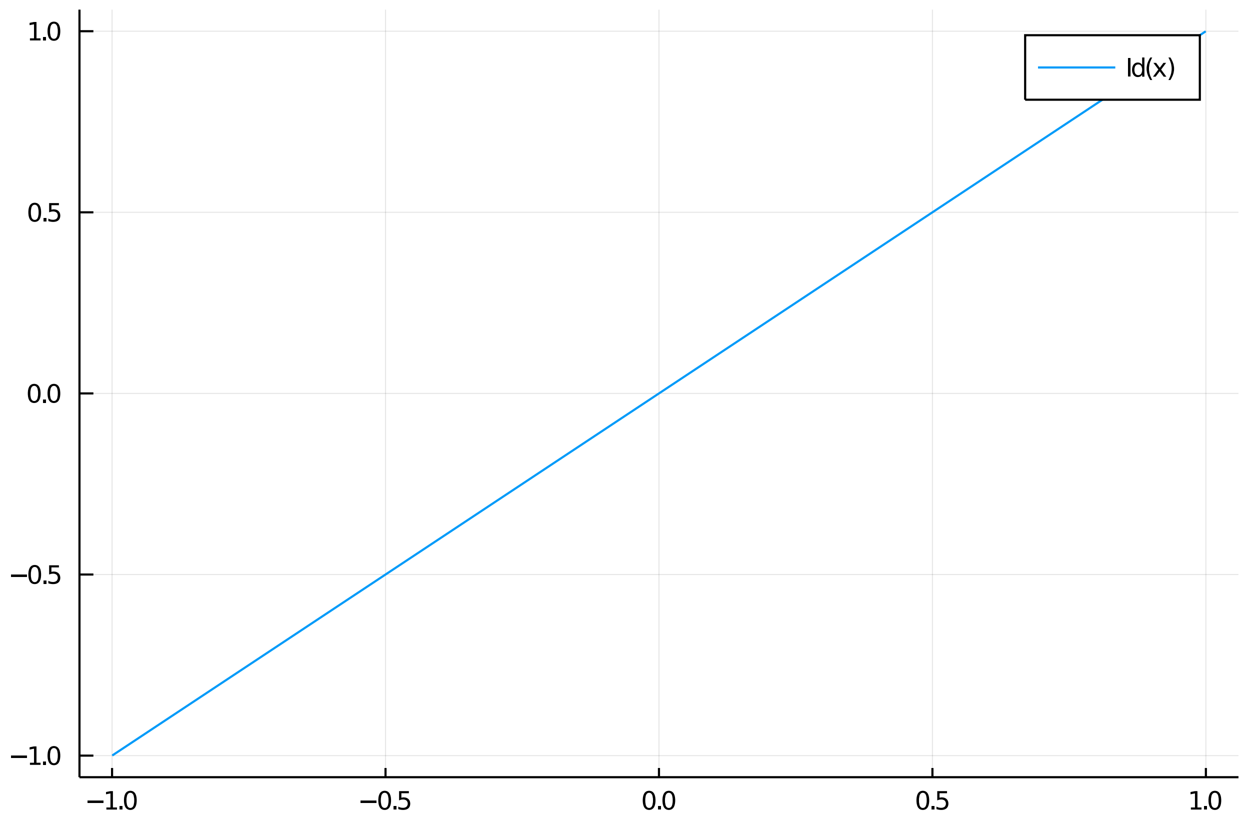
```julia
using ApproxFun, LinearAlgebra, Plots
```

ApproxFun is a Julia package for numerically computing with functions. The core concept is that it represents a function by a Julia struct `Fun` with two fields: `coefficients` to store the floating-point data; and `space` to distinguish how the coefficients are interpreted.

Let's start by creating the identity function.

```
x = Fun(Chebyshev(),[0.0, 1.0])
```

```julia
x = Fun(identity)
```




```julia
plot(x; label="Id(x)")
```

The default behaviour is to use the `Chebyshev()` approximation space so that the coefficients are first kind Chebyshev coefficients of the function `x`. A `Fun` `f` in the `Chebyshev()` space is the numerical implementation of:

$$f(x) = \sum_{k=0}^{n} c_k T_k(x) \qquad T_k(x) = \cos(k \cos^{-1}(x)).$$

The `Fun` can be evaluated at any point in its domain.

```
-1.0..1.0 (Chebyshev)
```

```julia
domain(x)
```

```
0.5
```

```julia
x(0.5)
```

or extrapolation can be used outside $[-1, 1]$.

```
1.5
```

```julia
extrapolate(x, 1.5)
```

It can also be differentiated by using the apostrophe, `'`.

```
Fun(Chebyshev(),[1.0])
```

```julia
x'
```

Let's now use `x` to create a weight function for which we will compute the first few orthonormal polynomials through the function `lanczos`, which implements Gram–Schmidt orthonormalization (compare Lemma 3.2.6 to the **source**).

```
w = Fun(Chebyshev(),[1.0])
```

```julia
w = one(x)
```

```
([Fun(Chebyshev(),[0.707107]), Fun(Chebyshev(),[0.0, 1.22474]), Fun(Chebyshev(),[0.
```

```julia
P, β, γ = lanczos(w, 10)
```

We can check that they're degree-graded by using `ncoefficients` to return the number of coefficients in each expansion.

```
Int64[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```julia
[ncoefficients(P[j]) for j in 1:length(P)]
```

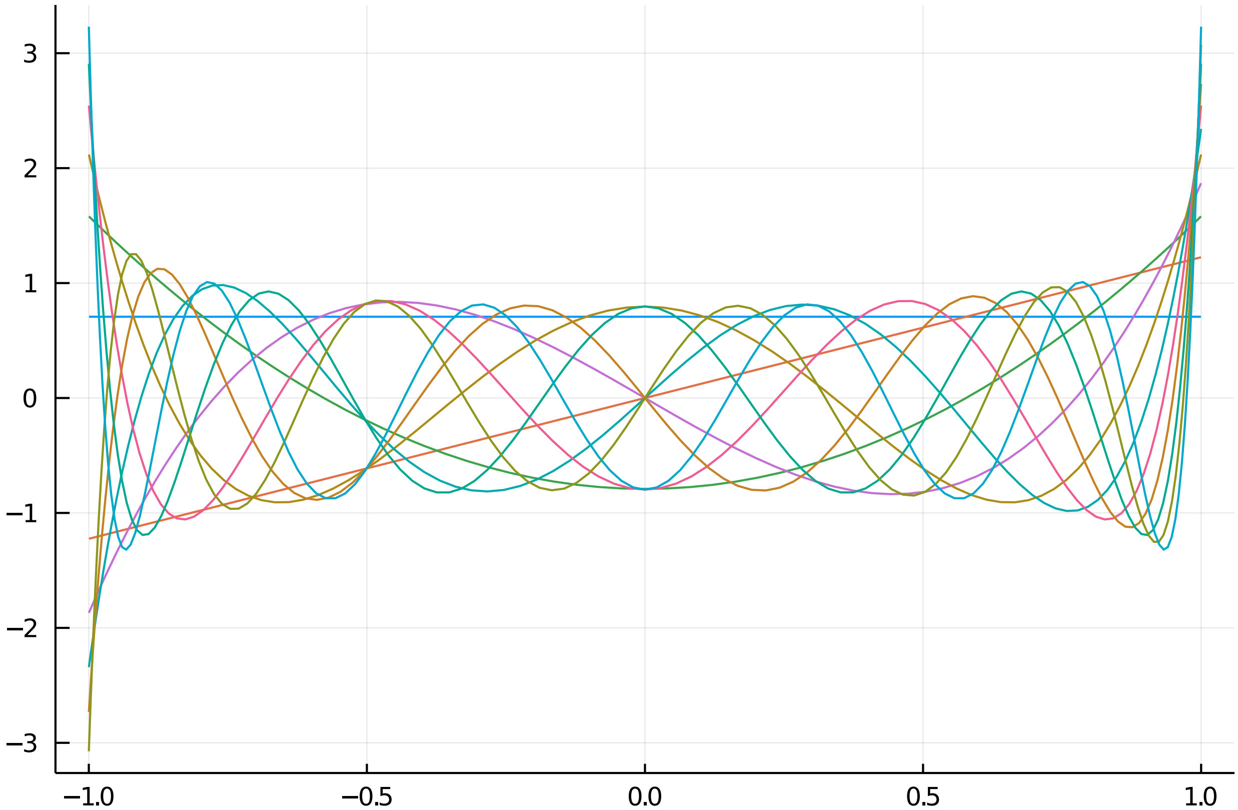
We can also check that they're orthonormal by using `sum`, which has a method to mean "definite integral." We'll compare the array of inner products against the identity and take the Frobenius norm of the difference.

```
3.0844799296940383e-15
```

```julia
norm([sum(P[k]*P[j]*w) for k in 1:length(P), j in 1:length(P)] - I)
```

Last but not least, we can plot them, where it is easy to observe the consequence of Theorem 3.2.8: the roots are all in

$(-1, 1)$. With $w(x) = 1$, these polynomials are the orthonormalized Legendre polynomials, though they are expanded in the first kind Chebyshev polynomial basis.




```
begin
    p = plot(P[1]; legend=false)
    for k in 2:length(P)
        plot!(pad(P[k], 3*ncoefficients(P[k])))
    end
    p
end
```

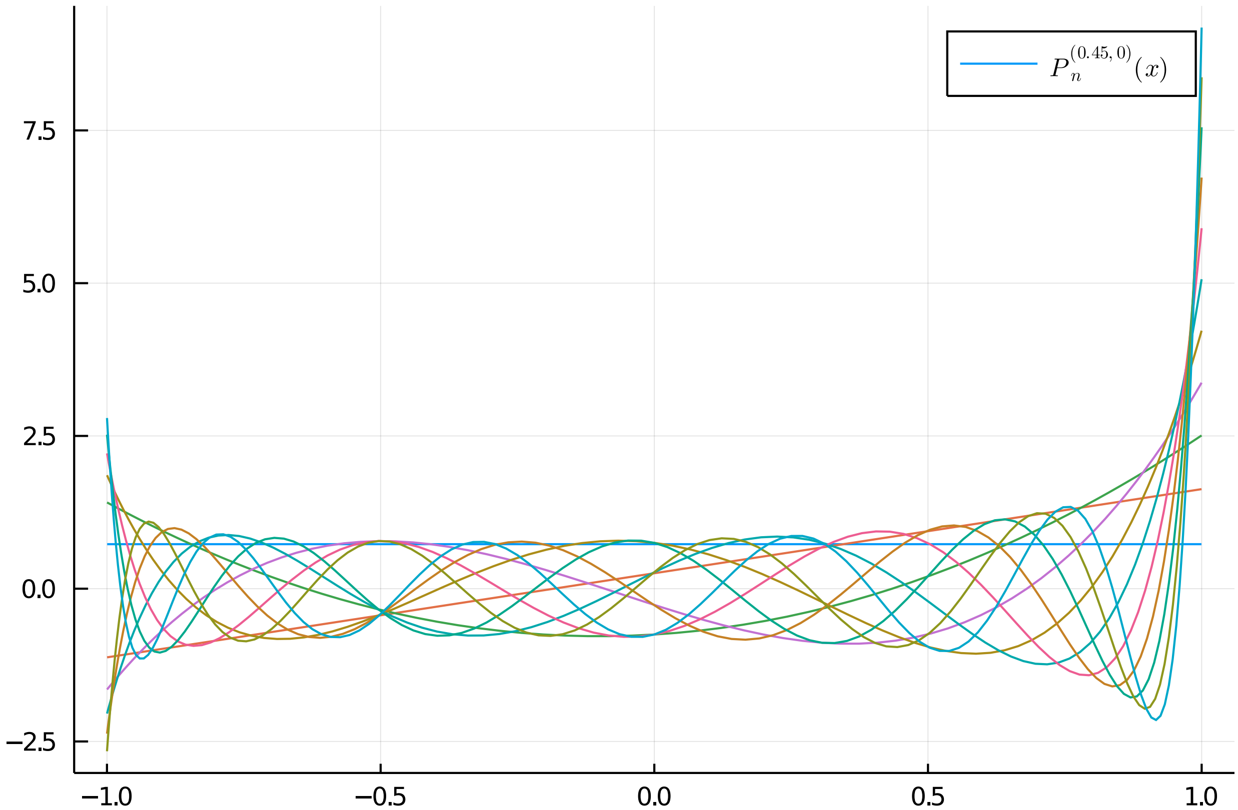
Binding a Julia variable a to an html widget gives us some freedom to explore orthogonal polynomials reactively. In `ApproxFun`, the weight function can't be just anything, so let me show you a couple examples to give you an idea.

```
@bind a html"""<input type='range' min = 0.0 max = 0.9 step = 0.01>"""
```

w1 = Fun((1-x)^0.45[Chebyshev()],[1.0])

```
w1 = (1-x)^a
```

```
P1, β1, γ1 = lanczos(w1, 10);
```

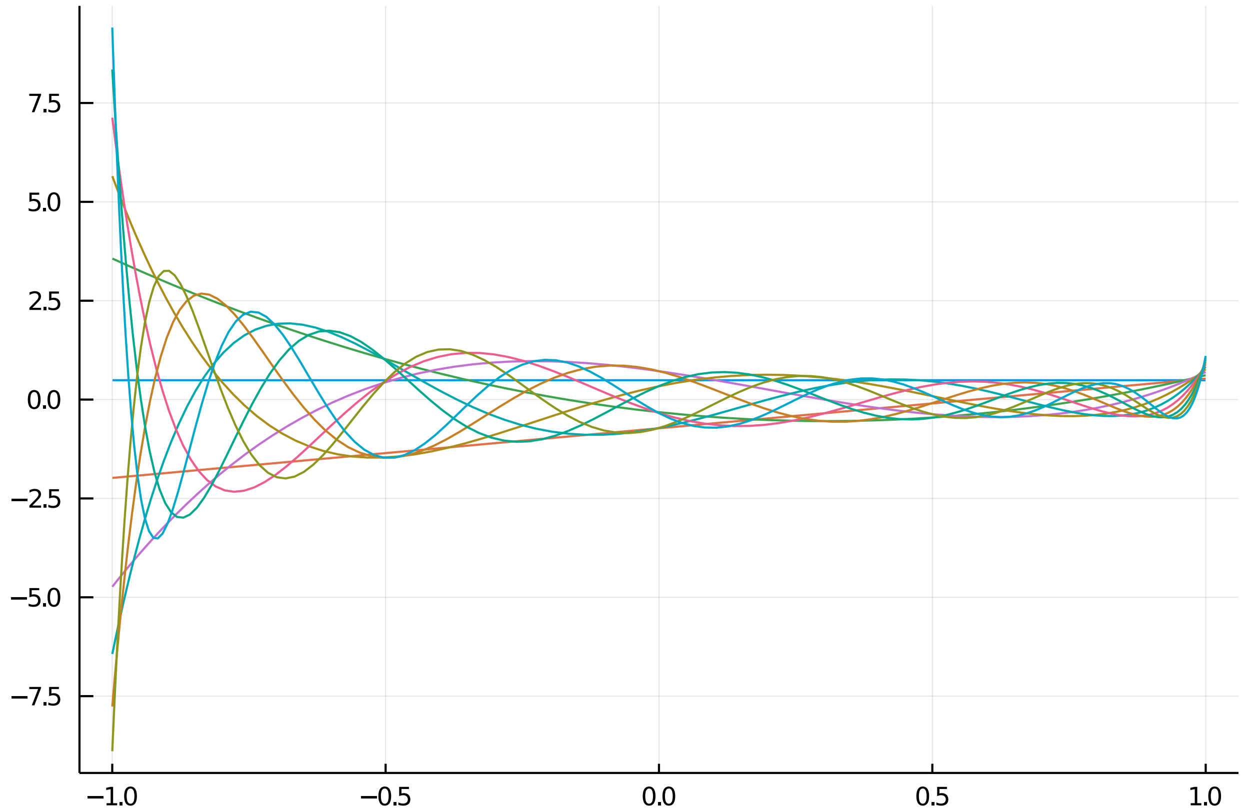



```
begin
    p1 = plot(P1[1]; label="\$P_n^{($a, 0)}(x)\$") # Jacobi polynomials
    for k in 2:length(P1)
        plot!(pad(P1[k], 3*ncoefficients(P1[k])); label="")
    end
    p1
end
```

```
w2 = Fun(Chebyshev(),[2.727078307190795, 4.007934913859185, 1.8915478020623147, 0.6451832657484051,
0.17105909339990077, 0.03697315588209169, 0.006733956146159941, 0.0010587231025719968, 0.000146
34573015640648, 1.8042354793106455e-5  …  1.890856718670662e-8, 1.6250848491490758e-9, 1.298089
2987284492e-10, 9.684832953672744e-12, 6.778238238737097e-13, 4.467190300697666e-14, 2.78173374
10336772e-15, 1.6415936317108726e-16, 9.203650625555105e-18, -4.9249661799454e-19])
```

```
w2 = exp(5*a*x) # A non-classical weight.
```

```
P2, β2, γ2 = lanczos(w2, 10);
```

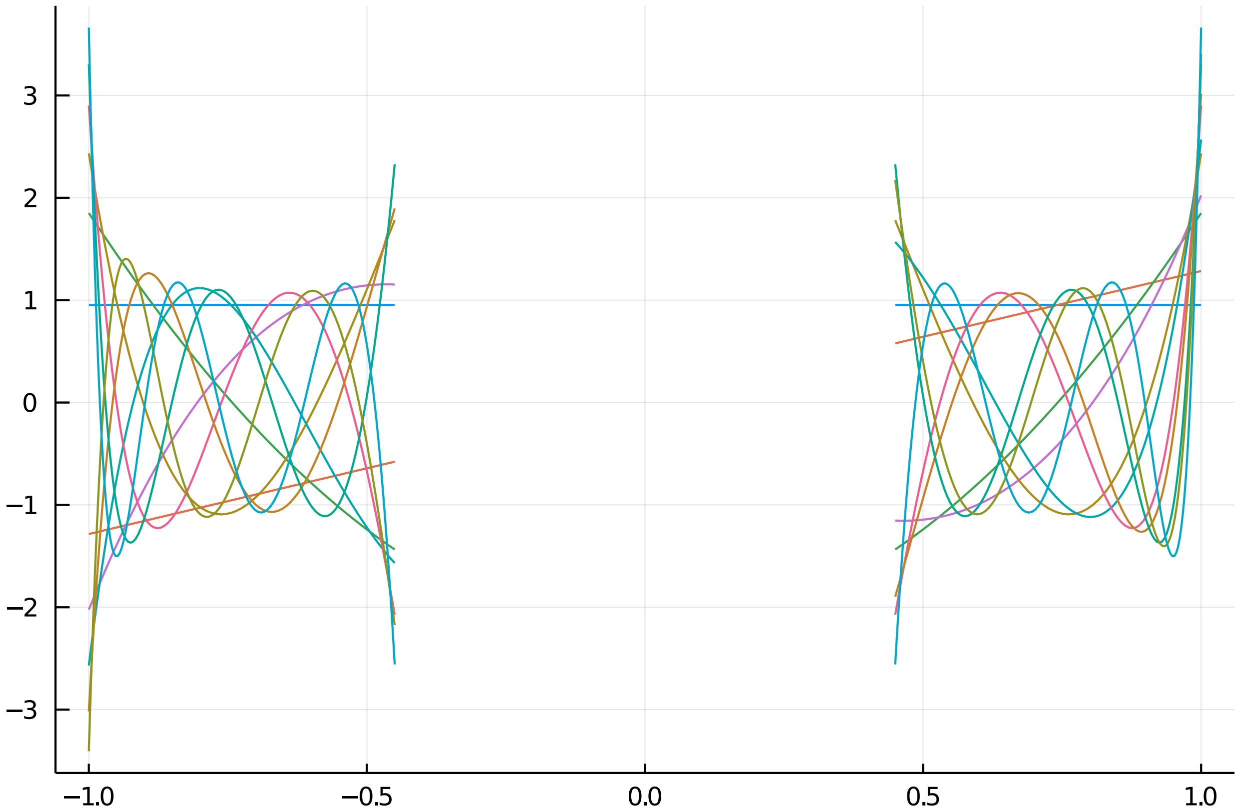



```julia
begin
    p2 = plot(P2[1]; legend=false)
    for k in 2:length(P2)
        plot!(pad(P2[k], 3*ncoefficients(P2[k])))
    end
    p2
end
```

```
w3 = Fun(Chebyshev(-1.0..-0.45)⊎Chebyshev(0.45..1.0),[1.0, 1.0])
```

```julia
w3 = Fun(one, Chebyshev(-1..(-a))∪Chebyshev(a..1)) # A constant weight on a piecewise-defined domain, also non-classical.
```

```julia
P3, β3, γ3 = lanczos(w3, 10);
```




```
begin
    p3 = plot(P3[1]; legend=false)
    for k in 2:length(P3)
        plot!(pad(P3[k], 3*ncoefficients(P3[k])))
    end
    p3
end
```