(x_s))_{k,s=0,...,N-1} = (K(x_s, x_i))_{s,i=0,...,N-1} \cdot diag((w_i)_{i=0,...,N-1}) \cdot T$$

where `T=cpv(n,x,dom)` and then we apply `x2t`, see also (Driscoll, 2010).
The Fredholm integral equation (16) becomes $(I_N - K)\mathbf{c} = \mathbf{f}$ where $\mathbf{f}$ are the Chebyshev coefficients of f and we obtain the solution by solving this linear system. The corresponding model code from the folder `Chebpack\level1` is

```
[x,solnum]=fred_eq(n,dom,kind)
```

Similarly, for a Volterra integral equation

$$y(x) = \int_a^x K(x, t) y(t) dt + f(x) \equiv A(y)(x) + f(x), \; x \in [a, b]$$

we obtain, using (11), for the Volterra integral operator

$$A(y)(x) = \int_a^x K(x, t) y(t) dt = \left[T J_0 T^{-1} .* K(x_i, x_j) \right] \mathbf{y},$$

where $\mathbf{y}$ are the values of $y(x)$ at the grid points $\mathbf{x}$. Consequently, the physical representation of the Volterra integral operator is the matrix $V_{phys} = T J_0 T^{-1} .* K(x_i, x_j)$, see again (Driscoll, 2010) while its spectral representation is $V_{spec} = T^{-1} V_{phys} T$. The Volterra integral equation becomes in physical representation $(I_N - K)\mathbf{y} = \mathbf{f}$ where $\mathbf{f}$ are now the values of f at the grid points $\mathbf{x}$.
The corresponding model code from *Chebpack* is

```
[x,solnum]=volt_eq(n,dom,kind)
```

from the folder `Chebpack\level1`. The folder `Chebpack\examples\ex_level1` contains some examples in the files `fred_eq_ex*` and `volt_eq_ex*`.
Of course, these codes work well only if the kernel $K(x, t)$ is sufficiently smooth such that it can be spectrally represented by an acceptable number of Chebyshev polynomials.

3.6 Eigenvalues/eigenfunctions for Sturm-Liouville problems

Let us consider now the second order spectral problem

$$P_2(x)y'' + P_1(x)y' + P_0(x)y = \lambda R(x)y$$

with homogeneous boundary value conditions as above. Using tau method, we get the following N – dimensional matrix eigenproblem

$$(P_2(X)D^2 + P_1(X)D + P_0(X))\mathbf{c} = \lambda R(X)\mathbf{c}$$

of the form $A\mathbf{c} = \lambda B\mathbf{c}$. The conditions give $T\mathbf{c} = 0$ and combining these equations we derive the generalized eigenproblem

$$\begin{pmatrix} T \\ A \end{pmatrix} c = \lambda \begin{pmatrix} \frac{1}{\lambda^*} T \\ B \end{pmatrix} c \text{ i.e. } \widetilde{A}\mathbf{c} = \lambda \widetilde{B}\mathbf{c}$$

where we retain only the first N rows of the matrices to obtain $\widetilde{A}$ and $\widetilde{B}$. Here λ^* is chosen by the user as a large and known value. We remark that for $\lambda \neq \lambda^*$ we get $Tc = 0$ as above but now the matrix $\widetilde{B}$ behaves well. Consequently, the eigenproblem has instead of two infinite eigenvalues two known λ^* eigenvalues that can be eliminated. The same procedure is applied for higher order problems.
The corresponding model code from the folder `Chebpack\level1` is

```
[lam,phi,x]=eig_ode2(n,dom,kind,numeigval)
```

The folder `Chebpack\examples\ex_level1` contains some other examples in the files `eig_ode2_ex*`, `eig_ode3_ex` and `eig_ode4_ex`. An older package *LiScEig* is also freely accessible at (Trif, 2005).

4. Chebpack, nonlinear module

At the second level – level2, we have subroutines to solve
- initial and boundary value problems for nonlinear differential equations
- nonlinear integral equations.

Here the codes of the first level are used at each step of the Newton method applied in the functional space. Another method could be the successive iteration method.

4.1 Successive iteration method

Let us consider, as an example, the nonlinear Emden boundary value problem

$$xy'' + 2y' = -xy^5,\ y'(0) = 0,\ y(1) = \frac{\sqrt{3}}{2},\ \ x \in [0,1]\,.$$

If the starting spectral approximation of y is, for example, $Y_0 = 1$ then the discretization of the problem is $AY = F$. Here $A = XD^2 + 2D$ and $F = F(x, Y_0)$ are the coefficients of the r.h.s. modified by using the boundary value conditions. We apply successively

$$Y_{n+1} = A^{-1}F(x, Y_n),\ n = 0, 1, 2, ..., n_{\max}.$$

If Y_n converges, then it converges to a solution of the bvp.
The Matlab codes are `ibvp_ode_succapprox.m` from the folder `Chebpack\level2` or `ibvp_ode_ex1.m` from the folder `Chebpack\examples\ex_level2`.
Of course, the discretization could be performed using the integration matrix J instead of D.
Let us consider, for example, the Lotka-Volterra system

$$y_1' - Ay_1 = -By_1y_2,\ y_2' + Cy_2 = By_1y_2,\ t \in [0, 100]$$
$$y_1(0) = y_2(0) = 0.5,\ A = 0.1,\ B = 0.2,\ C = 0.05.$$

We transform this system to integral form

$$y_1 - A\int y_1 dt = -B\int y_1 y_2 dt,\ y_2 + C\int y_2 dt = B\int y_1 y_2 dt$$

and the discretized form in spectral representation is

$$\begin{pmatrix} I_n - AJ & 0 \\ 0 & I_n + CJ \end{pmatrix}\begin{pmatrix} Y_1^{new} \\ Y_2^{new} \end{pmatrix} = \begin{pmatrix} -BJF \\ BJF \end{pmatrix}$$

where F is the column vector of the coefficients of $y_1^{old}(t)y_2^{old}(t)$ obtained by using `t2x.m` and `x2t.m`. In this discretized form we implement the initial conditions as usually and we must solve this system which has a sparsity factor of about 6% for $n = 32$ and about 3% for $n = 64$. For a long interval dom given as $dom = [d_0, d_1, ..., d_m]$, we apply the successive approximations method at each subinterval. The initial approximation for the following subinterval is given by the final values of the solution for the current subinterval. To test the numerical solution, we remark that the Lotka-Volterra system has as invariant $\Lambda = By_1 + By_2 - C\log y_1 - A\log y_2$. The code is used by the command

```
[x,solnum]=ibvp_sys_succapprox(32,[0:10:100],2,[0.5,0.5]);
```

and the result is given in Fig. 4, with the value of the invariant $\Lambda = 0.3039720770839$.

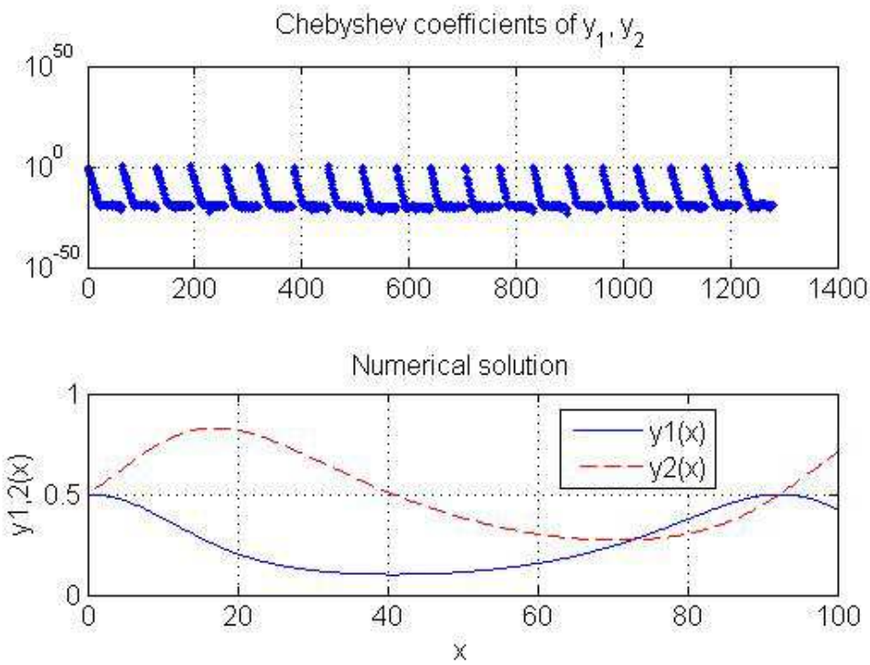

Fig. 4. The solution of the Lotka-Volterra problem

In the case of nonlinear integral equations, for example

$$y(x) = f(x) + \int_0^b K(x, t, y(t))dt,$$

we perform successive iterations (if this method works)

$$y^{new}(x) = f(x) + \int_0^b K(x, t, y^{old}(t))dt$$

starting with a suitable y^{old}. For each $x = x_s$ in the grid and at each iteration, the integral is evaluated as

$$\int_0^b K(x_s, t, y^{old}(t))dt \approx \sum_{k=1}^{n} w_k K(x_s, t_k, y^{old}(t_k)),$$

where $\{t_k, k = 1, ..., n\}$ is the Chebyshev grid on $[0, b]$ and $\{w_k, k = 1, ..., n\}$ are the corresponding weights. Consequently, we obtain

$$y^{new}(x_s) = \sum_{k=1}^{n} w_k K(x_s, t_k, y^{old}(t_k)),\ s = 1, ..., n.$$

Taking into account the nonlinearities, all the calculations are performed into the physical space. Next, $y^{old} \longleftarrow y^{new}$ until $\left\| y^{new} - y^{old} \right\| < \varepsilon$. The corresponding code is `ibvp_int_succapprox.m` from the folder `Chebpack\level2`.

4.2 Newton method

Let us consider again a nonlinear differential problem of the form

$$Ly(x) = f(x, y(x)),\ x \in [a, b]$$

where L is a linear differential operator, such as $Ly(x) = xy''(x) + 2y'(x)$ and f is the nonlinear part. We have also the boundary or initial conditions BC/IC. If we denote by

$$P(y)(x) = Ly(x) - f(x, y(x)),\ BC/IC$$

the problem is of the form $P(y)(x) = 0$ where P is the operator of the problem and y is the unknown.
The Newton method works as follows. Starting with an initial approximation $y_0(x)$ verifying the initial or boundary conditions, we must solve at each step n the linear problem

$$P'(y_n)(y_{n+1} - y_n)(x) = -P(y_n)(x),$$

for $y_{n+1} - y_n$, with the corresponding homogeneous IC/BC. Next, $y_{n+1} = y_n + (y_{n+1} - y_n)$ and we perform these iterations until $\|y_{n+1} - y_n\| < \varepsilon$. For our problem, the linear step is

$$\left[L - \frac{\partial f(x, y_n(x))}{\partial y} \right] (y_{n+1} - y_n)(x) = -\left[Ly_n(x) - f(x, y_n(x)) \right].$$

The corresponding code is `ibvp_ode_newton.m` from the folder `Chebpack \level2`. It starts with $y_0(x)$ verifying or not the IC/BC and solves at each step the above linear problem for y_{n+1} with the nonhomogeneous IC/BC.
A nonlinear system (of order 2 for example)

$$y_1' + a_{11}y_1 + a_{12}y_2 = f_1(x, y_1, y_2),\ x \in [a, b]$$
$$y_2' + a_{21}y_1 + a_{22}y_2 = f_2(x, y_1, y_2),\ y_1(a) = y_{1a}, y_2(a) = y_{2a}$$

is in matrix form

$$Y'(x) + A(x)Y(x) = F(x, Y).$$

At the linear step it becomes

$$\left[Z' + \left[A(x) - JacF(x, Y_n) \right] Z = -\left[Y_n' + AY_n - F(x, Y_n) \right] \right.$$

where $Z = Y_{n+1} - Y_n$ is the correction, $Y_n \equiv Y(x_n)$ and

$$JacF(x, Y_n) = \begin{vmatrix} \frac{\partial f_1(x, y_{1n}, y_{2n})}{\partial y_1} & \frac{\partial f_1(x, y_{1n}, y_{2n})}{\partial y_2} \\ \frac{\partial f_2(x, y_{1n}, y_{2n})}{\partial y_1} & \frac{\partial f_2(x, y_{1n}, y_{2n})}{\partial y_2} \end{vmatrix}.$$

As above, $Y_{n+1} = Y_n + Z$ until $\|Z\| < \varepsilon$.
We remark that in the linear step we use the integration

$$Z + \int [A(x) - JacF(x, Y_n)]\, Z = -\int \left[Y_n' + AY_n - F(x, Y_n)\right]$$

which becomes in the discrete form

$$[I_n + J\,(A(X) - JacF(X, Y_n))]\, Z = -[Y_n + JA(x)]Y_n + JF(x, Y_n).$$

i.e $AZ = B$. In the l.h.s the matrix $J\,(A(X) - JacF(X, Y_n))$ is calculated using the code `fact.m`. This code uses the physical values of $A(x) - JacF(x, Y_n)$ and converts them into spectral coefficients. In the r.h.s the code also starts with the physical values and converts them into their spectral coefficients. The initial conditions are implemented now in the rows $1, n+1, ...$
Of course, this code `ibvp_sys_newton.m` from the folder `Chebpack\level2` can be used in a long-term integration algorithm that starts with the initial values y_{a1}, y_{b1}, calculates the solution on $[a, b]$, extracts the final values y_{b1}, y_{b2} which become initial values for the same code on a new interval $[b, c]$ and so on.
A short comparison between the successive approximations method (SA) and the Newton method (N) for the example from `ibvp_sys_newton_ex1` shows that, for the same $n = 64$, $dom = [0, 1, 2 : 2 : 200]$ and $M = 8$,
- SA: 2910 iterations, 3.4 sec elapsed time, 12940 Chebyshev coefficients calculated
- N : 490 iterations, 6 sec elapsed time, 6475 Chebyshev coefficients calculated.
In the case of a nonlinear integral equation (of Fredholm or Volterra) type

$$P(y) \equiv y(x) - \int_a K(x, t, y(t))dt - f(x) = 0,$$

we start with an initial approximation y_0 (in physical space) and at each Newton step we obtain the linear equation for the correction z

$$z(x) - \int_a \frac{\partial K}{\partial y}(x, t, y_0(t))z(t)dt = -y_0(x) + \int_a K(x, t, y_0(t))dt + f(x).$$

We solve this equation in spectral form as in the previous section and the corrected value of y is $y_0 + z$. We repeat this step until convergence, i.e. until $\|z\| < \varepsilon$. There are many examples in the folder `Chebpack\examples\level2`.

5. Chebpack, experimental module

Finally, the package contains an experimental level – level3, in progress, for
- partial differential equations of evolution type
- fractional differential equations (i.e. differential equations of non-integer order).

5.1 Partial differential evolution equations

Let us consider, as a simple example, a problem from (Trefethen et al., 2011)

$$u_t = u_{xx},\ x \in [-4,2],\ t > 0$$
$$u(x,0) = u^{(0)}(x) \equiv \max(0, 1 - abs(x)),$$
$$u(-4,t) = 1,\ u(2,t) = 2.$$

First, we discretize in time by a backward finite difference on the grid $0 = t_0 < t_1 < ... < t_K$ starting with $u^{(0)}(x)$

$$\frac{u^{(k+1)}(x) - u^{(k)}(x)}{dt} = u_{xx}^{(k+1)}(x),$$

where $u^{(k)}(x) = u(x,t_k),\ u_{xx}^{(k)}(x) = u_{xx}(x,t_k)$. We obtain the second order boundary value problems in x

$$\left(I - dt\frac{\partial^2}{\partial x^2}\right) u^{(k+1)}(x) = u^{(k)}(x),\ \ u^{(k+1)}(-4) = 1,\ u^{(k+1)}(2) = 2.$$

These problems, for each $k = 1,2,...,K$, are also discretized by the spectral Chebyshev method with some N, $dom = [-4,2]$, $kind$ as

$$\left(I_n - dtD^2\right) c^{(k+1)} = c^{(k)},\ \ u^{(k+1)}(-4) = 1,\ u^{(k+1)}(2) = 2.$$

Here $c^{(k)}$ are the Chebyshev spectral coefficients $\left(c_0^{(k)}, c_1^{(k)}, ..., c_{N-1}^{(k)}\right)$ of $u^{(k)}(x)$ corresponding to the grid $x_1, ..., x_N$ in dom. This way we may obtain the solution $u(x_j,t_k)$ on the computational grid $\left(x_j,t_k\right)_{j=1,...,N}^{k=1,...,K}$. The corresponding code is `pde_lin.m` from the folder `Chebpack\level3`.

Similarly, for nonlinear equations of the form $u_t = Lu + Nu$ where L is a linear operator and N a nonlinear one, for example for the Burgers equation

$$u_t = \nu u_{xx} - \left(\frac{u^2}{2}\right)_x,\ x \in [-1,1],\ t > 0,\ \nu = \frac{0.01}{\pi},$$
$$u(x,0) = \sin \pi x,\ u(0,t) = u(1,t) = 0,$$

we may take the backward Euler finite difference for the linear part while the forward Euler finite difference for the nonlinear part.

$$\frac{u^{(k+1)}(x) - u^{(k)}(x)}{dt} = Lu^{(k+1)}(x) + Nu^{(k)}(x),$$

We obtain

$$(I - dt\,L)\,u^{(k+1)}(x) = u^{(k)}(x) + dt\,Nu^{(k)}(x),\ \ u^{(k+1)}(0) = u^{(k+1)}(1) = 0$$

which is implemented in `pde_nonlin.m`.

Of course, we may take the approximating solution in the physical representation on the grid $x_1, ..., x_n$ and the semidiscrete problem becomes $u'(\mathbf{x},t) = \widetilde{D}u(\mathbf{x},t)$ where $\widetilde{D} = TD^2T^{-1}$ is the physical second order derivative. The boundary value condition imposes $u(x_1,t) = \alpha$,

$u(x_n,t)=\beta$ and therefore at these points we don't need to satisfy the equation. Consequently, if $\widehat{D}=\widetilde{D}\,(2:n-1,2:n-1)$ is obtained by eliminating the first and last rows and columns from $\widetilde{D}$, the problem becomes

$$\widehat{u}'(t)=\widehat{D}*\widehat{u}(t)+\widetilde{D}(2:n-1,[1,n])*BC,\ \widehat{u}(0)=\widehat{u}^{(0)},$$

where $BC=(\alpha,\beta)^T$, i.e. $\widehat{u}'(t)=\widehat{D}*\widehat{u}(t)+b$, $\widehat{u}(0)=\widehat{u}^{(0)}$ with the solution

$$\widehat{u}(t)=expm(t\widehat{D})\cdot\widehat{u}^{(0)}+\widehat{D}^{-1}\cdot\left(expm(t\widehat{D})-I_{n-2}\right)\cdot b.$$

Here *expm(A)* is the matricial exponential function of A. The code `pde_lin_matr.m` uses this procedure.
The same thing may be performed in spectral space. The problem

$$u'=L\cdot u,\ u(0)=u_0,\ \ BC=T\cdot u$$

(where L is a linear differential operator with constant coefficients and T is given by `cpv` from `level0`) may be expanded as

$$\begin{pmatrix}\widehat{u}'\\ BC\end{pmatrix}=\begin{pmatrix}\widehat{L}&\widehat{\widehat{L}}\\ \widehat{T}&\widehat{\widehat{T}}\end{pmatrix}\cdot\begin{pmatrix}\widehat{u}\\ \widehat{\widehat{u}}\end{pmatrix}.$$

Therefore, successively,

$$\widehat{u}'=\widehat{L}\widehat{u}+\widehat{\widehat{L}}\widehat{\widehat{u}},\ \ BC=\widehat{T}\widehat{u}+\widehat{\widehat{T}}\widehat{\widehat{u}},\ \widehat{\widehat{u}}=\widehat{\widehat{T}}^{-1}\left(BC-\widehat{T}\widehat{u}\right),\tag{17}$$

$$\widehat{u}'=\left(\widehat{L}-\widehat{\widehat{L}}\widehat{\widehat{T}}^{-1}\widehat{T}\right)\widehat{u}+\widehat{\widehat{L}}\widehat{\widehat{T}}^{-1}BC,\ \widehat{u}'=\widetilde{L}\cdot\widehat{u}+\widetilde{\widetilde{L}}\cdot BC.$$

The exact solution of the last equation is given by

$$\widehat{u}\,(t)=e^{t\widetilde{L}}\cdot\widehat{u}_0+\frac{e^{t\widetilde{L}}-I}{\widetilde{L}}\widetilde{\widetilde{L}}\cdot BC$$

and, using (17), $u=(\widehat{u},\widehat{\widehat{u}})^T$.
This procedure is coded in `pde_lin_ex2.m` from the folder `Chebpack\examples\ex_level3`.
In the case of a nonlinear problem

$$u'=L\cdot u+Nu,\ u(0)=u_0,\ BC=T\cdot u$$

the same procedure leads to

$$\widehat{u}\,(t)=e^{t\widetilde{L}}\cdot\widehat{u}_0+\frac{e^{t\widetilde{L}}-I}{\widetilde{L}}\left(\widetilde{\widetilde{L}}\cdot BC+\widehat{Nu}\right)$$

and to the same $\widehat{\widehat{u}}$ given by (17). This fixed point equation must now be solved using successive iterations method (for t sufficiently small) or using Newton method. It is coded in

`pde_nonlin_ex4.m` in the folder `Chebpack\examples\ex_level3` for the Korteweg-de Vries problem

$$u_t + 6u\,u_x + u_{xxx} = 0,\ x \in [-20, 20]\,,\ t \in [0, 4]$$
$$u(x, 0) = 2sech(x)^2,\ u(-20, t) = u(20, t) = u_x(20, t) = 0.$$

This problem of the form $u_t = Lu + N(u)$ is numerically solved in spectral space $c'(t) = Lc(t) + N(c(t))$ by using the so called implicit exponential time differencing Euler method

$$c^{(k+1)} = e^{Ldt}c^{(k)} + \frac{e^{Ldt} - I}{L}N(c^{(k+1)}),$$

applied in a symmetric form

$$c = e^{Ldt/2}c^{(k)} + \frac{e^{Ldt/2} - I}{L}N(c),\ \ c^{(k+1)} = e^{Ldt/2}c + \frac{e^{Ldt/2} - I}{L}N(c).$$

Here, the first fixed point problem is solved using successive iterations starting with $c^{(k)}$, where $c^{(k)}$ are the Chebyshev coefficients of the numerical solution at the time level k.

5.2 Fractional differential equations

The fractional derivative $D^q f(x)$ with $0 < q < 1, 0 < x \leq b$, in the Riemann-Liouville version, is defined by (Podlubny, 1999)

$$D^q f(x) = \frac{1}{\Gamma(1-q)}\frac{d}{dx}\int_0^x f(t)(x-t)^{-q}dt$$

while the Caputo fractional derivative is

$$D_*^q f(x) = \frac{1}{\Gamma(1-q)}\int_0^x f'(t)(x-t)^{-q}dt$$

and we have

$$D^q f(x) = \frac{f(0)x^{-q}}{\Gamma(1-q)} + D_*^q f(x).$$

Let us consider a function $f : [0, b] \to \mathbb{R}$, with the spectral representation

$$f(x) = \sum_{k=0}^{n-1} c_k T_k(\alpha x + \beta),\ \alpha = \frac{2}{b},\ \beta = -1$$

Using the spectral derivative matrix D, we have

$$f'(x) = \sum_{k=0}^{n-1} (Dc)_k\, T_k(\alpha x + \beta)$$

and using the linearity of the fractional derivative of order $q \in (0, 1)$, we obtain

$$D_*^q f(x) = \frac{1}{\Gamma(1-q)}\int_0^x \frac{f'(t)dt}{(x-t)^q} = \frac{1}{\Gamma(1-q)}\sum_{k=0}^{n-1}(Dc)_k \int_0^x \frac{T_k(\alpha x + \beta)}{(x-t)^q}dt.$$

By calculating the physical values of the above integrals in the columns k of a matrix I, each row corresponding to a sample of x from the Chebyshev grid, the formula for the fractional derivative is

$$D_*^q f(x) = \frac{1}{\Gamma(1-q)} \cdot I \cdot D \cdot T^{-1} f(x)$$

where T is the matrix given by `cpv.m`. This means that the Caputo fractional differentiation matrix $\mathbf{D}$ in physical form is given by

$$\mathbf{D}_{phys} = \frac{1}{\Gamma(1-q)} \cdot I \cdot D \cdot T^{-1}$$

i.e. $\mathbf{D}_{phys}$ times the vector of physical values of f gives the vector of physical values of $D_*^q f$. For the spectral form,

$$\mathbf{D}_{spec} = \frac{1}{\Gamma(1-q)} \cdot T^{-1} \cdot I \cdot D,$$

i.e. $\mathbf{D}_{spec}$ times the vector $T^{-1} f(x)$ of the coefficients of f gives the vector of the coefficients of $D_*^q f$.
If $q > 1$ then let ex be $[q]$ and q will be replaced by $q - [q]$. In this case, the differentiation matrix will be

$$\mathbf{D}_{spec} = \frac{1}{\Gamma(1-q)} \cdot T^{-1} \cdot I \cdot D^{(ex+1)}.$$

In order to avoid the singularity of the fractional derivative $D^q f$ if $f(0) \neq 0$, **we suppose that** $\mathbf{f}(\mathbf{0}) = \mathbf{0}$. The problem of computing the integrals

$$I_k(x) = \int_0^x \frac{T_k(\alpha x - 1)}{(x-t)^q} dt$$

for each x of the grid may be solved iteratively, see (Piessens & Branders, 1976). Indeed, we have, by direct calculation, using the recurrence formula for Chebyshev polynomials, as well as for the derivatives of the Chebyshev polynomials, for $k = 3, 4, ..$

$$I_k(x) \cdot \left(1 + \frac{1-q}{k}\right) = 2(\alpha x - 1) \cdot I_{k-1}(x) + \left(\frac{1-q}{k-2} - 1\right) \cdot I_{k-2}(x) - \frac{2(-1)^k}{k(k-2)} x^{1-q}$$

and

$$I_0(x) = \frac{x^{1-q}}{1-q}/2,\ I_1(x) = \frac{\alpha x^{2-q}}{(2-q)(1-q)} - \frac{x^{1-q}}{1-q},$$
$$I_2(x) = \frac{4\alpha^2 x^{3-q}}{(3-q)(2-q)(1-q)} - \frac{4\alpha x^{2-q}}{(2-q)(1-q)} + \frac{x^{1-q}}{1-q}.$$

This recurrence is as stable as the recurrence which calculates the Chebyshev polynomials. The calculation of the fractional derivative matrix $\mathbf{D}_{spec}$ is coded in `deriv_frac.m` from the folder `Chebpack\level3`. Next, if needed, $\mathbf{D}_{phys} = \frac{1}{\Gamma(1-q)} \cdot I \cdot D^{(ex+1)} \cdot T^{-1} = T \cdot \mathbf{D}_{spec} \cdot T^{-1}$.
Using an idea of Sugiura & Hasegawa (Sugiura & Hasegawa, 2009), let $J(s; f)$ be

$$J(s; f) = \int_0^s f'(t)(s-t)^{-q} dt$$

and, approximating $f(t)$ by a combination $p_n(t)$ of Chebyshev polynomials on $(0,1)$ we have the approximations $|J(s;f) - J(s;p_n)| \sim O(n\rho^{-n})$, for some $\rho > 1$.
Of course, this method is not suitable for the functions with a singularity in $[0,1]$ or singularities of lower-order derivatives, like $x^{0.1}$ for example. In this case, n must be excessively large.
For the initial value problems for fractional differential equations, let us consider the problem

$$D_*^q y(x) = x^2 + \frac{2}{\Gamma(3-q)} x^{2-q} - y(x),\ y(0) = 0,\ q = 0.5,\ x \in [0,1]\,.$$

The physical discretization is

$$A\mathbf{y} = \mathbf{b},\ A = T \cdot DF \cdot T^{-1} + I_N,\ \mathbf{b} = x^2 + 2/\Gamma(3-q) \cdot x^{2-q}$$

where, in order to implement the initial condition, the first row of A is replaced by $[1,0,...,0]$ and $\mathbf{b}(1)$ is replaced by 0. The solution is now $y(x) = A^{-1}B$. The example is coded in `deriv_frac_ex1.m` from the folder `Chebpack\examples\ex_level3`.
If we use the spectral representation, for example for the problem

$$D^q y(x) + y(x) = 1,\ y(0) = 0,\ y\prime(0) = 0,\ q = 1.8,\ x \in [0,5],$$

with the exact solution $yex = x^q E_{q,q+1}(-x^q)$, the spectral discretized form becomes

$$Ay \equiv (D^q + I_n)\, y = T^{-1} \cdot \mathbf{1} = B,$$
$$A(n-1,:) = T(1,:),\ B(n-1) = 0,\ A(n,:) = T(1,:) * D,\ B(n) = 0,$$

see `deriv_frac_ex2.m`
For nonhomogeneous initial conditions like $y(0) = c_0$, $y'(0) = c_1$, we perform a function change $y(x) = c_0 + c_1 x + z(x)$ where z verifies the same equation but with homogeneous initial conditins ($c_0 + c_1 x$ disappears by differentiation), see `deriv_frac_ex3.m`. Examples for discontinuous data, boundary value problems or eigenproblems are also given.

6. Conclusion

The new package *Chebpack* (Trif, 2011) is a large and powerful extension of *LiScEig* (Trif, 2005). It is based on the Chebyshev tau spectral method and it is applied to linear and nonlinear differential or integral problems, including eigenvalue problems. An experimental module contains applications to linear or nonlinear partial differential problems and to fractional differential problems. Each module is illustrated by many examples. A future version will handle also piecewise functions as well as functions with singularities.
The following comparisons with MATLAB codes *bvp4c, bvp5c* as well as with *DMS* or *Chebfun* prove the efficiency of *Chebpack*. The elapsed time was evaluated after three code executions.
The first test problem (`test1.m` in the folder `Chebpack\tutorial`) is

$$-u'' - \frac{1}{6x}u' + \frac{1}{x^2}u = \frac{19}{6}x^{1/2},\ u(0) = u(1) = 0,\ x \in [0,1]$$

with the exact solution $u_{ex}(x) = x^{3/2} - x^{5/2}$. The elapsed time and the errors are presented in Table 1. Here *Chebpack* uses the differentiation matrix.
The second test problem (`test2.m` in the folder `Chebpack\tutorial`) is (13) with $\varepsilon = 10^{-4}$. The elapsed time and the errors are presented in Table 2. Here *Chebpack* uses the integration matrix.

	bvp4c RelTol=1.e-5	bvp5c RelTol=1.e-4	dms N=64	Chebfun N=98	Chebpack N=80
elapsed time (sec)	0.737	0.744	0.004	0.192	0.005
errors	3.2e-7	9.2e-8	2.3e-7	1.8e-7	4.6e-7

Table 1. Test 1

	bvp4c default	bvp5c default	dms N=1024	Chebfun N=[13,114,109,14]	Chebpack N=1024
elapsed time (sec)	0.131	0.244	3.471	1.053	0.041
errors	8.6e-5	6.6e-6	8.4e-13	1.0e-12	1.e-14

Table 2. Test 2

7. References

Berrut, J. P. & Trefethen, L. N. (2004). Barycentric Lagrange interpolation, *SIAM Review* Vol.46(No.3): 501–517.

Boyd, J. P. (2000). *Chebyshev and Fourier Spectral Methods*, Dover Publications, Inc.

Canuto, C., Hussaini, M. Y., Quarteroni, A. & Zang, T. A. (1988). *Spectral Methods in Fluid Dynamics*, Springer-Verlag, Berlin.

Driscoll, T. A. (2010). Automatic spectral collocation for integral, integro-differential, and integrally reformulated differential equations, *J. Comp. Phys.* Vol.229: 5980–5998.

Mason, J. & Handscomb, D. (2003). *Chebyshev Polynomials*, Chapman & Hall/CRC.

Ortiz, E. L. & Samara, H. (1981). An operational approach to the tau method for the numerical solution of non-linear differential equations, *Computing* Vol.27: 15–25.

Piessens, R. & Branders, M. (1976). Numerical solution of integral equations of mathematical physics, using Chebyshev polynomials, *J. Comp. Phys. (1976)* Vol.21: 178–196.

Podlubny, I. (1999). *Fractional Differential Equations*, Academic Press.

Sugiura, H. & Hasegawa, T. (2009). Quadrature rule for Abel's equations: Uniformly approximating fractional derivatives, *J. Comp. Appl. Math.* Vol.223: 459–468.

Trefethen, L. N. (2000). *Spectral Methods in MATLAB*, SIAM, Philadelphia.

Trefethen, L. N. (2008). Is Gauss quadrature better than Clenshaw–Curtis?, *SIAM Review* Vol.50(No.1): 67–87.

Trefethen, L. N. et al. (2011). *Chebfun Version 4.0*, The Chebfun Development Team. URL: *http://www.maths.ox.ac.uk/chebfun/*

Trif, D. (2005). LiScEig, MATLAB Central. URL: *http://www.mathworks.com/matlabcentral/fileexchange/8689-lisceig*

Trif, D. (2011). Chebpack, MATLAB Central. URL: *http://www.mathworks.com/matlabcentral/fileexchange/32227-chebpack*

Waldvogel, J. (2006). Fast construction of the Fejér and Clenshaw–Curtis quadrature rules, *BIT Numerical Mathematics* Vol.46: 195–202.

Weideman, J. A. C. & Reddy, S. C. (2000). A MATLAB differentiation matrix suite. URL: *http://www.mathworks.com/matlabcentral/fileexchange/29-dmsuite*

MATLAB - A Ubiquitous Tool for the Practical Engineer

Edited by Prof. Clara Ionescu

ISBN 978-953-307-907-3
Hard cover, 564 pages
Publisher InTech
Published online 13, October, 2011
Published in print edition October, 2011

A well-known statement says that the PID controller is the â€œbread and butterâ€ of the control engineer. This is indeed true, from a scientific standpoint. However, nowadays, in the era of computer science, when the paper and pencil have been replaced by the keyboard and the display of computers, one may equally say that MATLAB is the â€œbreadâ€ in the above statement. MATLAB has became a de facto tool for the modern system engineer. This book is written for both engineering students, as well as for practicing engineers. The wide range of applications in which MATLAB is the working framework, shows that it is a powerful, comprehensive and easy-to-use environment for performing technical computations. The book includes various excellent applications in which MATLAB is employed: from pure algebraic computations to data acquisition in real-life experiments, from control strategies to image processing algorithms, from graphical user interface design for educational purposes to Simulink embedded systems.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Damian Trif (2011). Matrix Based Operatorial Approach to Differential and Integral Problems, MATLAB - A Ubiquitous Tool for the Practical Engineer, Prof. Clara Ionescu (Ed.), ISBN: 978-953-307-907-3, InTech, Available from: http://www.intechopen.com/books/matlab-a-ubiquitous-tool-for-the-practical-engineer/matrix-based-operatorial-approach-to-differential-and-integral-problems

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2011 The Author(s). Licensee IntechOpen. This is an open access article distributed under the terms of the Creative Commons Attribution 3.0 License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

IntechOpen

IntechOpen