1 Lecture 20: Orthogonal polynomials and differential equations

This lecture we do the following:

1. Recurrence relationships for Chebyshev and ultraspherical polynomials
   - Conversion
   - Three-term recurrence and Jacobi operators
2. Application: solving differential equations
   - First order constant coefficients differential equations
   - Second order constant coefficient differential equations with boundary conditions
   - Non-constant coefficients
3. Differential equations satisfied by orthogonal polynomials

That is, we introduce recurrences related to ultraspherical polynomials. This allows us to represent general linear differential equations as almost-banded systems.

1.1 Recurrence relationships for Chebyshev and ultraspherical polynomials

We have discussed general properties, but now we want to discuss some classical orthogonal polynomials, beginning with Chebyshev (first kind) $T_n(x)$, which is orthogonal w.r.t. $\frac{1}{\sqrt{1-x^2}}$ and ultraspherical $C_n^{(\lambda)}(x)$, which is orthogonal w.r.t. $(1-x^2)^{\lambda-\frac{1}{2}}$ for $\lambda > 0$. Note that Chebyshev (second kind) satisfies $U_n(x) = C_n^{(1)}(x)$.

For Chebyshev, recall we have the normalization constant (here we use a superscript $T_n(x) = k_n^{\mathrm{T}} x^n + O(x^{n-1})$)

$$k_0^{\mathrm{T}} = 1, k_n^{\mathrm{T}} = 2^{n-1}$$

For Ultraspherical $C_n^{(\lambda)}$, this is

$$k_n^{(\lambda)} = \frac{2^n (\lambda)_n}{n!} = \frac{2^n \lambda(\lambda+1)(\lambda+2)\cdots(\lambda+n-1)}{n!}$$

where $(\lambda)_n$ is the Pochhammer symbol. Note for $U_n(x) = C_n^{(1)}(x)$ this simplifies to $k_n^{\mathrm{U}} = k_n^{(1)} = 2^n$.

We have already found the recurrence for Chebyshev:

$$xT_n(x) = \frac{T_{n-1}(x)}{2} + \frac{T_{n+1}(x)}{2}$$

We will show that we can use this to find the recurrence for *all* ultraspherical polynomials. But first we need some special recurrences.

Remark Jacobi, Laguerre, and Hermite all have similar relationships, which will be discussed further in the problem sheet.

1.1.1 Derivatives

It turns out that the derivative of $T_n(x)$ is precisely a multiple of $U_{n-1}(x)$, and similarly the derivative of $C_n^{(\lambda)}$ is a multiple of $C_{n-1}^{(\lambda+1)}$.

Proposition (Chebyshev derivative)

$$T_n'(x) = nU_{n-1}(x)$$

Proof We first show that $T_n'(x)$ is orthogonal w.r.t. $\sqrt{1-x^2}$ to all polynomials of degree $m < n-1$, denoted f_m, using integration by parts:

$$\begin{aligned}
\langle T_n', f_m \rangle_{\mathrm{U}} &= \int_{-1}^{1} T_n'(x) f_m(x) \sqrt{1-x^2} \mathrm{d}x \\
&= -\int_{-1}^{1} T_n(x)(f_m'(x)(1-x^2) - x f_m) \frac{1}{\sqrt{1-x^2}} \mathrm{d}x \\
&= -\left\langle T_n, f_m'(1-x^2) - x f_m \right\rangle_{\mathrm{T}} = 0
\end{aligned}$$

since $f_m'(1-x^2) - x f_m$ is degree $m - 1 + 2 = m + 1 < n$.

The constant works out since

$$T_n'(x) = \frac{\mathrm{d}}{\mathrm{d}x}(2^{n-1}x^n) + O(x^{n-2}) = n2^{n-1}x^{n-1} + O(x^{n-2})$$

■

The exact same proof shows the following:

Proposition (Ultraspherical derivative) $\frac{\mathrm{d}}{\mathrm{d}x} C_n^{(\lambda)}(x) = 2\lambda C_{n-1}^{(\lambda+1)}(x)$

Like the three-term recurrence and Jacobi operators, it is useful to express this in matrix form. That is, for the derivatives of $T_n(x)$ we get

$$\frac{\mathrm{d}}{\mathrm{d}x} \begin{pmatrix} T_0(x) \\ T_1(x) \\ T_2(x) \\ \vdots \end{pmatrix} = \begin{pmatrix} 0 & & & \\ 1 & & & \\ & 2 & & \\ & & 3 & \\ & & & \ddots \end{pmatrix} \begin{pmatrix} U_0(x) \\ U_1(x) \\ U_2(x) \\ \vdots \end{pmatrix}$$

which lets us know that, for

$$f(x) = (T_0(x), T_1(x), \ldots) \begin{pmatrix} f_0 \\ f_1 \\ \vdots \end{pmatrix}$$

we have a derivative operator in coefficient space as

$$f'(x) = (U_0(x), U_1(x), \ldots) \begin{pmatrix} 0 & 1 & & & \\ & & 2 & & \\ & & & 3 & \\ & & & & \ddots \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \end{pmatrix}$$

Demonstration Here we see that applying a matrix to a vector of coefficients successfully calculates the derivative:

```
using ApproxFun, Plots, LinearAlgebra
f = Fun(x -> cos(x^2), Chebyshev())   # f is expanded in Chebyshev coefficients
n = ncoefficients(f)   # This is the number of coefficients
D = zeros(n-1,n)
for k=1:n-1
    D[k,k+1] = k
end
D
```

```
31×32 Array{Float64,2}:
 0.0  1.0  0.0  0.0  0.0  0.0  0.0  …   0.0   0.0   0.0   0.0   0.0   0.0
 0.0  0.0  2.0  0.0  0.0  0.0  0.0      0.0   0.0   0.0   0.0   0.0   0.0
 0.0  0.0  0.0  3.0  0.0  0.0  0.0      0.0   0.0   0.0   0.0   0.0   0.0
 0.0  0.0  0.0  0.0  4.0  0.0  0.0      0.0   0.0   0.0   0.0   0.0   0.0
 0.0  0.0  0.0  0.0  0.0  5.0  0.0      0.0   0.0   0.0   0.0   0.0   0.0
 0.0  0.0  0.0  0.0  0.0  0.0  6.0  …   0.0   0.0   0.0   0.0   0.0   0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0   0.0   0.0   0.0   0.0   0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0   0.0   0.0   0.0   0.0   0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0   0.0   0.0   0.0   0.0   0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0   0.0   0.0   0.0   0.0   0.0
 ⋮                        ⋮         ⋱                          ⋮
 0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0   0.0   0.0   0.0   0.0   0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0   0.0   0.0   0.0   0.0   0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0   0.0   0.0   0.0   0.0   0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  26.0   0.0   0.0   0.0   0.0   0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0  27.0   0.0   0.0   0.0   0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0   0.0  28.0   0.0   0.0   0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0   0.0   0.0  29.0   0.0   0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0      0.0   0.0   0.0   0.0  30.0   0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0   0.0   0.0   0.0   0.0  31.0
```

Here `D*f.coefficients` gives the vector of coefficients corresponding to the derivative, but now the coefficients are in the $U_n(x)$ basis, that is, `Ultraspherical(1)`:

```
fp = Fun(Ultraspherical(1), D*f.coefficients)
fp(0.1)
```

```
-0.001999966666833569
```

Indeed, it matches the "true" derivative:

```
f'(0.1),-2*0.1*sin(0.1^2)
```

```
(-0.0019999666668335634, -0.0019999666668333335)
```

Note that in ApproxFun.jl we can construct these operators rather nicely:

```
D = Derivative()
(D*f)(0.1)
```

```
-0.001999966666833569
```

Here we see that we can produce the ∞-dimensional version as follows:

```
D : Chebyshev() → Ultraspherical(1)
```

```
ConcreteDerivative : Chebyshev() → Ultraspherical(1)
 ⋅  1.0  ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅   ⋅
 ⋅   ⋅  2.0   ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅   ⋅
 ⋅   ⋅   ⋅   3.0   ⋅    ⋅    ⋅    ⋅    ⋅    ⋅   ⋅
 ⋅   ⋅   ⋅    ⋅   4.0   ⋅    ⋅    ⋅    ⋅    ⋅   ⋅
 ⋅   ⋅   ⋅    ⋅    ⋅   5.0   ⋅    ⋅    ⋅    ⋅   ⋅
 ⋅   ⋅   ⋅    ⋅    ⋅    ⋅   6.0   ⋅    ⋅    ⋅   ⋅
 ⋅   ⋅   ⋅    ⋅    ⋅    ⋅    ⋅   7.0   ⋅    ⋅   ⋅
 ⋅   ⋅   ⋅    ⋅    ⋅    ⋅    ⋅    ⋅   8.0   ⋅   ⋅
 ⋅   ⋅   ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅   9.0  ⋅
 ⋅   ⋅   ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅   ⋱
 ⋅   ⋅   ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅   ⋱
```

1.1.2 Conversion

We can convert between any two polynomial bases using a lower triangular operator, because their spans are equivalent. In the case of Chebyshev and ultraspherical polynomials, they have the added property that they are banded.

Proposition (Chebyshev T-to-U conversion)

$$\begin{aligned} T_0(x) &= U_0(x) \\ T_1(x) &= \frac{U_1(x)}{2} \\ T_n(x) &= \frac{U_n(x)}{2} - \frac{U_{n-2}(x)}{2} \end{aligned}$$

Proof

Before we do the proof, note that the fact that there are limited non-zero entries follows immediately: if $m < n-2$ we have

$$\langle T_n, U_m \rangle_{\mathrm{U}} = \left\langle T_n, (1-x^2) U_m \right\rangle_{\mathrm{T}} = 0$$

To actually determine the entries, we use the trigonometric formulae. Recall for $x = (z + z^{-1})/2$, $z = \mathrm{e}^{\mathrm{i}\theta}$, we have

$$\begin{aligned} T_n(x) &= \cos n\theta = \frac{z^{-n} + z^n}{2} \\ U_n(x) &= \frac{\sin(n+1)\theta}{\sin\theta} = \frac{z^{n+1} - z^{-n-1}}{z - z^{-1}} = z^{-n} + z^{2-n} + \cdots + \cdots + z^{n-2} + z^n \end{aligned}$$

The result follows immediately.

■

Corollary (Ultraspherical λ-to-(λ+1) conversion)

$$C_n^{(\lambda)}(x) = \frac{\lambda}{n+\lambda}(C_n^{(\lambda+1)}(x) - C_{n-2}^{(\lambda+1)}(x))$$

Proof This follows from differentiating the previous result. For example:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x} T_0(x) &= \frac{\mathrm{d}}{\mathrm{d}x} U_0(x) \\
\frac{\mathrm{d}}{\mathrm{d}x} T_1(x) &= \frac{\mathrm{d}}{\mathrm{d}x} \frac{U_1(x)}{2} \\
\frac{\mathrm{d}}{\mathrm{d}x} T_n(x) &= \frac{\mathrm{d}}{\mathrm{d}x} \left(\frac{U_n(x)}{2} - \frac{U_{n-2}}{2} \right)
\end{aligned}$$

becomes

$$\begin{aligned}
0 &= 0 \\
U_0(x) &= C_0^{(2)}(x) \\
nU_{n-1}(x) &= C_{n-1}^{(2)}(x) - C_{n-3}^{(2)}(x)
\end{aligned}$$

Differentiating this repeatedly completes the proof.

■

Note we can write this in matrix form, for example, we have

$$\underbrace{\begin{pmatrix} 1 & & & \\ 0 & \frac{1}{2} & & \\ -\frac{1}{2} & 0 & \frac{1}{2} & \\ & \ddots & \ddots & \ddots \end{pmatrix}}_{(R_T^U)^\top} \begin{pmatrix} U_0(x) \\ U_1(x) \\ U_2(x) \\ \vdots \end{pmatrix} = \begin{pmatrix} T_0(x) \\ T_1(x) \\ T_2(x) \\ \vdots \end{pmatrix}$$

therefore,

$$f(x) = (T_0(x), T_1(x), \ldots) \begin{pmatrix} f_0 \\ f_1 \\ \vdots \end{pmatrix} = (U_0(x), U_1(x), \ldots) R_T^U \begin{pmatrix} f_0 \\ f_1 \\ \vdots \end{pmatrix}$$

Again, we can construct this nicely in ApproxFun:

```
R_TU = I : Chebyshev() → Ultraspherical(1)

f = Fun(exp, Chebyshev())
g = R_TU*f

g(0.1) , exp(0.1)
```

```
(1.1051709180756477, 1.1051709180756477)
```

1.1.3 Ultraspherical Three-term recurrence

Theorem (three-term recurrence for Chebyshev U)

$$
\begin{aligned}
xU_0(x) &= \frac{U_1(x)}{2} \\
xU_n(x) &= \frac{U_{n-1}(x)}{2} + \frac{U_{n+1}(x)}{2}
\end{aligned}
$$

Proof Differentiating

$$
\begin{aligned}
xT_0(x) &= T_1(x) \\
xT_n(x) &= \frac{T_{n-1}(x)}{2} + \frac{T_{n+1}(x)}{2}
\end{aligned}
$$

we get

$$
\begin{aligned}
T_0(x) &= U_0(x) \\
T_n(x) + nxU_{n-1}(x) &= \frac{(n-1)U_{n-2}(x)}{2} + \frac{(n+1)U_n(x)}{2}
\end{aligned}
$$

substituting in the conversion $T_n(x) = (U_n(x) - U_{n-2}(x))/2$ we get

$$
\begin{aligned}
T_0(x) &= U_0(x) \\
nxU_{n-1}(x) &= \frac{(n-1)U_{n-2}(x)}{2} + \frac{(n+1)U_n(x)}{2} - (U_n(x) - U_{n-2}(x))/2 = \frac{nU_{n-2}(x)}{2} + \frac{nU_n(x)}{2}
\end{aligned}
$$

■

Differentiating this theorem again and applying the conversion we get the following

Corollary (three-term recurrence for ultraspherical)

$$
\begin{aligned}
xC_0^{(\lambda)}(x) &= \frac{1}{2\lambda} C_1^{(\lambda)}(x) \\
xC_n^{(\lambda)}(x) &= \frac{n+2\lambda-1}{2(n+\lambda)} C_{n-1}^{(\lambda)}(x) + \frac{n+1}{2(n+\lambda)} C_{n+1}^{(\lambda)}(x)
\end{aligned}
$$

Here's an example of the Jacobi operator (which is the transpose of the multiplication by x operator):

```
Multiplication(Fun(), Ultraspherical(2))'
```

```
AdjointOperator : Ultraspherical(2) → Ultraspherical(2)
 0.0                 0.25   ...   ⋅       ⋅                    ⋅    ⋅
 0.6666666666666666  0.0          ⋅       ⋅                    ⋅    ⋅
  ⋅                  0.625        ⋅       ⋅                    ⋅    ⋅
  ⋅                   ⋅           ⋅       ⋅                    ⋅    ⋅
  ⋅                   ⋅           ⋅       ⋅                    ⋅    ⋅
```

```
 ⋅                     ⋅     …    ⋅         ⋅                    ⋅     ⋅
 ⋅                     ⋅          0.4375  ⋅                     ⋅     ⋅
 ⋅                     ⋅          0.0      0.4444444444444444    ⋅     ⋅
 ⋅                     ⋅          0.55     0.0                  0.45   ⋅
 ⋅                     ⋅            ⋅     0.5454545454545454  0.0     ⋱
 ⋅                     ⋅     …    ⋅         ⋅                    ⋱       ⋱
```

1.2 Application: solving differential equations

The preceding results allowed us to represent

1. Differentiation
2. Conversion
3. Multiplication

as banded operators. We will see that we can combine these, along with

4. Evaluation

to solve ordinary differential equations.

1.2.1 First order, constant coefficient differential equations

Consider the simplest ODE:

$$
\begin{aligned}
u(0) &= 1 \\
u'(x) - u(x) &= 0, \qquad x \in [-1, 1]
\end{aligned}
$$

and suppose we represent $u(x)$ in its Chebyshev expansion, with coefficients to be determined. In other words, we want to calculate coefficients u_k such that

$$
u(x) = \sum_{k=0}^{\infty} u_k T_k(x) = (T_0(x), T_1(x), \ldots) \begin{pmatrix} u_0 \\ u_1 \\ \vdots \end{pmatrix}
$$

In this case we know that $u(x) = \mathrm{e}^x$, but we would still need other means to calculate u_k (They are definitely not as simple as Taylor series coefficients).

We can express the constraints as acting on the coefficients. For example, we have

$$
u(0) = (T_0(0), T_1(0), \ldots) \begin{pmatrix} u_0 \\ u_1 \\ \vdots \end{pmatrix} = (1, 0, -1, 0, 1, \ldots) \begin{pmatrix} u_0 \\ u_1 \\ \vdots \end{pmatrix}
$$

We also have

$$u'(x) = (U_0(x), U_1(x), \ldots) \begin{pmatrix} 0 & 1 & & & \\ & & 2 & & \\ & & & 3 & \\ & & & & \ddots \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \end{pmatrix}$$

To represent $u'(x) - u(x)$, we need to make sure the bases are compatible. In other words, we want to express $u(x)$ in its $U_k(x)$ basis using the conversion operator R_T^U:

$$u(x) = (U_0(x), U_1(x), \ldots) \begin{pmatrix} 1 & 0 & -\frac{1}{2} & & \\ & \frac{1}{2} & 0 & -\frac{1}{2} & \\ & & \ddots & \ddots & \ddots \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \end{pmatrix}$$

Which gives us,

$$u'(x) - u(x) = (U_0(x), U_1(x), \ldots) \begin{pmatrix} -1 & 1 & \frac{1}{2} & & & \\ & -\frac{1}{2} & 2 & \frac{1}{2} & & \\ & & -\frac{1}{2} & 3 & \frac{1}{2} & \\ & & & \ddots & \ddots & \ddots \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \end{pmatrix}$$

Combing the differential part and the evaluation part, we arrive at an (infinite) system of equations for the coefficients $u_0, u_1, \ldots$:

$$\begin{pmatrix} 1 & 0 & -1 & 0 & 1 & \cdots \\ -1 & 1 & \frac{1}{2} & & & \\ & -\frac{1}{2} & 2 & \frac{1}{2} & & \\ & & -\frac{1}{2} & 3 & \frac{1}{2} & \\ & & & \ddots & \ddots & \ddots \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{pmatrix}$$

How to solve this system is outside the scope of this course (though a simple approach is to truncate the infinite system to finite systems). We can however do this in ApproxFun:

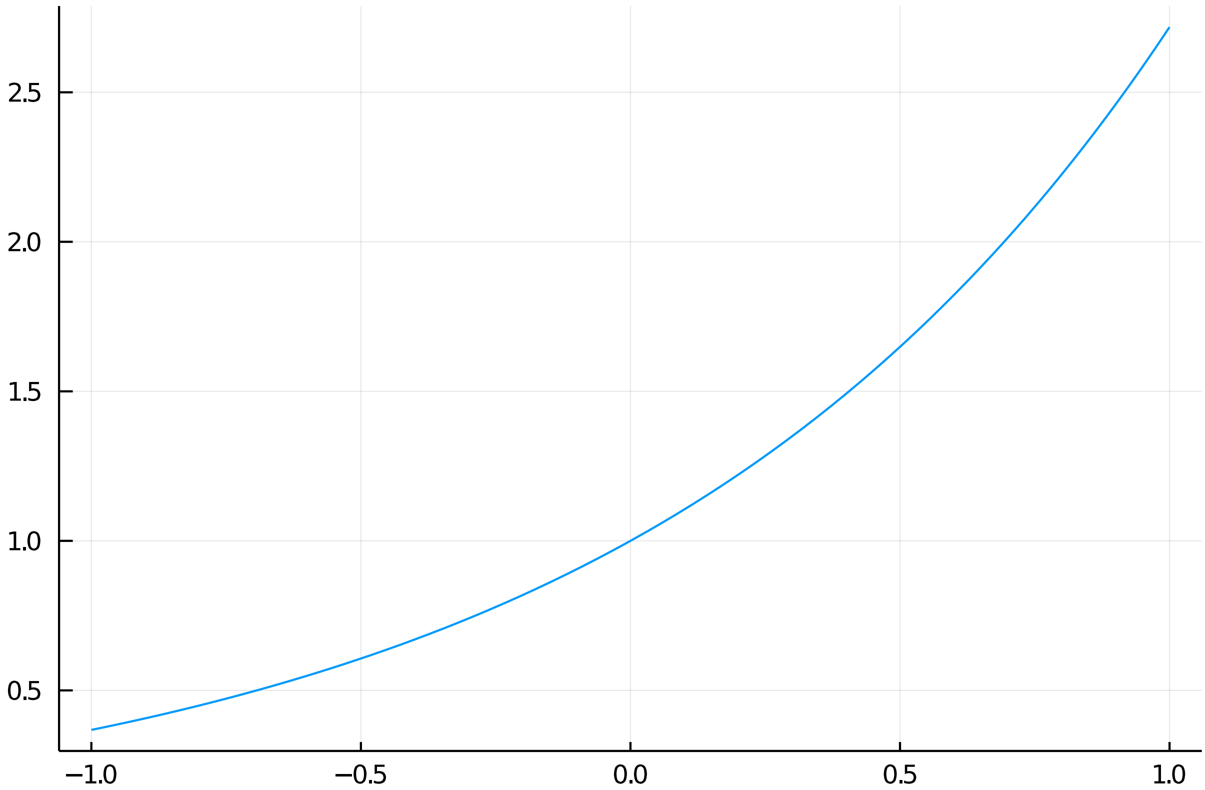
```
B  = Evaluation(0.0) : Chebyshev()
D  = Derivative() : Chebyshev() → Ultraspherical(1)
R_TU = I : Chebyshev() → Ultraspherical(1)
L = [B;
     D - R_TU]
```

```
InterlaceOperator : Chebyshev() → 2-element ArraySpace:
ApproxFunBase.Space{D,Float64} where D[ConstantSpace(Point(0.0)), Ultrasphe
rical(1)]
  1.0   0.0  -1.0   0.0   1.0   0.0  -1.0   0.0   1.0  0.0  ⋯
 -1.0   1.0   0.5   0.0   0.0   0.0   0.0   0.0   0.0  0.0  ⋱
  0.0  -0.5   2.0   0.5   0.0   0.0   0.0   0.0   0.0  0.0  ⋱
  0.0   0.0  -0.5   3.0   0.5   0.0   0.0   0.0   0.0  0.0  ⋱
  0.0   0.0   0.0  -0.5   4.0   0.5   0.0   0.0   0.0  0.0  ⋱
  0.0   0.0   0.0   0.0  -0.5   5.0   0.5   0.0   0.0  0.0  ⋱
  0.0   0.0   0.0   0.0   0.0  -0.5   6.0   0.5   0.0  0.0  ⋱
  0.0   0.0   0.0   0.0   0.0   0.0  -0.5   7.0   0.5  0.0  ⋱
```

```
0.0   0.0   0.0   0.0   0.0   0.0   0.0  -0.5   8.0  0.5  ⋱
0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  -0.5  9.0  ⋱
 ⋮     ⋱     ⋱     ⋱     ⋱     ⋱     ⋱     ⋱     ⋱     ⋱    ⋱
```

We can solve this system as follows:

```julia
u = L \ [1; 0]
plot(u;legend=false)
```



It matches the "true" result:

```julia
u(0.1) , exp(0.1)
```

```
(1.1051709180756477, 1.1051709180756477)
```

Note we can incorporate right-hand sides as well, for example, to solve $u'(x) - u(x) = f(x)$, by expanding f in its Chebyshev U series.

1.2.2 Second-order constant coefficient equations

This approach extends to second-order constant-coefficient equations by using ultraspherical polynomials. Consider

$$\begin{aligned} u(-1) &= 1 \\ u(1) &= 0 \\ u''(x) + u'(x) + u(x) &= 0 \end{aligned}$$

Evaluation works as in the first-order case. To handle second-derivatives, we need $C^{(2)}$ polynomials:

```
D_0 = Derivative() : Chebyshev() → Ultraspherical(1)
D_1 = Derivative() : Ultraspherical(1) → Ultraspherical(2)
D_1*D_0  # 2 zeros not printed in (1,1) and (1,2) entry
```

```
ConcreteDerivative : Chebyshev() → Ultraspherical(2)
 ⋅  ⋅  4.0  ⋅    ⋅    ⋅     ⋅     ⋅     ⋅     ⋅    ⋅
 ⋅  ⋅   ⋅  6.0   ⋅    ⋅     ⋅     ⋅     ⋅     ⋅    ⋅
 ⋅  ⋅   ⋅   ⋅   8.0   ⋅     ⋅     ⋅     ⋅     ⋅    ⋅
 ⋅  ⋅   ⋅   ⋅    ⋅   10.0   ⋅     ⋅     ⋅     ⋅    ⋅
 ⋅  ⋅   ⋅   ⋅    ⋅    ⋅    12.0   ⋅     ⋅     ⋅    ⋅
 ⋅  ⋅   ⋅   ⋅    ⋅    ⋅     ⋅    14.0   ⋅     ⋅    ⋅
 ⋅  ⋅   ⋅   ⋅    ⋅    ⋅     ⋅     ⋅    16.0   ⋅    ⋅
 ⋅  ⋅   ⋅   ⋅    ⋅    ⋅     ⋅     ⋅     ⋅    18.0  ⋅
 ⋅  ⋅   ⋅   ⋅    ⋅    ⋅     ⋅     ⋅     ⋅     ⋅    ⋱
 ⋅  ⋅   ⋅   ⋅    ⋅    ⋅     ⋅     ⋅     ⋅     ⋅    ⋱
 ⋅  ⋅   ⋅   ⋅    ⋅    ⋅     ⋅     ⋅     ⋅     ⋅    ⋱
```

For the identity operator, we use two conversion operators:

```
R_TU = I : Chebyshev() → Ultraspherical(1)
R_U2 = I : Ultraspherical(1) → Ultraspherical(2)
R_T2 = R_U2*R_TU
```

```
TimesOperator : Chebyshev() → Ultraspherical(2)
 1.0  0.0   -0.6666666666666666   0.0   …    ⋅                     ⋅
 ⋅
  ⋅   0.25   0.0                 -0.375      ⋅                     ⋅
 ⋅
  ⋅    ⋅     0.16666666666666666  0.0         ⋅                     ⋅
 ⋅
  ⋅    ⋅      ⋅                   0.125       ⋅                     ⋅
 ⋅
  ⋅    ⋅      ⋅                    ⋅         0.07142857142857142    ⋅
 ⋅
  ⋅    ⋅      ⋅                    ⋅    …    0.0                   0.0625
 ⋅
  ⋅    ⋅      ⋅                    ⋅        -0.12698412698412698   0.0
 ⋱
  ⋅    ⋅      ⋅                    ⋅         0.0                  -0.1125
 ⋱
  ⋅    ⋅      ⋅                    ⋅         0.05555555555555555   0.0
 ⋱
  ⋅    ⋅      ⋅                    ⋅          ⋅                    0.05
 ⋱
  ⋅    ⋅      ⋅                    ⋅    …     ⋅                     ⋅
 ⋱
```

And for the first derivative, we use a derivative and then a conversion:

```
R_U2*D_0  # or could have been D_1*R_TU
```

```
TimesOperator : Chebyshev() → Ultraspherical(2)
```

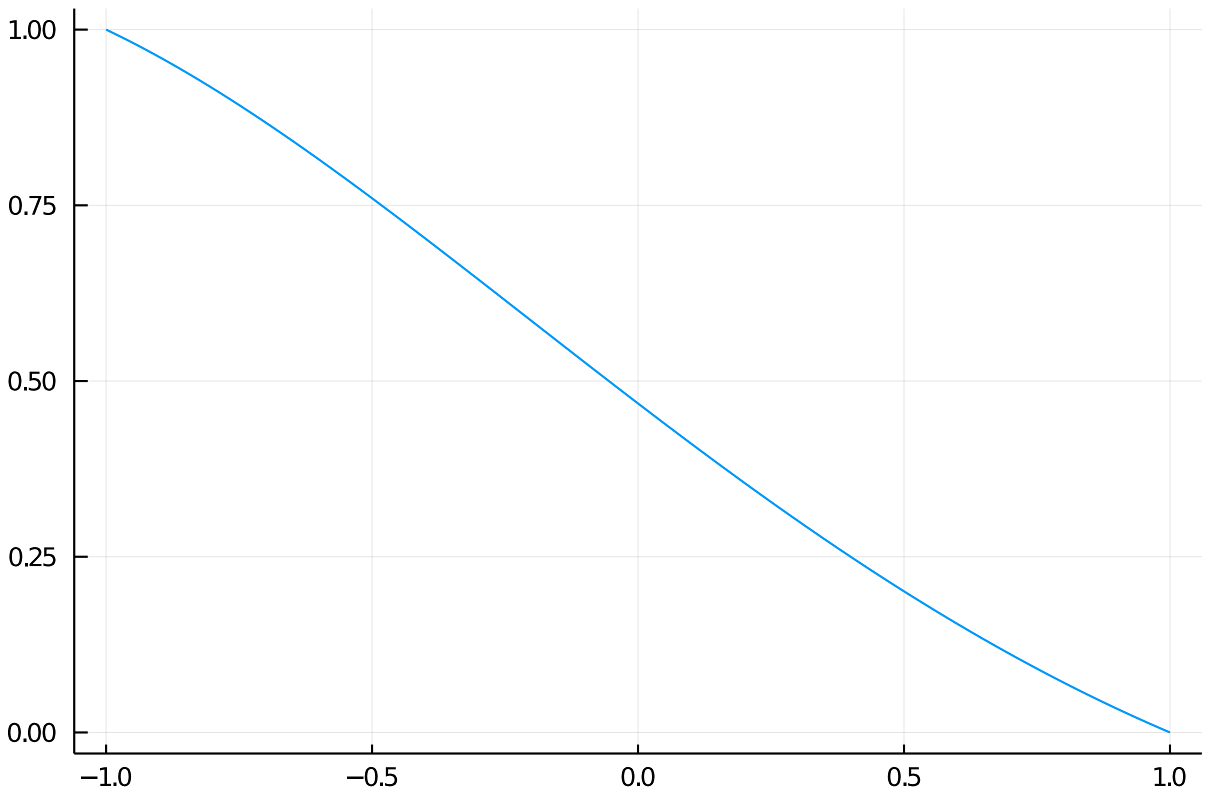
```
⋅  1.0  0.0  -1.0   ⋅     ⋅     ⋅     ⋅     ⋅     ⋅    ⋅
⋅   ⋅   1.0   0.0  -1.0   ⋅     ⋅     ⋅     ⋅     ⋅    ⋅
⋅   ⋅    ⋅    1.0   0.0  -1.0   ⋅     ⋅     ⋅     ⋅    ⋅
⋅   ⋅    ⋅     ⋅    1.0   0.0  -1.0   ⋅     ⋅     ⋅    ⋅
⋅   ⋅    ⋅     ⋅     ⋅    1.0   0.0  -1.0   ⋅     ⋅    ⋅
⋅   ⋅    ⋅     ⋅     ⋅     ⋅    1.0   0.0  -1.0   ⋅    ⋅
⋅   ⋅    ⋅     ⋅     ⋅     ⋅     ⋅    1.0   0.0  -1.0  ⋅
⋅   ⋅    ⋅     ⋅     ⋅     ⋅     ⋅     ⋅    1.0   0.0  ⋱
⋅   ⋅    ⋅     ⋅     ⋅     ⋅     ⋅     ⋅     ⋅    1.0  ⋱
⋅   ⋅    ⋅     ⋅     ⋅     ⋅     ⋅     ⋅     ⋅     ⋅   ⋱
⋅   ⋅    ⋅     ⋅     ⋅     ⋅     ⋅     ⋅     ⋅     ⋅   ⋱
```

Putting everything together we get:

```
B_-1 = Evaluation(-1) : Chebyshev()
B_1 = Evaluation(1) : Chebyshev()
# u(-1)
# u(1)
# u'' + u' +u

L = [B_-1;
     B_1;
     D_1*D_0 + R_U2*D_0 + R_U2*R_TU]

u = L \ [1.0,0.0,0.0]
plot(u;legend=false)
```



1.2.3 Variable coefficients

Consider the Airy ODE

$$
\begin{aligned}
u(-1) &= 1 \\
u(1) &= 0 \\
u''(x) - xu(x) &= 0
\end{aligned}
$$

to handle this, we need only use the Jacobi operator to represent multiplication by x:

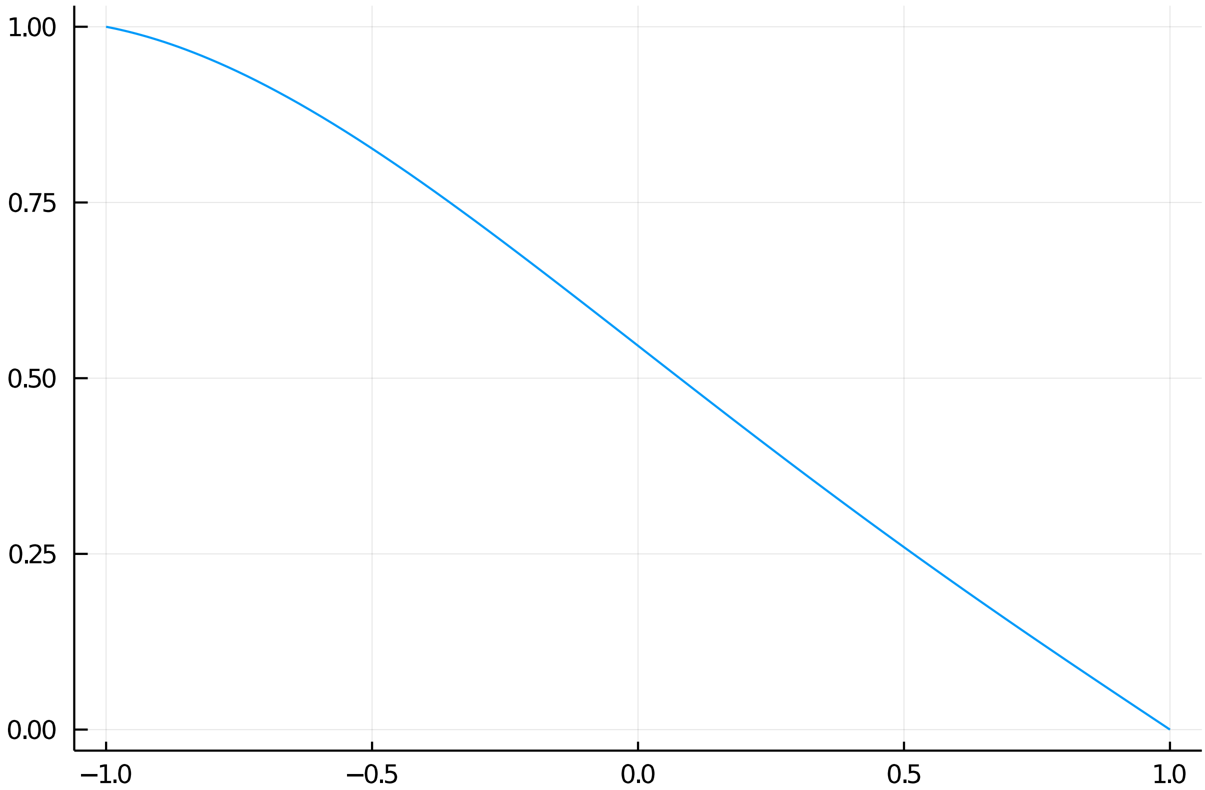
```
x = Fun()
X = Multiplication(x) : Chebyshev() → Chebyshev()  # transpose of the Jacobi operator
```

```
ConcreteMultiplication : Chebyshev() → Chebyshev()
 0.0  0.5   ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅   ⋅
 1.0  0.0  0.5   ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅   ⋅
  ⋅   0.5  0.0  0.5   ⋅    ⋅    ⋅    ⋅    ⋅    ⋅   ⋅
  ⋅    ⋅   0.5  0.0  0.5   ⋅    ⋅    ⋅    ⋅    ⋅   ⋅
  ⋅    ⋅    ⋅   0.5  0.0  0.5   ⋅    ⋅    ⋅    ⋅   ⋅
  ⋅    ⋅    ⋅    ⋅   0.5  0.0  0.5   ⋅    ⋅    ⋅   ⋅
  ⋅    ⋅    ⋅    ⋅    ⋅   0.5  0.0  0.5   ⋅    ⋅   ⋅
  ⋅    ⋅    ⋅    ⋅    ⋅    ⋅   0.5  0.0  0.5   ⋅   ⋅
  ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅   0.5  0.0  0.5  ⋅
  ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅   0.5  0.0  ⋱
  ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋅    ⋱    ⋱
```

We set up the system as follows:

```
L = [B_−_1;    # u(-1)
     B_1 ;    # u(1)
     D_1*D_0 - R_U2*R_TU*X]    # u'' - x*u

u = L \ [1.0;0.0;0.0]
plot(u; legend=false)
```



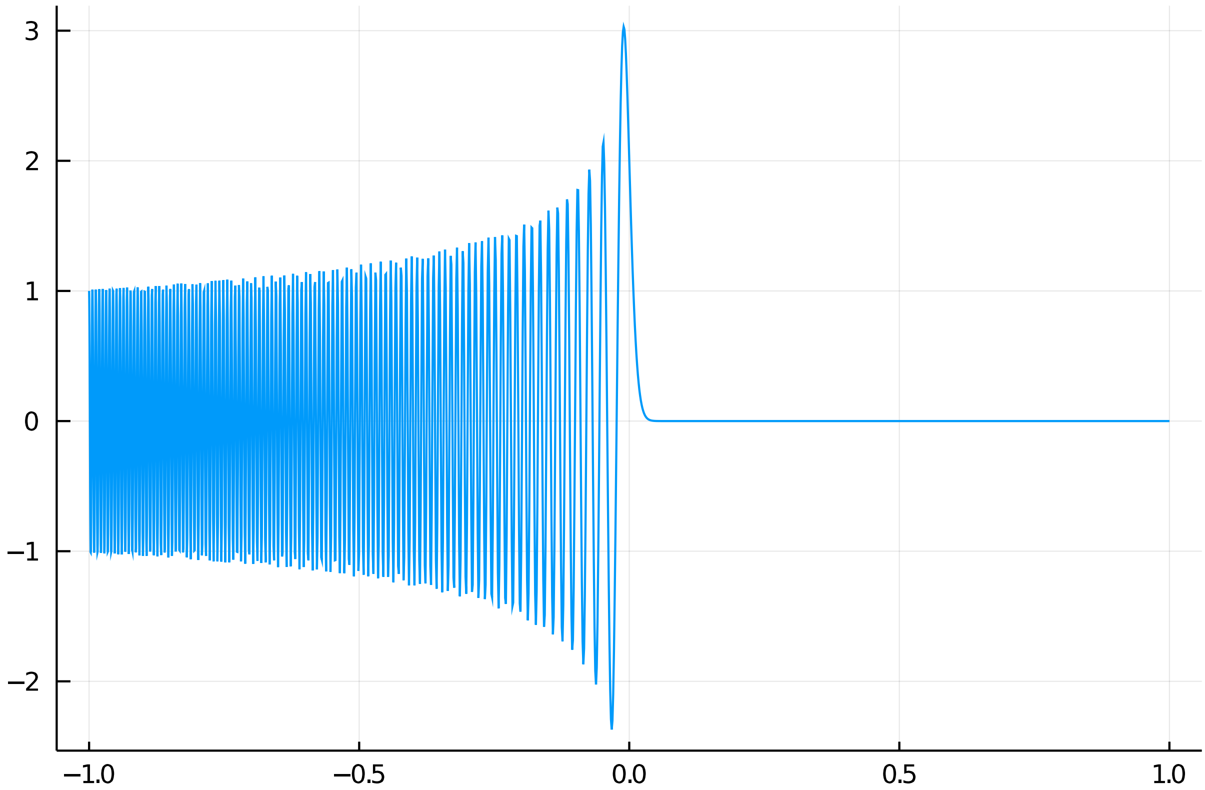
If we introduce a small parameter, that is, solve

$$
\begin{aligned}
u(-1) &= 1 \\
u(1) &= 0 \\
\epsilon u''(x) - xu(x) &= 0
\end{aligned}
$$

we can see it's pretty hard to compute solutions:

```
ε = 1E-6
L = [B_-1;
     B_1 ;
     ε*D_1*D_0 - R_U2*R_TU*X]

u = L \ [1.0;0.0;0.0]
plot(u; legend=false)
```



Because of the banded structure, this can be solved fast:

```
ε = 1E-10
L = [B_−_1;
     B_1 ;
     ε*D_1*D_0 - R_U2*R_TU*X]

@time u = L \ [1.0;0.0;0.0]
@show ncoefficients(u);

2.478050 seconds (11.40 M allocations: 271.981 MiB, 2.22% gc time)
ncoefficients(u) = 62496
```

To handle other variable coefficients, first consider a polynomial $p(x)$. If Multiplication by x is represented by multiplying the coefficients by $J^\top$, then multiplication by p is represented by multiplying the coefficients by $p(J^\top)$:

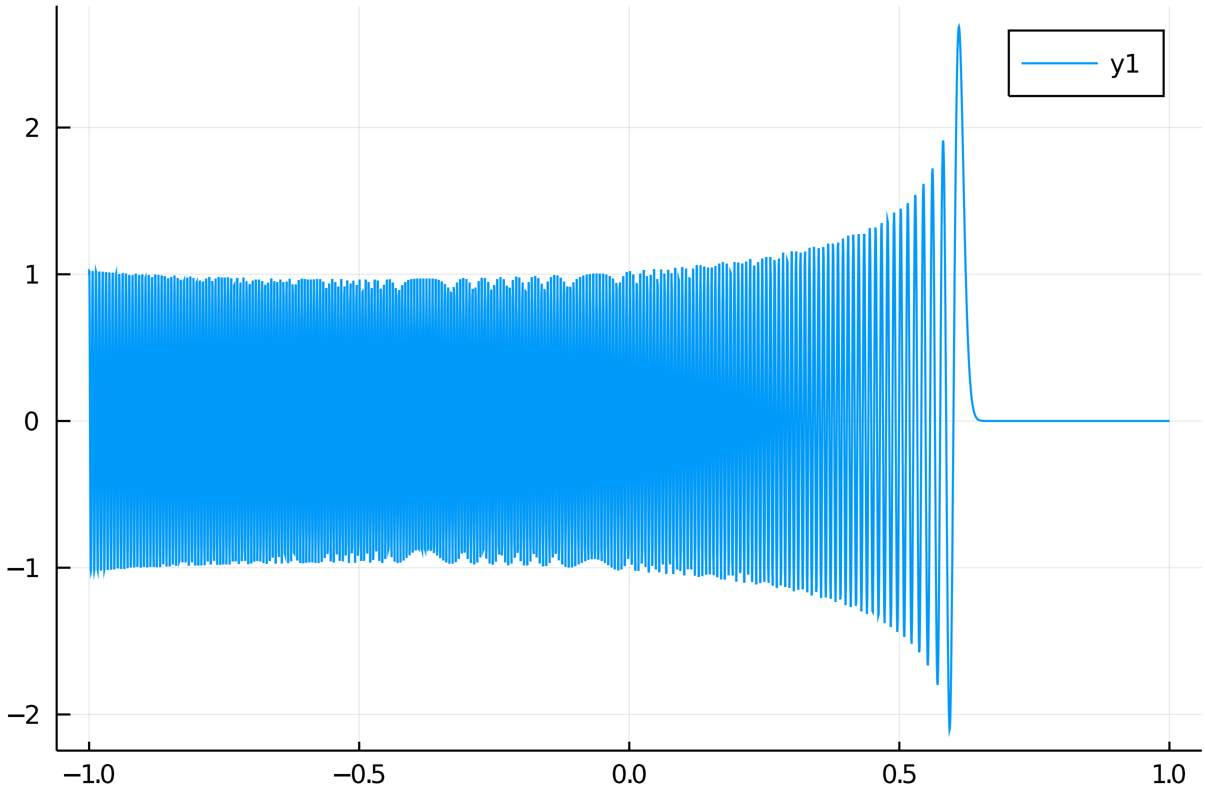
```
M = -I + X + (X)^2  # represents -1+x+x^2

ε = 1E-6
L = [B_−_1;
     B_1 ;
     ε*D_1*D_0 - R_U2*R_TU*M]

@time u = L \ [1.0;0.0;0.0]

@show ε*u''(0.1) - (-1+0.1+0.1^2)*u(0.1)
plot(u)

0.119528 seconds (245.87 k allocations: 6.237 MiB)
ε@*( * ((u(*@')')(0.1) - (-1 + 0.1 + 0.1 ^ 2) * u(0.1) = -1.3572476476042539e-14
```



For other smooth functions, we first approximate in a polynomial basis, and without loss of generality we use Chebyshev T basis. For example, consider

$$\begin{aligned} u(-1) &= 1 \\ u(1) &= 0 \\ \epsilon u''(x) - \mathrm{e}^x u(x) &= 0 \end{aligned}$$

where

$$\mathrm{e}^x \approx p(x) = \sum_{k=0}^{m-1} p_k T_k(x)$$

Evaluating at a point x, recall Clenshaw's algorithm:

$$\begin{aligned} \gamma_{n-1} &= 2p_{n-1} \\ \gamma_{n-2} &= 2p_{n-2} + 2x\gamma_{n-1} \\ \gamma_{n-3} &= 2p_{n-3} + 2x\gamma_{n-2} - \gamma_{n-1} \\ &\vdots \\ \gamma_1 &= p_1 + x\gamma_2 - \frac{1}{2}\gamma_3 \\ p(x) = \gamma_0 &= p_0 + x\gamma_1 - \frac{1}{2}\gamma_2 \end{aligned}$$

If multiplication by x becomes $J^\top$, then multiplication by $p(x)$ becomes $p(J^\top)$, and hence we calculate:

$$
\begin{aligned}
\Gamma_{n-1} &= 2p_{n-1}I \\
\Gamma_{n-2} &= 2p_{n-2}I + 2J^\top\Gamma_{n-1} \\
\Gamma_{n-3} &= 2p_{n-3}I + 2J^\top\Gamma_{n-2} - \Gamma_{n-1} \\
&\vdots \\
\Gamma_1 &= p_1 I + J^\top\Gamma_2 - \frac{1}{2}\Gamma_3 \\
p(J^\top) = \Gamma_0 &= p_0 + J^\top\Gamma_1 - \frac{1}{2}\Gamma_2
\end{aligned}
$$

Here is an example:

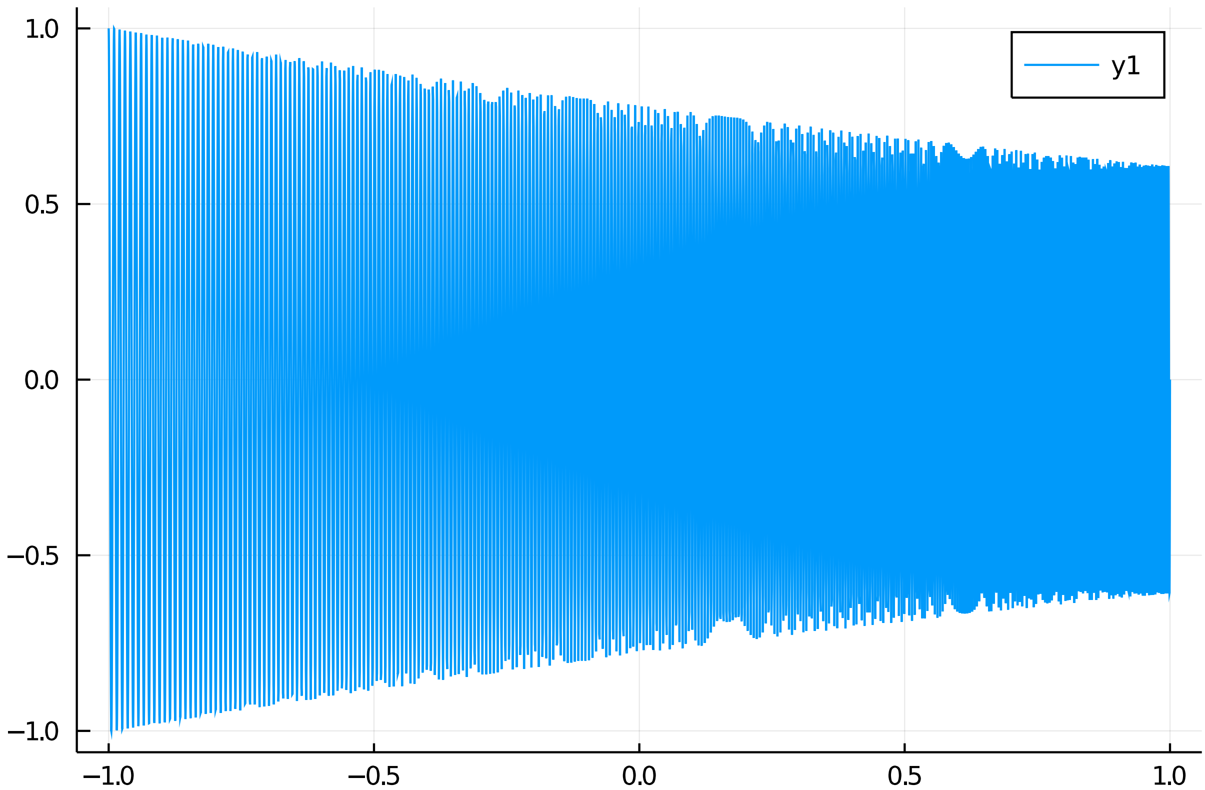
```
p = Fun(exp, Chebyshev()) # polynomial approximation to exp(x)
M = Multiplication(p) : Chebyshev() # constructed using Clenshaw:

ε = 1E-6
L = [B_-1;
     B_1 ;
     ε*D_1*D_0 + R_U2*R_TU*M]

@time u = L \ [1.0;0.0;0.0]

@show ε*u''(0.1) + exp(0.1)*u(0.1)
plot(u)

0.316422 seconds (908.05 k allocations: 20.038 MiB)
ε@*( * ((u(*@')')(0.1) + exp(0.1) * u(0.1) = 4.773959005888173e-15
```



2 Differential equations satisfied by orthogonal polynomials

This lecture we do the following:

1. Differential equations for orthogonal polynomials
   - Sturm–Liouville equations
   - Weighted differentiation for ultraspherical polynomials
   - Differential equation for ultraspherical polynomials
2. Application: Eigenstates of Schrödinger operators with quadratic potentials

The three classical weights are (Hermite) $w(x) = \mathrm{e}^{-x^2}$, (Laguerre) $w_\alpha(x) = x^\alpha \mathrm{e}^{-x}$ and (Jacobi) $w_{\alpha,\beta}(x) = (1-x)^\alpha (1+x)^\beta$. Note all weights form a simple hierarchy: when differentiated, they give a linear polynomial times the previous weight in the hierarchy. For Hermite,

$$\frac{\mathrm{d}}{\mathrm{d}x} w(x) = -2x w(x)$$

for Laguerre,

$$\frac{\mathrm{d}}{\mathrm{d}x} w^{(\alpha)}(x) = (\alpha - x) w^{(\alpha-1)}(x)$$

and for Jacobi

$$\frac{\mathrm{d}}{\mathrm{d}x} w^{(\alpha,\beta)}(x) = (\beta(1-x) - \alpha(1+x)) w^{(\alpha-1,\beta-1)}(x)$$

These relationships lead to simple differential equations that have the classical orthogonal polynomials as eigenfunctions.

2.0.1 Sturm–Liouville operator

We first consider a simple class of operators that are self-adjoint:

Proposition (Sturm–Liouville self-adjointness) Consider the weighted inner product

$$\langle f, g \rangle_w = \int_a^b f(x) g(x) w(x) \mathrm{d}x$$

then for any continuously differentiable function $q(x)$ satisfying $q(a) = q(b) = 0$, the operator

$$Lu = \frac{1}{w(x)} \frac{\mathrm{d}}{\mathrm{d}x} \left[q(x) \frac{\mathrm{d}u}{\mathrm{d}x} \right]$$

is self-adjoint in the sense

$$\langle Lf, g \rangle_w = \langle f, Lg \rangle_w$$

Proof Simple integration by parts argument:

$$
\begin{aligned}
\langle Lf, g\rangle_w &= \int_a^b \frac{\mathrm{d}}{\mathrm{d}x}\left[q(x)\frac{\mathrm{d}u}{\mathrm{d}x}\right] g(x)\mathrm{d}x \\
&= -\int_a^b q(x)\frac{\mathrm{d}u}{\mathrm{d}x}\frac{\mathrm{d}g}{\mathrm{d}x}\mathrm{d}x = \int_a^b u(x)\frac{\mathrm{d}}{\mathrm{d}x}\left[q(x)\frac{\mathrm{d}g}{\mathrm{d}x}\right]\mathrm{d}x \\
&= \int_a^b u(x)\frac{1}{w(x)}\frac{\mathrm{d}}{\mathrm{d}x}\left[q(x)\frac{\mathrm{d}g}{\mathrm{d}x}\right] w(x)\mathrm{d}x = \langle f, Lg\rangle_w
\end{aligned}
$$

■

We claim that the classical orthogonal polynomials are eigenfunctions of a Sturm–Liouville problem, that is, in each case there exists a $q(x)$ so that

$$
Lp_n(x) = \lambda_n p_n(x)
$$

where λ_n is the (real) eigenvalue. We will derive this for the ultraspherical polynomials.

2.0.2 Weighted differentiation for ultraspherical polynomials

We have already seen that Chebyshev and ultraspherical polynomials have simple expressions for derivatives where we decrement the degree and increment the parameter:

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x}T_n(x) &= nU_{n-1}(x) = nC_{n-1}^{(1)}(x) \\
\frac{\mathrm{d}}{\mathrm{d}x}C_n^{(\lambda)}(x) &= 2\lambda C_{n-1}^{(\lambda+1)}(x)
\end{aligned}
$$

In this section, we see that differentiating the weighted polynomials actually decrements the parameter and increments the degree:

Proposition (weighted differentiation)

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x}[\sqrt{1-x^2}U_n(x)] &= -\frac{n+1}{\sqrt{1-x^2}}T_{n+1}(x) \\
\frac{\mathrm{d}}{\mathrm{d}x}[(1-x^2)^{\lambda-\frac{1}{2}}C_n^{(\lambda)}(x)] &= -\frac{(n+1)(n+2\lambda-1)}{2(\lambda-1)}(1-x^2)^{\lambda-\frac{3}{2}}C_{n+1}^{(\lambda-1)}(x)
\end{aligned}
$$

Proof We show the first result by showing that the left-hand side is orthogonal to all polynomials of degree less than $n+1$ by integration by parts:

$$
\left\langle \sqrt{1-x^2}\frac{\mathrm{d}}{\mathrm{d}x}[\sqrt{1-x^2}U_n(x)], p_m(x)\right\rangle_{\mathrm{T}} = -\int_{-1}^1 \sqrt{1-x^2}U_n(x)p_m'\mathrm{d}x = 0
$$

Note that

$$
\sqrt{1-x^2}\frac{\mathrm{d}}{\mathrm{d}x}\sqrt{1-x^2}f(x) = (1-x^2)f'(x) - xf(x)
$$

Thus we just have to verify the constant in front:

$$\sqrt{1-x^2}\frac{\mathrm{d}}{\mathrm{d}x}[\sqrt{1-x^2}U_n(x) = (-n-1)2^n x^{n+1}$$

The other ultraspherical polynomial follow similarly. ■

2.0.3 Eigenvalue equation for Ultraspherical polynomials

Note that differentiating increments the parameter and decrements the degree while weighted differentiation decrements the parameter and increments the degree. Therefore combining them brings us back to where we started.

In the case of Chebyshev polynomials, this gives us a Sturm–Liouville equation:

$$\sqrt{1-x^2}\frac{\mathrm{d}}{\mathrm{d}x}\sqrt{1-x^2}\frac{\mathrm{d}T_n}{\mathrm{d}x} = n\sqrt{1-x^2}\frac{\mathrm{d}}{\mathrm{d}x}\sqrt{1-x^2}U_{n-1}(x) = -n^2 T_n(x)$$

Note that the chain rule gives us a simple expression as

$$(1-x^2)\frac{\mathrm{d}^2 T_n}{\mathrm{d}x^2} - x\frac{\mathrm{d}T_n}{\mathrm{d}x} = -n^2 T_n(x)$$

Similarly,

$$(1-x^2)^{\frac{1}{2}-\lambda}\frac{\mathrm{d}}{\mathrm{d}x}(1-x^2)^{\lambda+\frac{1}{2}}\frac{\mathrm{d}C_n^{(\lambda)}}{\mathrm{d}x} = -n(n+2\lambda)C_n^{(\lambda)}(x)$$

or in other words,

$$(1-x^2)\frac{\mathrm{d}^2 C_n^{(\lambda)}}{\mathrm{d}x^2} - (2\lambda+1)x\frac{\mathrm{d}C_n^{(\lambda)}}{\mathrm{d}x} = -\frac{n(n+2\lambda)}{2\lambda}C_n^{(\lambda)}(x)$$