Introduction to the Chebfun System

Titus Pinta

Presentation at the ICTP Institute Seminar, January 30, 2019.

Contents I

1. Introduction
2. Approximating Functions
3. Chebyshev Series
4. Chebyfun and Approximations
5. Clenshaw-Curtis Algorithm
6. Chebfun and Integration
7. Chebfun and Primitives
8. Differentiation
9. Chebops
10. Chebfun and ODEs
11. Solving IVPs
12. Solving BVPs
13. References

Definitions

Given x and y, two vectors from $\mathbb{R}^n$ with $x_i \neq x_j$, $i \neq j$, there exists an unique polynomial p of degree at most $n-1$ with the property

$$p(x_i) = y_i$$

Definitions

Given x and y, two vectors from $\mathbb{R}^n$ with $x_i \neq x_j$, $i \neq j$, there exists an unique polynomial p of degree at most $n-1$ with the property

$$p(x_i) = y_i$$

This unique polynomial is called the Lagrange Interpolation polynomial.

Definitions

Given x and y, two vectors from $\mathbb{R}^n$ with $x_i \neq x_j$, $i \neq j$, there exists an unique polynomial p of degree at most $n-1$ with the property

$$p(x_i) = y_i$$

This unique polynomial is called the Lagrange Interpolation polynomial. If the vector x given by

$$x_i = \cos\left(\frac{i}{n}\pi\right), \quad x_i \in [-1, 1]$$

the nodes of interpolation are called Chebyshev-Gauss-Lobatto nodes.

Definitions

Given x and y, two vectors from $\mathbb{R}^n$ with $x_i \neq x_j$, $i \neq j$, there exists an unique polynomial p of degree at most $n-1$ with the property

$$p(x_i) = y_i$$

This unique polynomial is called the Lagrange Interpolation polynomial. If the vector x given by

$$x_i = \cos\left(\frac{i}{n}\pi\right), \quad x_i \in [-1, 1]$$

the nodes of interpolation are called Chebyshev-Gauss-Lobatto nodes. These points are the extremum points of the Chebyshev polynomials, defined as

$$\begin{cases} T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \\ T_0(x) = 1, \quad T_1(x) = x. \end{cases}$$

Approximating Functions

Given a function $f\colon [-1,1] \to R$, we can evaluate f at n Chebyshev-Gauss-Lobatto points and construct the Lagrange polynomial that interpolates the given points.

$$p_n(x_i) = f(x_i), \quad \forall x_i = \cos\left(\frac{i}{n}\pi\right).$$

Approximating Functions

Given a function $f\colon [-1,1] \to R$, we can evaluate f at n Chebyshev-Gauss-Lobatto points and construct the Lagrange polynomial that interpolates the given points.

$$p_n(x_i) = f(x_i), \quad \forall x_i = \cos\left(\tfrac{i}{n}\pi\right).$$

Using the barycentric form of the Lagrange polynomial, we can derive a fast algorithm for evaluating the resulting polynomial

$$P_n(x) = \frac{\frac{f(-1)}{2(x+1)} + \frac{f(1)(-1)^n}{2(x-1)} \sum_{k=1}^{n-1} \frac{f(x_k)(-1)^k}{x - x_k}}{\frac{1}{2(x+1)} + \frac{(-1)^n}{2(x-1)} + \sum_{k=1}^{n-1} \frac{(-1)^k}{x - x_k}}$$

Approximating Functions

Given a function $f\colon [-1,1] \to R$, we can evaluate f at n Chebyshev-Gauss-Lobatto points and construct the Lagrange polynomial that interpolates the given points.

$$p_n(x_i) = f(x_i), \quad \forall x_i = \cos\left(\frac{i}{n}\pi\right).$$

Using the barycentric form of the Lagrange polynomial, we can derive a fast algorithm for evaluating the resulting polynomial

$$P_n(x) = \frac{\frac{f(-1)}{2(x+1)} + \frac{f(1)(-1)^n}{2(x-1)} \sum_{k=1}^{n-1} \frac{f(x_k)(-1)^k}{x - x_k}}{\frac{1}{2(x+1)} + \frac{(-1)^n}{2(x-1)} + \sum_{k=1}^{n-1} \frac{(-1)^k}{x - x_k}}$$

Chebyshev Series

We can approximate a function f defined on the interval $[-1, 1]$ with a series of Chebyshev polynomials

$$f(x) \approx \sum_{i=0}^{n} a_i T_i(x).$$

Chebyshev Series

We can approximate a function f defined on the interval $[-1, 1]$ with a series of Chebyshev polynomials

$$f(x) \approx \sum_{i=0}^{n} a_i T_i(x).$$

We can define the function $F(\theta) = f(\cos(\theta))$ where $\theta \in [0, \pi]$ and using the fact that

$$T_n(x) = \cos(n \arccos(x)),$$

we can approximate

$$F(\theta) \approx \sum_{i=1}^{n} a_i \cos(n\theta).$$

Chebyshev Series

We can approximate a function f defined on the interval $[-1, 1]$ with a series of Chebyshev polynomials

$$f(x) \approx \sum_{i=0}^{n} a_i T_i(x).$$

We can define the function $F(\theta) = f(\cos(\theta))$ where $\theta \in [0, \pi]$ and using the fact that

$$T_n(x) = \cos(n \arccos(x)),$$

we can approximate

$$F(\theta) \approx \sum_{i=1}^{n} a_i \cos(n\theta).$$

This suggests using a FFT to compute the coefficients of the Chebyshev expansion.

Chebfun and Approximations

Chebfun is a Matlab package with the primary goal to make computing with functions as similar as possible with computing with vectors, in other words it aims to "Feel symbolic, but work numeric".

Chebfun and Approximations

Chebfun is a Matlab package with the primary goal to make computing with functions as similar as possible with computing with vectors, in other words it aims to "Feel symbolic, but work numeric".
It expands a given function f as a Chebyshev series, using a FFT based algorithm, and then performs its calculations on vectors of coefficients.

```
chebfun(@(x)sin(x));
```

Chebfun System

Chebfuns are polynomial approximations of functions. They can be constructed from a string representing the expression of a function or from an anonymous function.

Chebfun System

Chebfuns are polynomial approximations of functions. They can be constructed from a string representing the expression of a function or from an anonymous function. Normally, chebfuns are represented on $[-1, 1]$, but the range can be specified in the constructor and the Chebyshev polynomials employed will be rescaled.

Chebfun System

Chebfuns are polynomial approximations of functions. They can be constructed from a string representing the expression of a function or from an anonymous function. Normally, chebfuns are represented on $[-1, 1]$, but the range can be specified in the constructor and the Chebyshev polynomials employed will be rescaled.

Chebfun System

Chebfuns are polynomial approximations of functions. They can be constructed from a string representing the expression of a function or from an anonymous function. Normally, chebfuns are represented on $[-1, 1]$, but the range can be specified in the constructor and the Chebyshev polynomials employed will be rescaled.
The usual mathematical operators are overloaded in Chebfun, in order to allow operations with chebfuns, combining existing chebfuns to create a new chebfun.

```
x = chebfun('x')
x.^2 + 1
exp(x).*sin(x)
```

Chebyshev-Gauss-Lobatto Nodes vs. Equispaced Nodes

Using Chebyshev-Gauss-Lobatto nodes gives a great advantage over equispaced nodes. On an uniform grid the interpolant may oscillate at the ends of the interval, no matter how smooth the function is. To put it another way, in equispaced nodes $p_n(x)$ may diverge even if f is analytic.

Chebyshev-Gauss-Lobatto Nodes vs. Equispaced Nodes

Using Chebyshev-Gauss-Lobatto nodes gives a great advantage over equispaced nodes. On an uniform grid the interpolant may oscillate at the ends of the interval, no matter how smooth the function is. To put it another way, in equispaced nodes $p_n(x)$ may diverge even if f is analytic.

A classic example of this phenomenon is Runge's function

$$f(x) = \frac{1}{1 + 25x^2},$$

for which the interpolation polynomial on an uniform grid on $[-1, 1]$ diverges.

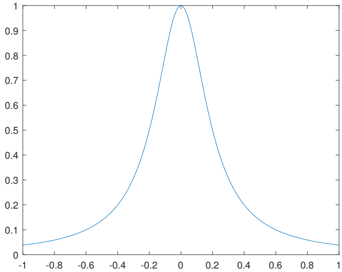
Chebyshev-Gauss-Lobatto Nodes vs. Equispaced Nodes

In contrast the Chebfun System has no problem approximating Runge's function up to machine precision using 189 terms of a Chebyshev expansion.

Chebyshev-Gauss-Lobatto Nodes vs. Equispaced Nodes

In contrast the Chebfun System has no problem approximating Runge's function up to machine precision using 189 terms of a Chebyshev expansion.

```
f = chebfun(@(x)(1/(1 + 25 * x^2)));
length(f)
plot(f)
```



It is proven that for a Lipschitz continuous function f the Chebyshev series of the function

$$\sum_{k=0}^{\infty} a_k T_k(x), \quad a_k = \int_{-1}^{1} \frac{f(x)\, T_k(x)}{\sqrt{1-x^2}}\, dx,$$

converges.

Edge Detection

It is proven that for a Lipschitz continuous function f the Chebyshev series of the function

$$\sum_{k=0}^{\infty} a_k T_k(x), \quad a_k = \int_{-1}^{1} \frac{f(x)\, T_k(x)}{\sqrt{1-x^2}}\, dx,$$

converges.

Unfortunately, in practice, not all functions are smooth enough. Real functions may satisfy some smoothness conditions only piece-wise. This kind of situation was taken into consideration, as part of the design of the Chebfun System, and it can represent functions as piece-wise polynomials.

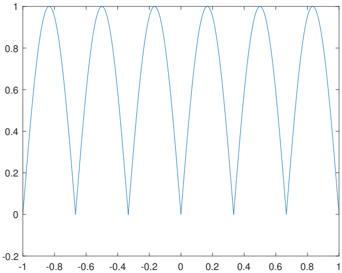
Edge Detection

The break points of a picewise continuous function are detected via the algorithm presented in [PPT10], when the option 'splitting' is on in a chebfun constructor.

Edge Detection

The break points of a picewise continuous function are detected via the algorithm presented in [PPT10], when the option 'splitting' is on in a chebfun constructor. When a chebfun is constructed from a combination of other chebfuns, using commands like abs or max breakpoints are also introduced automatically.

```
f = chebfun(@(x)abs(sin(3 * pi * x)), 'splitting', 'on')
plot(f)
```



Clenshaw-Curtis Algorithm

By approximating the function f with a Chebyshev expansion, we can construct a cosine approximation for a function F, given by $F(\theta) = f(\cos\theta)$.

$$f(\cos\theta) = F(\theta) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(k\theta).$$

Clenshaw-Curtis Algorithm

By approximating the function f with a Chebyshev expansion, we can construct a cosine approximation for a function F, given by $F(\theta) = f(\cos\theta)$.

$$f(\cos\theta) = F(\theta) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(k\theta).$$

We can make a change of variable in order to use the cosine series

$$\int_{-1}^{1} f(x)dx = \int_{0}^{\pi} f(\cos(\theta)) \sin\theta d\theta.$$

Clenshaw-Curtis Algorithm

To compute the definite integral of f we can integrate the cosine series term by term

$$\int_0^\pi f(\cos\theta)\sin\theta d\theta = a_0 + \sum_{k=1}^{\infty} \frac{2a_{2k}}{1-(2k)^2},$$

where a_{2k} are Fourier coefficients defined by

$$a_{2k} = \frac{2}{\pi}\int_0^\pi f(cos\theta)\sin k\theta d\theta$$

Clenshaw-Curtis Algorithm

In our descrete setting we can compute a_{2k} using an algorithm based on the Discrete Cosine Transform

$$a_{2k} \approx \frac{2}{n}\left(\frac{f(1)}{2} + (-1)^k \frac{f(-1)}{2} + \sum_{k=1}^{n-1} f\left(\cos\frac{i\pi}{n}\right)\cos ki\pi n\right)$$

Clenshaw-Curtis Algorithm

In our descrete setting we can compute a_{2k} using an algorithm based on the Discrete Cosine Transform

$$a_{2k} \approx \frac{2}{n}\left(\frac{f(1)}{2} + (-1)^k \frac{f(-1)}{2} + \sum_{k=1}^{n-1} f\left(\cos\frac{i\pi}{n}\right)\cos ki\pi n\right)$$

Because we are interested in a discrete representation of a function we can take advantage of the aliasing property of Chebyshev polynomials and arrive at the formula

$$\int_{-1}^{1} f(x)dx = a_0 + \left(\sum_{k=1}^{n-1} \frac{2a_{2k}}{1-(2k)^2}\right) + \frac{a_{2n}}{1-(n)^2}$$

Chebfun and Integration

An algorithm similar to the one described above is implemented in the Chebfun System.

Chebfun and Integration

An algorithm similar to the one described above is implemented in the Chebfun System.
Using the FFT and the DCT the coefficients a_{2k} are computed.

Chebfun and Integration

An algorithm similar to the one described above is implemented in the Chebfun System.
Using the FFT and the DCT the coefficients a_{2k} are computed.
The command sum which originally returned the sum of the elements of a vector, was overloaded, and now takes also a chebfun as an argument and returns its integral on the domain of definition.

Chebfun and Integration

An algorithm similar to the one described above is implemented in the Chebfun System.
Using the FFT and the DCT the coefficients a_{2k} are computed.
The command sum which originally returned the sum of the elements of a vector, was overloaded, and now takes also a chebfun as an argument and returns its integral on the domain of definition.

```
f = chebfun(@(x)sin(x), [0, pi]);
sum(f)
```

Chebfun and Primitives

Chebfun can also handle indefinite integrals, evaluated as

$$\int_{-1}^{x} f(t)dt.$$

Chebfun and Primitives

Chebfun can also handle indefinite integrals, evaluated as

$$\int_{-1}^{x} f(t)dt.$$

If f is a chebfun, than its indefinite integral will be another chebfun, F, with $F' = f$ and $F(-1) = 0$. When we work with definite integrals we should never forget the arbitrary constant of integration.

Chebfun and Primitives

Chebfun can also handle indefinite integrals, evaluated as

$$\int_{-1}^{x} f(t)dt.$$

If f is a chebfun, than its indefinite integral will be another chebfun, F, with $F' = f$ and $F(-1) = 0$. When we work with definite integrals we should never forget the arbitrary constant of integration.
The command cumsum normally had a vector as its input and would return another vector which on each position k had the sum of the first k elements of the original vector. In Chebfun it was overloaded to give the indefinite integral.

Chebfun and Primitives

Chebfun can also handle indefinite integrals, evaluated as

$$\int_{-1}^{x} f(t)dt.$$

If f is a chebfun, than its indefinite integral will be another chebfun, F, with $F' = f$ and $F(-1) = 0$. When we work with definite integrals we should never forget the arbitrary constant of integration.
The command cumsum normally had a vector as its input and would return another vector which on each position k had the sum of the first k elements of the original vector. In Chebfun it was overloaded to give the indefinite integral.

```
f = chebfun(@(x)sin(x), [0, pi]);
cumsum(f)
```

Differentiation

The idea of a matrix acting on a vector is extended naturally to Chebfun, with the place of matrix occupied by linear operators, called chebops.

Differentiation

The idea of a matrix acting on a vector is extended naturally to Chebfun, with the place of matrix occupied by linear operators, called chebops.

One of the most important class of linear operators is the class of differential operators. Expanding the function f as a series of Chebyshev polynomials we can differentiate term by term arriving at a spectral method of differentiation.

Differentiation

The idea of a matrix acting on a vector is extended naturally to Chebfun, with the place of matrix occupied by linear operators, called chebops.
One of the most important class of linear operators is the class of differential operators. Expanding the function f as a series of Chebyshev polynomials we can differentiate term by term arriving at a spectral method of differentiation.
Chebfun overloads the standard Matlab command diff, which normally takes in a vector and returns a vector of finite differences. diff can now take as argument a chebfun and it will return its derivative.

Differentiation

The idea of a matrix acting on a vector is extended naturally to Chebfun, with the place of matrix occupied by linear operators, called chebops.

One of the most important class of linear operators is the class of differential operators. Expanding the function f as a series of Chebyshev polynomials we can differentiate term by term arriving at a spectral method of differentiation.

Chebfun overloads the standard Matlab command diff, which normally takes in a vector and returns a vector of finite differences. diff can now take as argument a chebfun and it will return its derivative.

```
f = chebfun(@(x)sin(x), [0, pi]);
cumsum(f)
```

Chebops

We can define a chebop using the constructor.

```
D = chebop(@(u) diff(u))
```

Chebops

We can define a chebop using the constructor.

```
D = chebop(@(u) diff(u))
```

Chebfun sees chebops as direct analogues to matrices. They act as linear operators on chebfuns, condition by sharing the same domain. Similar to the constructor for chebfuns we can change the domain of definition for a chebop.

```
I = chebop(@(u)cumsum(u), [0, 1])
```

Chebfun and ODEs

Chebops can be combined to create differential and integral equations of the form

$$Lu = f(u).$$

Chebfun and ODEs

Chebops can be combined to create differential and integral equations of the form

$$Lu = f(u).$$

Because both the operator and the function on which it operates are represented discreetly, solving a linear operator equation reduces to solving a system of linear equations. Chebfun can employ different techniques to solve equation ranging from direct methods like LU factorization to iterative methods like Krylov Subspace Methods and Conjugate Gradient.

Solving IVPs

To uniquely determine the solution of an Initial Value Problem we need to supply the value of the function, and eventually its derivatives, at a point x_0.

Solving IVPs

To uniquely determine the solution of an Initial Value Problem we need to supply the value of the function, and eventually its derivatives, at a point x_0.
In Chebfun the initial condition is stored as a field called lbc (left boundary condition) in the class chebop. The operator $\backslash$, that in Matlab solves a system of linear equations, was overloaded to solve a linear operator equation.

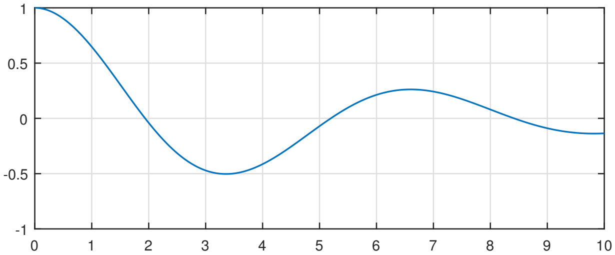
Solving IVPs

To uniquely determine the solution of an Initial Value Problem we need to supply the value of the function, and eventually its derivatives, at a point x_0.
In Chebfun the initial condition is stored as a field called lbc (left boundary condition) in the class chebop. The operator $\backslash$, that in Matlab solves a system of linear equations, was overloaded to solve a linear operator equation.
Since 2014 Chebfun has changed its approach to solve IVPs using the differentiation of Chebyshev polynomials, and now uses the classic methods from Matlab like ode45. All this functions were overloaded to accept chebfuns as arguments.

The damped harmonic oscillator:

```
D = chebop(@(u) diff(u, 2)+0.4*diff(u)+sin(u), [0, 10]);
D.lbc = [1, 0];
sol = D\chebfun('0', [0, 10]);
```

We get the following solution:



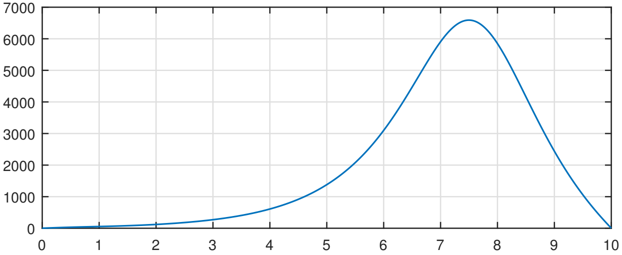
Solving BVPs

Chebfun can also handle Boundary Value Problems. The algorithm is also based on constructing a differential matrix and solving a system of linear equations.

Solving BVPs

Chebfun can also handle Boundary Value Problems. The algorithm is also based on constructing a differential matrix and solving a system of linear equations.

```
D = chebop(@(u) diff(u, 2)+0.4*diff(u)+sin(u), [0, 10]);
D.lbc = 1;
D.rbc = 0.3;
sol = D\chebfun('0', [0, 10]);
```



References

L. N. Trefethen, *Chebfun Guide*. 2009. URL: https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/23972/versions/22/previews/chebfun/guide/html/guide1.html.

R. Pachon, R. B. Platte, and L. N. Trefethen, Piecewise-smooth chebfuns. *IMA Journal of Numerical Analysis* 30 (4 2010).

L. N. Trefethen, *Approximation Theory and Approximation Practice*. SIAM, 2013.