

1 Fourier methods for periodic functions

We assume that f is 2π -periodic and its Fourier series converges on $x \in [0, 2\pi)$, i.e.,

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{ikx}$$

where the Fourier coefficients are defined as

$$c_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx.$$

Suppose we only know the values of $f(x)$ at the n equally spaced points $x_j = jh = \frac{2\pi j}{n}$ for $j = 0, \dots, n - 1$, can we somehow use Fourier series to approximate f and its derivatives?

The idea is to use a truncated Fourier series and approximate Fourier coefficients \tilde{c}_k to approximate f with

$$f(x) \approx \sum_{k=-(n-1)/2}^{(n-1)/2} \tilde{c}_k e^{ikx},$$

assuming n is odd.

1.1 Trapezoidal rule

We are going to approximate the Fourier coefficients c_k by using the *trapezoidal rule*.

Trapezoidal rule: Let x_j , $j = 0, \dots, n$ be $n + 1$ equally spaced points on the interval $[a, b]$. Hence, $x_j = a + jh$ with $h = (b - a)/n$. The $n + 1$ -point trapezoidal rule for approximating the integral

$$I[g] = \int_a^b g(x)dx$$

is denoted by $I_n[g]$ and defined as

$$I_n[g] := \frac{h}{2} (g(x_0) + 2g(x_1) + 2g(x_2) + \cdots + 2g(x_{n-1}) + g(x_n))$$

The trapezoidal rule is an example of a quadrature method, which are methods to approximate integrals by weighted sums.

Let's use the trapezoidal rule to approximate the Fourier coefficients. Setting $a = 0$, $b = 2\pi$, $h = 2\pi/n$ and using the fact that $g(x) = f(x)e^{-ikx}$ is a 2π -periodic function (since $f(x)$ is assumed to be 2π -periodic), it follows that $g(x_0) = g(x_0 + 2\pi) = g(x_n)$ and we obtain the approximation

$$\begin{aligned}
c_k &= \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx \\
&\approx \frac{1}{2\pi} I_n [f e^{-ikx}] \\
&= \frac{1}{n} \sum_{j=0}^{n-1} f(x_j) e^{-ikx_j} \\
&:= \tilde{c}_k
\end{aligned}$$

Later we'll see that the trapezoidal rule gives spectacularly accurate approximations to the Fourier coefficients of smooth periodic functions (the approximations \tilde{c}_k converge exponentially fast to c_k with n for sufficiently smooth functions f). The first step in analysing the accuracy of the approximations \tilde{c}_k is the following result:

Lemma (discrete orthogonality) For $x_j = jh = \frac{2\pi j}{n}$, we have

$$I_n [e^{ikx}] = \frac{1}{n} \sum_{j=0}^{n-1} e^{ikx_j} = \begin{cases} 1 & \text{if } k = \dots, -2n, -n, 0, n, 2n, \dots \\ 0 & \text{otherwise} \end{cases},$$

therefore

$$I_n [e^{i(k-\ell)x}] = \frac{1}{n} \sum_{j=0}^{n-1} e^{i(k-\ell)x_j} = \begin{cases} 1 & \text{if } k - \ell = \dots, -2n, -n, 0, n, 2n, \dots \\ 0 & \text{otherwise} \end{cases}.$$

Proof Case 1: $k = np$, $p \in \mathbb{Z}$, then

$$\sum_{j=0}^{n-1} e^{ikx_j} = \sum_{j=0}^{n-1} e^{2\pi i k j / n} = \sum_{j=0}^{n-1} e^{2\pi i p j} = \sum_{j=0}^{n-1} 1 = n.$$

Case 2: $k \neq np$, $p \in \mathbb{Z}$, then

$$\sum_{j=0}^{n-1} e^{ikx_j} = \sum_{j=0}^{n-1} \left(e^{2\pi i k / n} \right)^j = \frac{1 - (e^{2\pi i k / n})^n}{1 - e^{2\pi i k / n}} = 0,$$

where we have used the formula

$$\sum_{j=0}^{n-1} z^j = \frac{1 - z^n}{1 - z}, \quad z \neq 1,$$

and the fact that $e^{2\pi i k / n} \neq 1$ since $k \neq np$.

Corollary The approximate Fourier coefficients \tilde{c}_k and the exact Fourier coefficients c_k are related as follows,

$$\tilde{c}_k = \cdots + c_{k-2n} + c_{k-n} + c_k + c_{k+n} + c_{k+2n} + \cdots$$

Proof We have

$$\begin{aligned}
\tilde{c}_k &= \frac{1}{n} \sum_{j=0}^{n-1} f(x_j) e^{-ikx_j} \\
&= \frac{1}{n} \sum_{j=0}^{n-1} \sum_{\ell=-\infty}^{\infty} c_{\ell} e^{i\ell x_j} e^{-ikx_j} \\
&= \sum_{\ell=-\infty}^{\infty} c_{\ell} \left(\frac{1}{n} \sum_{j=0}^{n-1} e^{i(\ell-k)x_j} \right)
\end{aligned}$$

from which the result follows.

Note from the definition of \tilde{c}_k that we have

$$\tilde{c}_{k+pn} = \tilde{c}_k, \quad p \in \mathbb{Z}.$$

We can express the approximate Fourier coefficients \tilde{c}_k as the following matrix vector product:

$$\underbrace{\begin{pmatrix} \tilde{c}_0 \\ \tilde{c}_1 \\ \vdots \\ \tilde{c}_{n-2} \\ \tilde{c}_{n-1} \end{pmatrix}}_{\tilde{\mathbf{c}}} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & e^{-ix_1} & e^{-ix_2} & \cdots & e^{-ix_{n-1}} \\ 1 & e^{-i2x_1} & e^{-i2x_2} & \cdots & e^{-i2x_{n-1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & e^{-i(n-1)x_1} & e^{-i(n-1)x_2} & \cdots & e^{-i(n-1)x_{n-1}} \end{bmatrix} \underbrace{\begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) \end{pmatrix}}_{\mathbf{f}}.$$

We express this as

$$\tilde{\mathbf{c}} = \frac{1}{\sqrt{n}} Q_n \mathbf{f}$$

where Q_n is the *Discrete Fourier Transform (DFT)* matrix.

Definition (DFT) The *Discrete Fourier Transform (DFT)* matrix is defined as:

$$Q_n := \frac{1}{\sqrt{n}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & e^{-ix_1} & e^{-ix_2} & \dots & e^{-ix_{n-1}} \\ 1 & e^{-i2x_1} & e^{-i2x_2} & \dots & e^{-i2x_{n-1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & e^{-i(n-1)x_1} & e^{-i(n-1)x_2} & \dots & e^{-i(n-1)x_{n-1}} \end{bmatrix}$$

Note that

$$Q_n^* = \frac{1}{\sqrt{n}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & e^{ix_1} & e^{i2x_1} & \dots & e^{i(n-1)x_1} \\ 1 & e^{ix_2} & e^{i2x_2} & \dots & e^{i(n-1)x_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & e^{ix_{n-1}} & e^{i2x_{n-1}} & \dots & e^{i(n-1)x_{n-1}} \end{bmatrix}$$

Proposition (DFT is Unitary) Q_n is unitary: $Q_n^* Q_n = Q_n Q_n^* = I$.

Proof

$$Q_n Q_n^* = \begin{bmatrix} I_n[1] & I_n[e^{ix}] & \cdots & I_n[e^{i(n-1)x}] \\ I_n[e^{-ix}] & I_n[1] & \cdots & I_n[e^{i(n-2)x}] \\ \vdots & \vdots & \ddots & \vdots \\ I_n[e^{-i(n-1)x}] & I_n[e^{-i(n-2)x}] & \cdots & I_n[1] \end{bmatrix} = I$$

■

Proposition (trigonometric interpolant) Suppose n is odd with $n = 2m + 1$, then

$$p(x) = \sum_{k=-m}^m \tilde{c}_k e^{ikx}$$

interpolates f at x_j , $j = 0, \dots, n - 1$, i.e., $p(x_j) = f(x_j)$ for $j = 0, \dots, n - 1$.

Proof We want to show that $\mathbf{p} = \mathbf{f}$, where

$$\mathbf{p} = \begin{pmatrix} p(x_0) \\ p(x_1) \\ \vdots \\ p(x_{n-2}) \\ p(x_{n-1}) \end{pmatrix} \quad \text{and} \quad \mathbf{f} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) \end{pmatrix}$$

We have that

$$\mathbf{p} = \begin{pmatrix} p(x_0) \\ p(x_1) \\ \vdots \\ p(x_{n-2}) \\ p(x_{n-1}) \end{pmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ e^{-imx_1} & e^{-i(m-1)x_1} & e^{-i(m-2)x_1} & \cdots & e^{imx_1} \\ e^{-imx_2} & e^{-i(m-1)x_2} & e^{-i(m-2)x_2} & \cdots & e^{imx_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e^{-imx_{n-1}} & e^{-i(m-1)x_{n-1}} & e^{-i(m-2)x_{n-1}} & \cdots & e^{imx_{n-1}} \end{bmatrix}}_V \begin{pmatrix} \tilde{c}_{-m} \\ \tilde{c}_{-m+1} \\ \vdots \\ \tilde{c}_{m-1} \\ \tilde{c}_m \end{pmatrix}$$

where V is known as the Vandermonde matrix.

Since $\tilde{c}_{-1} = \tilde{c}_{n-1}$, $\tilde{c}_{-2} = \tilde{c}_{n-2}$, \dots , $\tilde{c}_{-m} = \tilde{c}_{n-m} = \tilde{c}_{m+1}$

$$\begin{pmatrix} \tilde{c}_{-m} \\ \vdots \\ \tilde{c}_m \end{pmatrix} = \underbrace{\begin{pmatrix} & I_m \\ I_{m+1} & \end{pmatrix}}_P \begin{pmatrix} \tilde{c}_0 \\ \vdots \\ \tilde{c}_{n-1} \end{pmatrix} = P \tilde{\mathbf{c}} = \frac{P}{\sqrt{n}} Q_n \mathbf{f}$$

where P is a permutation matrix.

Let $\mathbf{x} = (x_0 \ \cdots \ x_{n-1})^\top$, then we can express the Vandermonde matrix as

$$V = (e^{-im\mathbf{x}} \ \cdots \ e^{-i\mathbf{x}} \ \mathbf{1} \ e^{i\mathbf{x}} \ \cdots \ e^{im\mathbf{x}})$$

and

$$\sqrt{n} Q_n^* = (\mathbf{1} \ e^{i\mathbf{x}} \ e^{2i\mathbf{x}} \ \cdots \ e^{(n-1)i\mathbf{x}})$$

Since $e^{-i\mathbf{x}} = e^{i(n-1)\mathbf{x}}$, $e^{-2i\mathbf{x}} = e^{i(n-2)\mathbf{x}}$, \dots , $e^{-m i\mathbf{x}} = e^{i(n-m)\mathbf{x}} = e^{i(m+1)\mathbf{x}}$, we have that

$$V = \sqrt{n} Q_n^* \begin{pmatrix} I_{m+1} \\ I_m \end{pmatrix} = \sqrt{n} Q_n^* P^\top$$

Putting everything together, we have

$$\mathbf{p} = \sqrt{n} Q_n^* P^T P \frac{Q_n}{\sqrt{n}} \mathbf{f} = \mathbf{f}$$

■

Let's check the formula $V = \sqrt{n} Q_n^* P^\top$ in code

```
using Plots, LinearAlgebra, SparseArrays
```

```
n = 11
```

```
m = (n-1)÷2
```

```
x = range(0, 2π; length=n+1)[1:end-1] # x_0, ..., x_{n-1}, dropping
```

```
x_n == 2π
```

```
Q_n = [exp(-im*(k-1)*x[j]) for k = 1:n, j=1:n]/sqrt(n)
```

```
Q_n*Q_n' ≈ I
```

```
true
```

```
P = sparse(1:n, [m+2:n; 1:m+1], fill(1,n))
```

11×11 SparseMatrixCSC{Int64, Int64} with 11 stored entries:

```
. . . . . . 1 . . . .
. . . . . . . 1 . . . .
```

```

. . . . . . . . . 1 .
. . . . . . . . . 1 .
. . . . . . . . . 1
1 . . . . . . . . .
. 1 . . . . . . . .
. . 1 . . . . . . .
. . . 1 . . . . . .
. . . . 1 . . . . .
. . . . . 1 . . . .

```

`V = [exp(im*k*x[j]) for j = 1:n, k=-m:m]`

`V ≈ sqrt(n)*Q_n'*transpose(P)`

`true`

Here's an example illustrating that $p(x)$ interpolates f at x_j for $j = 0, \dots, n - 1$.

write a function that evaluates the interpolant

`function triginterp(c, x)`

ct = [c_0, ..., c_{n-1}] (the approximate Fourier coefficients)

`m = (n-1)÷2 # use coefficients between -m:m`

coefficients are complex so we need complex arithmetic

`for k = 0:m`

`p += c[k+1] * exp(im*k*x)`

```

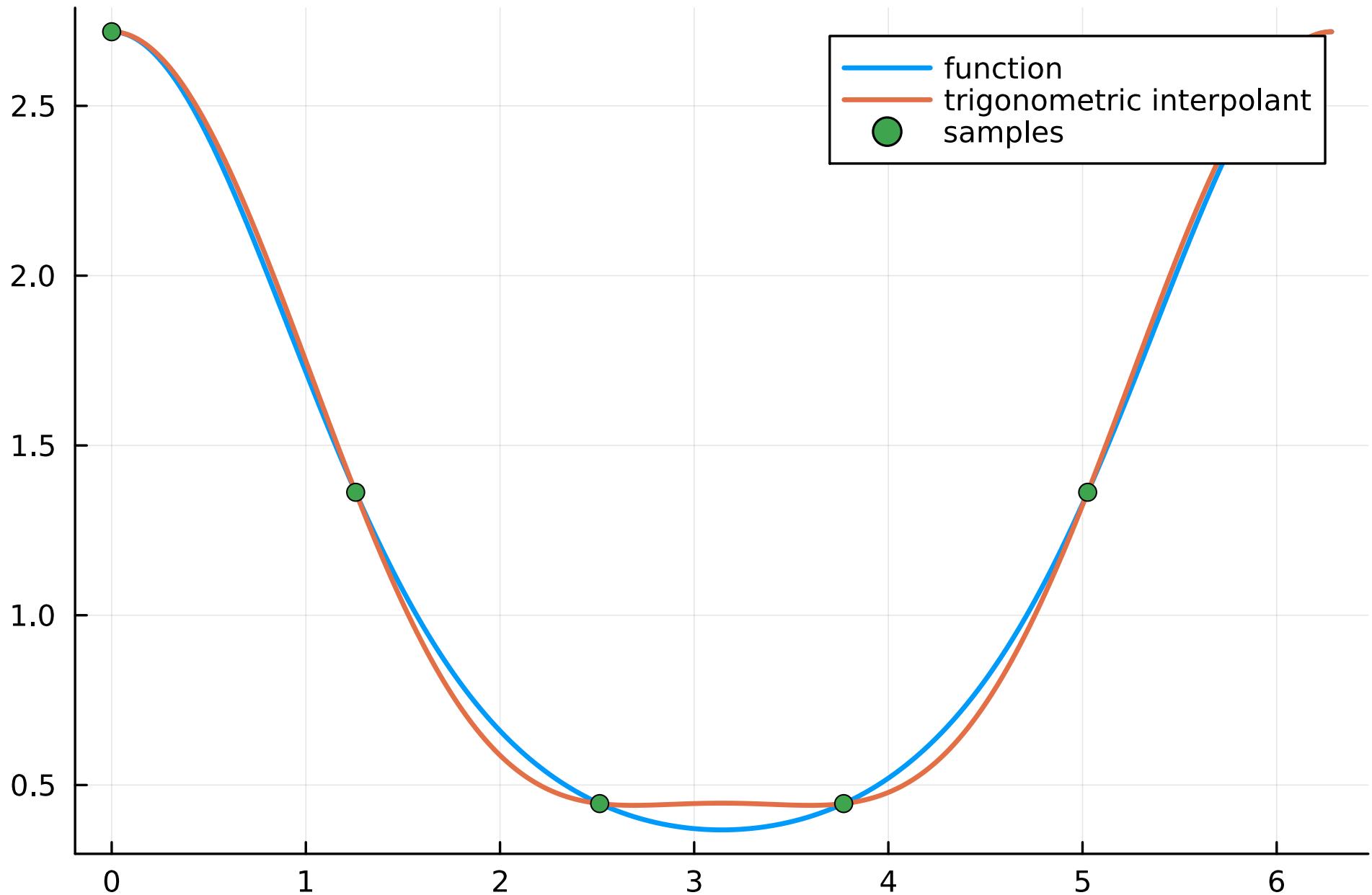
end
for k = -m:-1
    p += c[end+k+1] * exp(im*k*x)
end
p
end;

f = x -> exp(cos(x))
n = 5
# compute the approximate Fourier coefficients
x = range(0,2π; length=n+1)[1:end-1] # interpolation nodes x_0,
...,x_{n-1}
Q_n = 1/sqrt(n) * [exp(-im*(k-1)*x[j]) for k = 1:n, j=1:n]
c = 1/sqrt(n) * Q_n * f.(x)

p = x -> triginterp(c, x)

xx = range(0, 2π; length=1000) # plotting grid
plot(xx, f.(xx); lw=2, label="function", legend=:topright)
plot!(xx, real.(p.(xx)); lw=2, label="trigonometric interpolant")
scatter!(x, f.(x); label="samples")

```



Applying Q_n or its conjugate transpose Q_n^* to a vector naively takes $\mathcal{O}(n^2)$ operations. Both can be reduced to $\mathcal{O}(n \log n)$ operations using the celebrated *Fast Fourier Transform* (FFT), which is one of the [Top 10 Algorithms of the 20th Century](#).

```

# Here's an example of using the FFT to map function values to
coefficients and vice versa
using FFTW
f = x -> exp(cos(x-0.1))
n = 31
m = (n-1)÷2
# evenly spaced points from 0:2π, dropping last node
x = range(0, 2π; length=n+1)[1:end-1]
# fft returns the approximate Fourier coefficients n*[c_0, ...,
c_{n-1}]
c = fft(f.(x))/n
# check that ifft maps c to [f(x_0), ..., f(x_{n-1})]/n
f.(x) ≈ n*ifft(c)

true

```

Remark The FFTW.jl package wraps the FFTW (Fastest Fourier Transform in the West) library, which is a highly optimised implementation of the FFT. (As an aside, the creator of FFTW [Steven Johnson](#) is now a Julia contributor and user.)

Recall that

$$\tilde{\mathbf{c}} = \frac{1}{\sqrt{n}} Q_n \mathbf{f} \quad \text{and} \quad \mathbf{f} = \sqrt{n} Q_n^* \tilde{\mathbf{c}}.$$

When we use the FFT to map function values on the grid \mathbf{f} to approximate Fourier coeffi-

cients, $\tilde{\mathbf{c}}$, we shall express this as

$$\tilde{\mathbf{c}} = \mathcal{F}\{\mathbf{f}\}$$

and we denote the map from coefficients to function values via the (inverse) FFT by

$$\mathbf{f} = \mathcal{F}^{-1}\{\tilde{\mathbf{c}}\}$$

The magic of the FFT is because its complexity is $\mathcal{O}(n \log n)$ (almost linear) we can scale it to very high orders. Here we plot the Fourier coefficients for a function that requires around 100k coefficients to resolve:

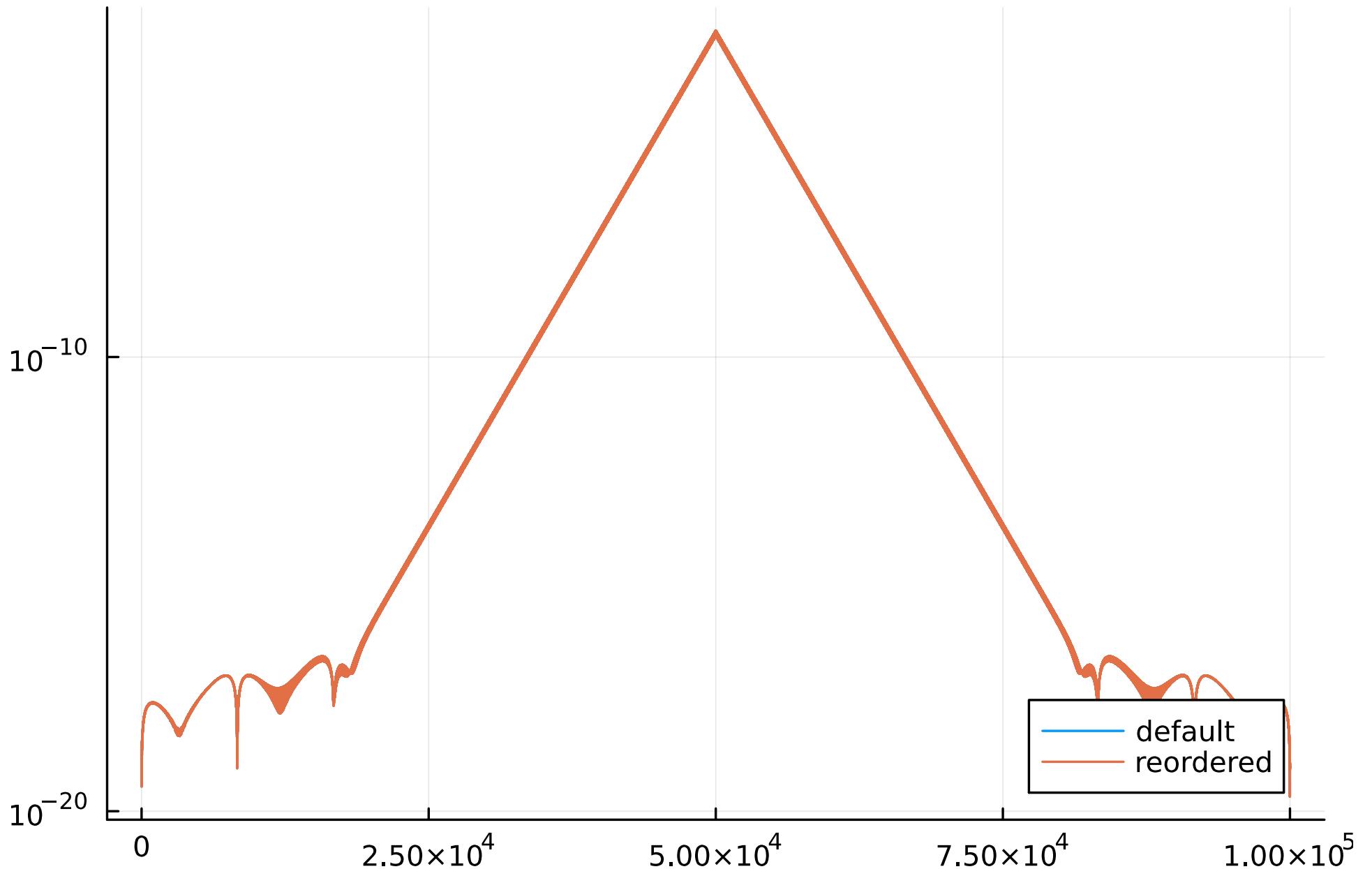
```
f = x -> exp(sin(x))/(1+1e6cos(x)^2)
n = 100_001
m = (n-1)÷2
# evenly spaced points from 0:2π, dropping last node
x = range(0, 2π; length=n+1)[1:end-1]

# fft returns discrete Fourier coefficients n*[c_0, ..., c_n--1]
c = @time fft(f.(x))/n

# Reorder using [c__m, ..., c__1] == [c_n__m, ..., c_n__1]
c = [c[m+2:end]; c[1:m+1]]

plot(abs.(c);yscale=:log10, legend=:bottomright, label="default")
```

```
plot!(abs.(c);yscale=:log10, label="reordered")  
0.093584 seconds (141.28 k allocations: 12.761 MiB, 71.21%  
compilation ti  
me)
```



The FFT allows us to efficiently transform between "function value space"/"physical space" and "coefficient space"/"frequency space". This is essential in the context of numerical methods for differential equations because we'll see that some operations (e.g. differenti-

ation) can be performed much more efficiently in coefficient space than in function value space, while computing nonlinear terms may be performed more efficiently in function value space.

1.2 Differentiation in function value space and in coefficient space

We have shown above that for n odd with $n = 2m + 1$,

$$p(x) = \sum_{k=-m}^m \tilde{c}_k e^{ikx}, \quad (1)$$

with

$$\tilde{c}_k = \frac{1}{n} \sum_{j=0}^{n-1} f(x_j) e^{ikx_j} \quad (2)$$

interpolates f at x_0, \dots, x_{n-1} . This is a representation of the interpolant p in terms of coefficients \tilde{c}_k and basis functions e^{ikx} . Now we want to construct a different representation of p in terms of function values to illustrate the difference between differentiation in function value space and coefficient space.

Similar to the formula for the Lagrange interpolating polynomial, we want a formula for a trigonometric interpolant of the form

$$p(x) = \sum_{j=0}^{n-1} \ell_j(x) f(x_j).$$

Notice that now p is represented as an expansion in function values $f(x_j)$ and basis functions $\ell_j(x)$. We require the basis functions to have property

$$\ell_j(x_i) = \begin{cases} 1 & \text{if } i = j, i = j \pm n, i = j \pm 2n, \dots \\ 0 & \text{otherwise} \end{cases}$$

so that $p(x)$ is periodic on the grid points ($p(x_{j+pn}) = p(x_j)$, for $p \in \mathbb{Z}$) and is an interpolant ($p(x_j) = f(x_j)$, for $j = 0, \dots, n - 1$).

The idea is first to find $\ell_0(x)$ and then to set $\ell_j(x) = \ell_0(x - x_j)$ because then $\ell_j(x_i) = \ell_0(x_i - x_j)$ and therefore $\ell_j(x_i) = 1$ if $i = j \bmod n$, as required.

The function $\ell_0(x)$ can be found by using (1) and (2) to construct the interpolant of a periodic function f with the property that $f(x_0) = 1$ and $f(x_j) = 0$, for $j = 1, \dots, n - 1$. For this function (a periodic, discrete delta function), the approximate Fourier coefficients are

$$\tilde{c}_k = \frac{1}{n} = \frac{h}{2\pi}, \quad \forall k,$$

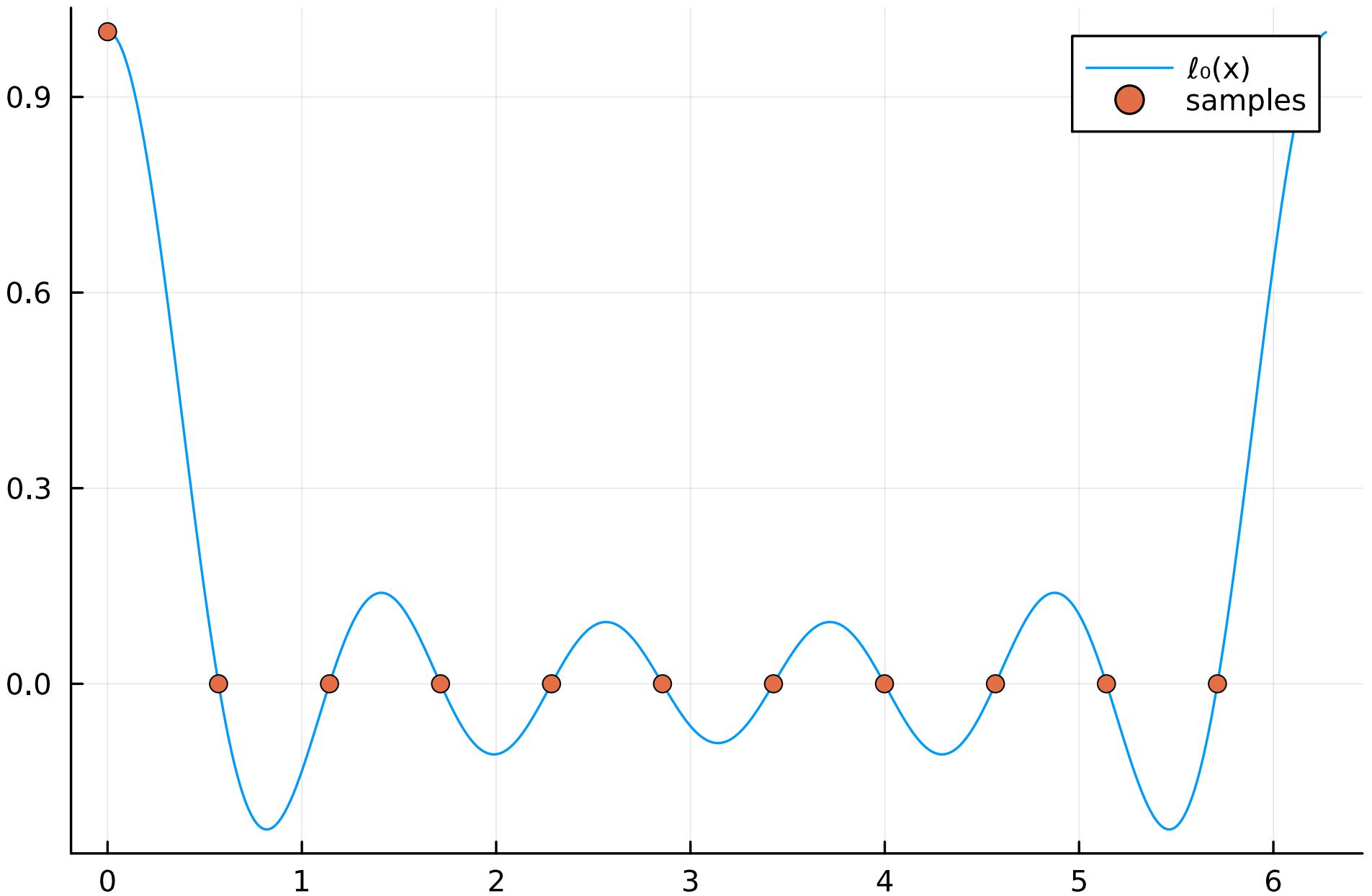
therefore

$$\ell_0(x) = p(x) = \sum_{k=-m}^m \tilde{c}_k e^{ikx} = \frac{h}{2\pi} \sum_{k=-m}^m e^{ikx} = \frac{1}{n} \frac{\sin((m+1/2)x)}{\sin(x/2)}$$

(deriving this formula is left as an exercise)

Let's check that $\ell_0(x)$ has the required properties:

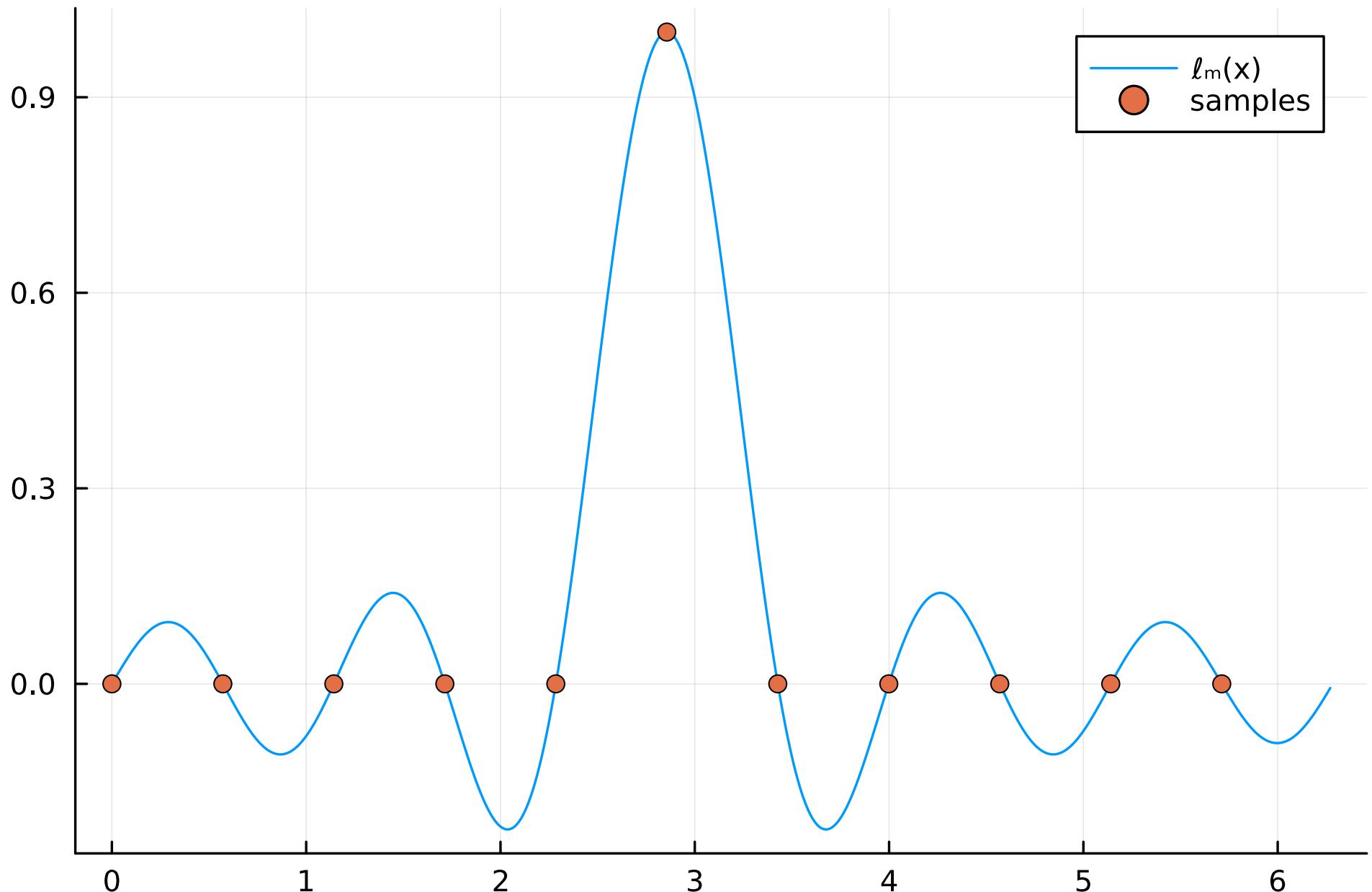
```
n = 11
m = (n-1)÷2
h = 2π/n
ℓ_0 = x -> x == 0 ? 1 : h/(2π)*sin((m+0.5)*x)/sin(x/2)
x = range(0,2π;length=n+1)[1:end-1] #x_0, ..., x_{n-1}
xx = range(0,2π;length=501)[1:end-1] #plotting grid
plot(xx,ℓ_0.(xx);label="ℓ_0(x)")
scatter!(x,ℓ_0.(x);label="samples")
```



Here is a translated version of $\ell_0(x)$, namely $\ell_0(x - x_m) = \ell_m(x)$

```
plot(xx, l_0.(xx .- x[m+1]); label="l_m(x)")
```

```
scatter!(x,ℓ_0.(x .- x[m+1]);label="samples")
```



Hence, another representation of the interpolant is

$$p(x) = \sum_{j=0}^{n-1} \ell_j(x) f(x_j) = \sum_{j=0}^{n-1} \ell_0(x - x_j) f(x_j), \quad \ell_0(x) = \frac{1}{n} \frac{\sin((m+1/2)x)}{\sin(x/2)}$$

from which we obtain approximations to the derivative at the grid points x_j , $j = 0, \dots, n-1$:

$$f'(x_i) \approx p'(x_i) = \sum_{j=0}^{n-1} \ell'_j(x_i) f(x_j) = \sum_{j=0}^{n-1} \ell'_0(x_i - x_j) f(x_j), \quad i = 0, \dots, n-1.$$

It is left as an exercise to show that

$$\ell'_0(x_i - x_j) = \begin{cases} 0 & \text{if } i = j \\ \frac{(-1)^{i-j}}{2} \csc\left(\frac{(i-j)h}{2}\right) & \text{if } i \neq j \end{cases}.$$

This implies that the differentiation matrix is Toeplitz and the entries on the $(i - j)$ -th diagonal is $\frac{(-1)^{i-j}}{2} \csc\left(\frac{(i-j)h}{2}\right)$, i.e.,

$$\begin{pmatrix} f'(x_0) \\ \vdots \\ f'(x_{n-1}) \end{pmatrix} \approx \begin{pmatrix} p'(x_0) \\ \vdots \\ p'(x_{n-1}) \end{pmatrix} = \begin{pmatrix} \vdots & & \\ \cdots & \frac{1}{2} \csc \frac{3h}{2} & \vdots \\ \cdots & -\frac{1}{2} \csc \frac{2h}{2} & \cdots \\ \cdots & \frac{1}{2} \csc \frac{1h}{2} & \cdots \\ 0 & & \ddots \\ -\frac{1}{2} \csc \frac{1h}{2} & & \cdots \\ \frac{1}{2} \csc \frac{2h}{2} & & \cdots \\ -\frac{1}{2} \csc \frac{3h}{2} & & \cdots \\ \vdots & & \end{pmatrix} \begin{pmatrix} f_0 \\ \vdots \\ f_{n-1} \end{pmatrix}.$$

Alternatively, we can approximate f' by differentiating the "coefficient space" interpolant (1):

$$f'(x) \approx p'(x) = \sum_{k=-m}^m ik \tilde{c}_k e^{ikx}.$$

Similarly, we approximate the ν -th order derivative with $\nu \in \mathbb{Z}_+$ as

$$f^{(\nu)}(x) \approx p^{(\nu)}(x) = \sum_{k=-m}^m (ik)^\nu \tilde{c}_k e^{ikx}.$$

Let's denote the Fourier coefficients of $f^{(\nu)}(x)$ by $c_k^{(\nu)}$, then

$$f^{(\nu)}(x) = \sum_{k=-\infty}^{\infty} c_k^{(\nu)} e^{ikx},$$

where $c_k^{(\nu)} = (ik)^\nu c_k$. Therefore we approximate $c_k^{(\nu)}$ using the Fourier coefficients of $p^{(\nu)}(x)$, i.e.,

$$\begin{pmatrix} c_{-m}^{(\nu)} \\ \vdots \\ c_m^{(\nu)} \end{pmatrix} \approx \begin{pmatrix} (-im)^\nu & & \\ & \ddots & \\ & & (im)^\nu \end{pmatrix} \begin{pmatrix} \tilde{c}_{-m} \\ \vdots \\ \tilde{c}_m \end{pmatrix}.$$

Note that in "coefficient space", the differentiation matrix is diagonal whereas it is dense in "function value" space.

Starting from the function values on the grid, $f(x_j)$, $j = 0, \dots, n - 1$, we can approximate $f^{(\nu)}(x_j) \approx p^{(\nu)}(x_j)$, $j = 0, \dots, n - 1$ using the FFT and inverse FFT as follows:

1. compute $\tilde{c}_{-m}, \dots, \tilde{c}_m$ using the FFT (function values to coefficients)
2. compute $(-im)^\nu \tilde{c}_{-m}, (-i(m - 1))^\nu \tilde{c}_{-(m-1)}, \dots, (im)^\nu \tilde{c}_m$ (differentiate in coefficient space)
3. compute $p^{(\nu)}(x_j)$ using the (inverse) FFT (coefficients to function values)

Note that this procedure has $\mathcal{O}(n \log n)$ complexity where naive multiplication by a dense differentiation matrix has quadratic complexity ($\mathcal{O}(n^2)$).

Here is an example:

```

f = x -> 1/(2 + cos(x))
Df = x -> sin(x)/(2 + cos(x))^2 # derivative
n = 101
m = (n-1)÷2
x = range(0,2π;length=n+1)[1:end-1] # grid points x_0, ..., x_{n-1}
# map function values to coefficients via the fft
c = fft(f.(x))/n; # [c_0, ..., c_{n-1}]
# re-order coefficients
ch = fftshift(c); # reorder the coefficients to [c_{-m}, ..., c_m]
# Fourier coefficients of the derivative
dch = (im*(-m:m)).*ch;
# re-order
dc = ifftshift(dch); # undoes what fftshift did
# map coefficients to function values via the ifft
df = n*ifft(dc);
norm(df - Df.(x),Inf)

```

1.7790211156964822e-14

```

# more briefly
df = ifft(ifftshift(im*(-m:m)).*fft(f.(x)))
norm(df - Df.(x),Inf)

```

1.7726178619566572e-14

Mathematically, we express approximate ν -th order differentiation of a function $f(x)$ on the equispaced grid $x_j = jh = 2\pi j/n$, $j = 0, \dots, n - 1$ with $n = 2m + 1$ as follows

$$\mathcal{F}^{-1} \{ (i(-m:m))^\nu \cdot \mathcal{F}\{f\} \}$$

Here we approximate $f'(x_j)$, $j = 0, \dots, n - 1$ using (i) differentiation matrices and (ii) the FFT.

```

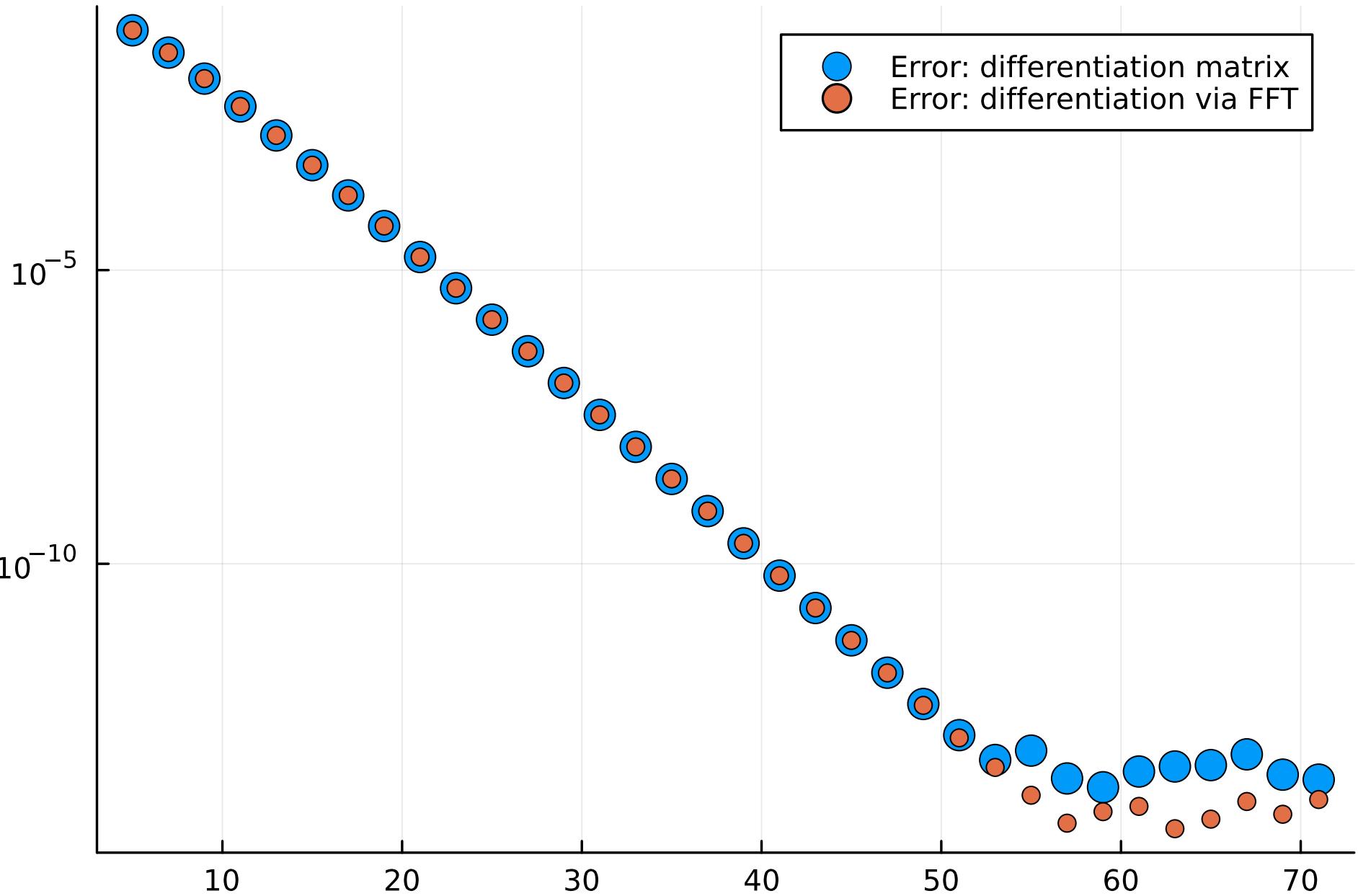
using ToeplitzMatrices
nv = 5:2:71
f = x -> 1/(2 + cos(x))
Df = x -> sin(x)/(2 + cos(x))^2 # derivative

# Differentiation matrices
errs =
[( h = 2π/n;
  column = [0; 0.5*(-1).^(1:n-1).*csc.((1:n-1)*h/2)];
  D_n = Toeplitz(column,[column[1]; column[n:-1:2]]); #
  Differentiation matrix:
  x = range(0,2π;length=n+1)[1:end-1]; # n equally spaced points
  on [0, 2π]
  norm(D_n*f.(x) - Df.(x),Inf) ) for n = nv] # maximum error at
  the set of N points

```

```
# Differentiation in coefficient space via the FFT
errsfft =
[( m = (n-1)÷2;
  x = range(0,2π;length=n+1)[1:end-1];
  df = ifft(ifftshift(im*(-m:m)).*fft(f.(x)));
  norm(df - Df.(x),Inf) ) for n = nv]

# plot the error on a semi-log (log-linear) scale
scatter(nv,errs;yscale=:log10,ms=7,label="Error: differentiation
matrix")
scatter!(nv,errsfft;yscale=:log10,label="Error: differentiation via
FFT")
```



1.3 PDE example

Now we have the tools to implement a numerical method for the *advection equation*

$$u_t + u_x = 0, \quad u = u(x, t).$$

We want to approximate the solution to this equation with the initial data

$$u(x, 0) = f(x) = e^{-100(x-1)^2}.$$

We can easily verify that the exact solution to this Cauchy problem is

$$u(x, t) = f(x - t).$$

This solution can be derived using the method of characteristics.

As it stands, this problem is posed on the entire real line. However, since the initial data decay very rapidly to zero away from $x = 1$, it is approximately periodic on $[0, 2\pi]$ and therefore we approximate $u(x, t)$ in the x -direction using Fourier methods.

We let the equispaced spatial grid be

$$x_j = jh = j \frac{2\pi}{n_x}, \quad j = 0, \dots, n_x - 1, \quad n_x = 2m + 1$$

and we let $t \in [0, T]$ and set

$$t_i = i\tau, \quad \tau = \frac{T}{n_t}, \quad i = 0, \dots, n_t.$$

We let u_j^i denote the approximation to the exact solution u at $x = x_j$ and $t = t_i$, i.e.,

$$u_j^i \approx u(x_j, t_i).$$

(the index i should not be confused with the imaginary unit $i = \sqrt{-1}$). We set $u_j^0 = u(x_j, 0) = f(x_j)$ or, in vector notation, $\mathbf{u}^0 = \mathbf{f}$.

We approximate the time-derivative with forward differences:

$$u_t(x_j, t_i) \approx \frac{u(x_j, t_{i+1}) - u(x_j, t_i)}{\tau} \approx \frac{u_j^{i+1} - u_j^i}{\tau}, \quad j = 0, \dots, n_x - 1$$

or, in vector notation,

$$u_t(\mathbf{x}, t_i) \approx \frac{\mathbf{u}^{i+1} - \mathbf{u}^i}{\tau}.$$

We approximate the spatial derivative using the FFT!:

$$u_x(\mathbf{x}, t_i) \approx \mathcal{F}^{-1} \left\{ i(-m:m) \cdot \mathcal{F}\{\mathbf{u}^i\} \right\}$$

Therefore we approximate the solution to the PDE as follows:

$$\mathbf{u}^{i+1} = \mathbf{u}^i - \tau \mathcal{F}^{-1} \left\{ i(-m:m) \cdot \mathcal{F}\{\mathbf{u}^i\} \right\}, \quad i = 0, \dots, n_t - 1, \quad \mathbf{u}^0 = \mathbf{f}.$$

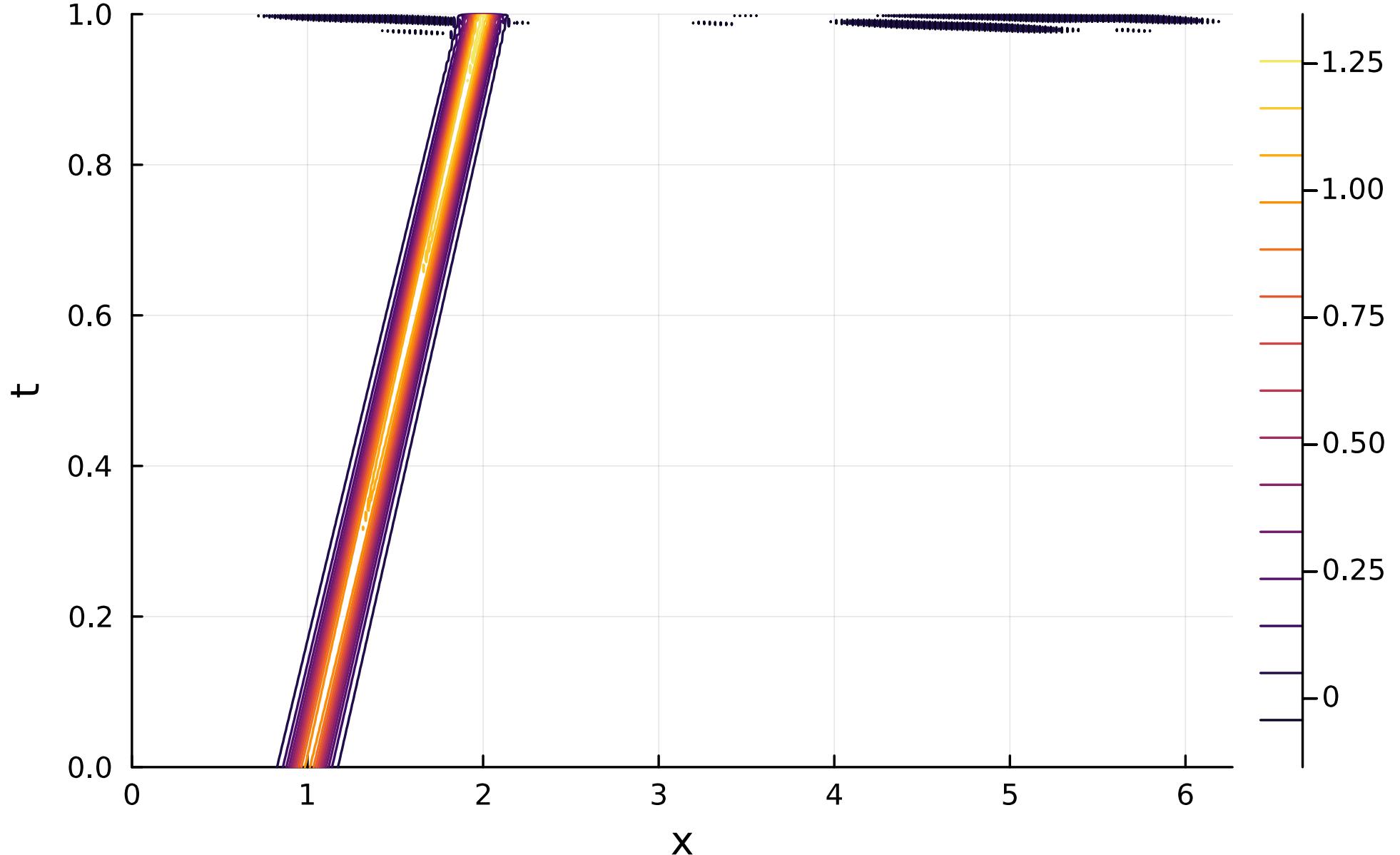
```

f = x -> exp(-100*(x-1)^2)
n_x = 401
m = (n_x - 1)÷2
x = range(0,2π;length=n_x+1)[1:end-1] # the equispaced grid in the
#x-direction
n_t = 500
T = 1
τ = T/n_t
u = zeros(n_t + 1,n_x)
u[1,:] = f.(x) # initial data
for n = 1:n_t-1
    u[n+1,:] = real.(u[n,:] -
τ*ifft(ifftshift(im*(-m:m)).*fft(u[n,:])))
end

t = range(0,T;length=n_t+1)
contour(x,t,u;xlabel="x",ylabel="t",title="Approximate solution to
the advection equation")

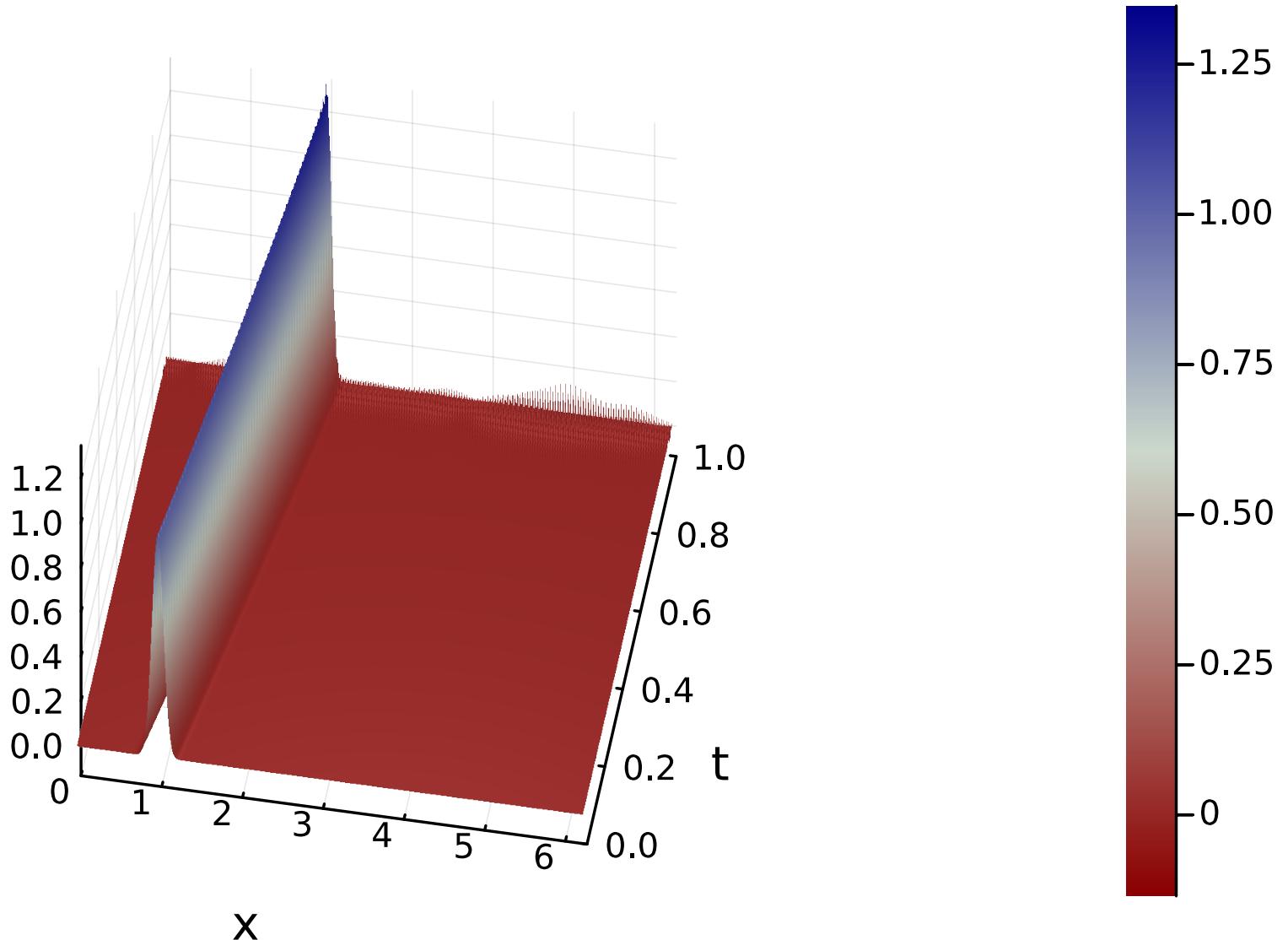
```

Approximate solution to the advection equation



```
surface(x,t,u;seriescolor=:redsblues, camera=(10,50),
xlabel="x",ylabel="t",title="Approximate solution to the advection
equation")
```

Approximate solution to the advection equation



Note the spurious oscillations in the numerical solution. We shall analyse why this instability occurs later in this module. We shall also see that if we approximate the time derivative with a central difference, then we can take larger time steps while retaining stability. That

is, if we use

$$u_t(x_j, t_i) \approx \frac{u(x_j, t_{i+1}) - u(x_j, t_{i-1})}{2\tau} \approx \frac{u_j^{i+1} - u_j^{i-1}}{2\tau}, \quad j = 0, \dots, n_x - 1$$

and again approximate the spatial derivative via FFTs, then we obtain another numerical method for the advection equation:

$$\mathbf{u}^{i+1} = \mathbf{u}^{i-1} - 2\tau \mathcal{F}^{-1} \left\{ i(-m:m) \cdot \mathcal{F}\{\mathbf{u}^i\} \right\}, \quad i = 0, \dots, n_t - 1, \quad \mathbf{u}^0 = \mathbf{f}.$$

This is known as a leap frog method.

We have \mathbf{u}^0 , however we also need \mathbf{u}^1 to implement the leap frog method. There are various one-step methods one could use to obtain \mathbf{u}^1 from \mathbf{u}^0 (e.g., Runge-Kutta methods, which we might get to later) but here we use 10 steps of the forward difference method above with step size $\tau/10$:

```
n_t = 500
T = 2.5
tau = T/n_t
u = zeros(n_t + 1,n_x)
u[1,:] = f.(x)
```

```
# First take ten steps using the forward difference formula
ut = zeros(11,n_x)
u[1,:] = ut[1,:] = f.(x) # initial data
```

```

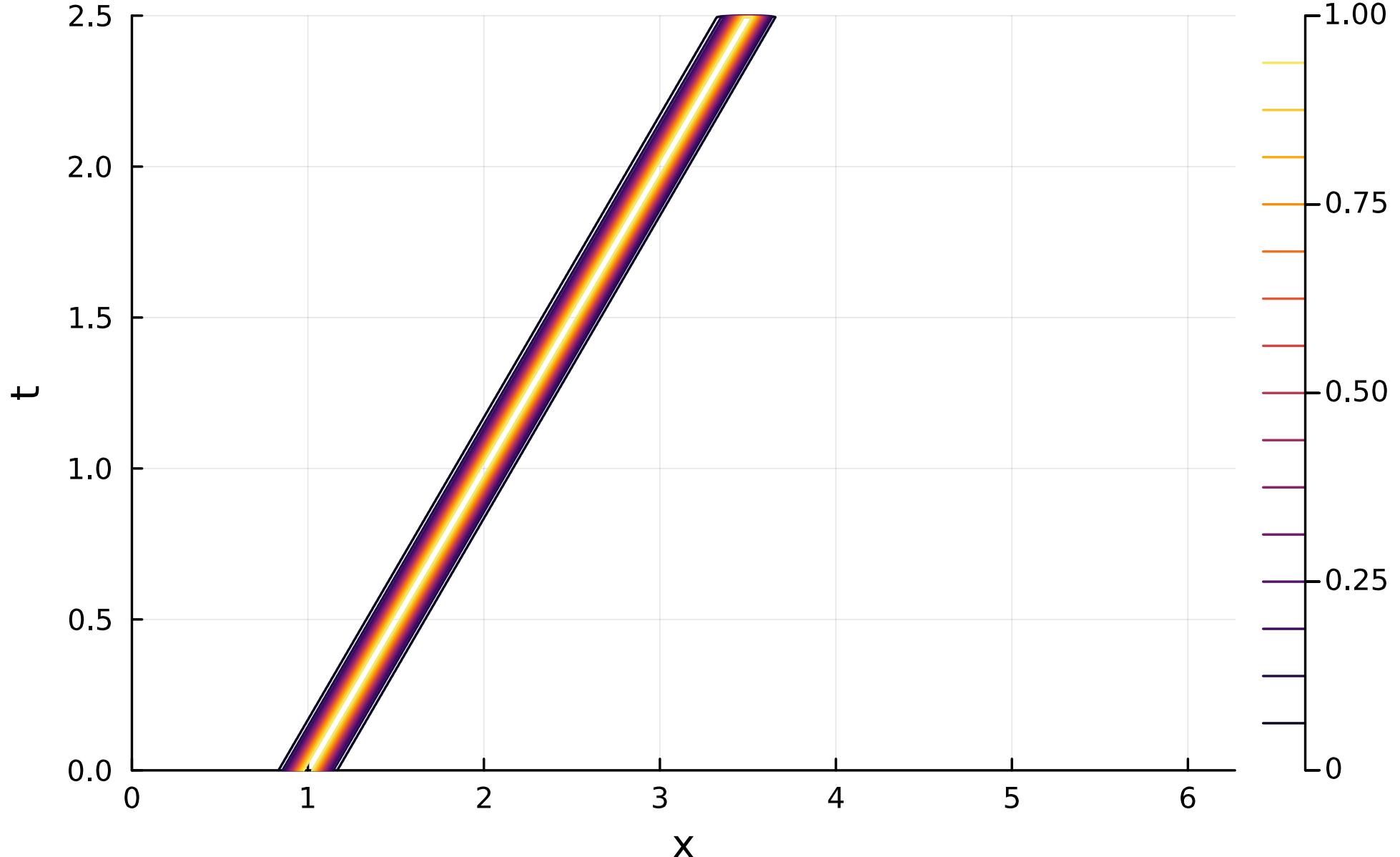
for n = 1:10
    ut[n+1,:] = real.(ut[n,:] -
 $\tau/10 * \text{ifft}(\text{ifftshift}(im*(-m:m)) .* \text{fft}(ut[n,:]))$ )
end

# Now use the leap frog method
u[2,:] = ut[11,:]
for n = 2:n_t-1
    u[n+1,:] = real.(u[n-1,:] -
 $2\tau * \text{ifft}(\text{ifftshift}(im*(-m:m)) .* \text{fft}(u[n,:]))$ )
end

t = range(0,T;length=n_t+1)
contour(x,t,u;xlabel="x",ylabel="t",title="Leapfrog solution to the
advection equation")

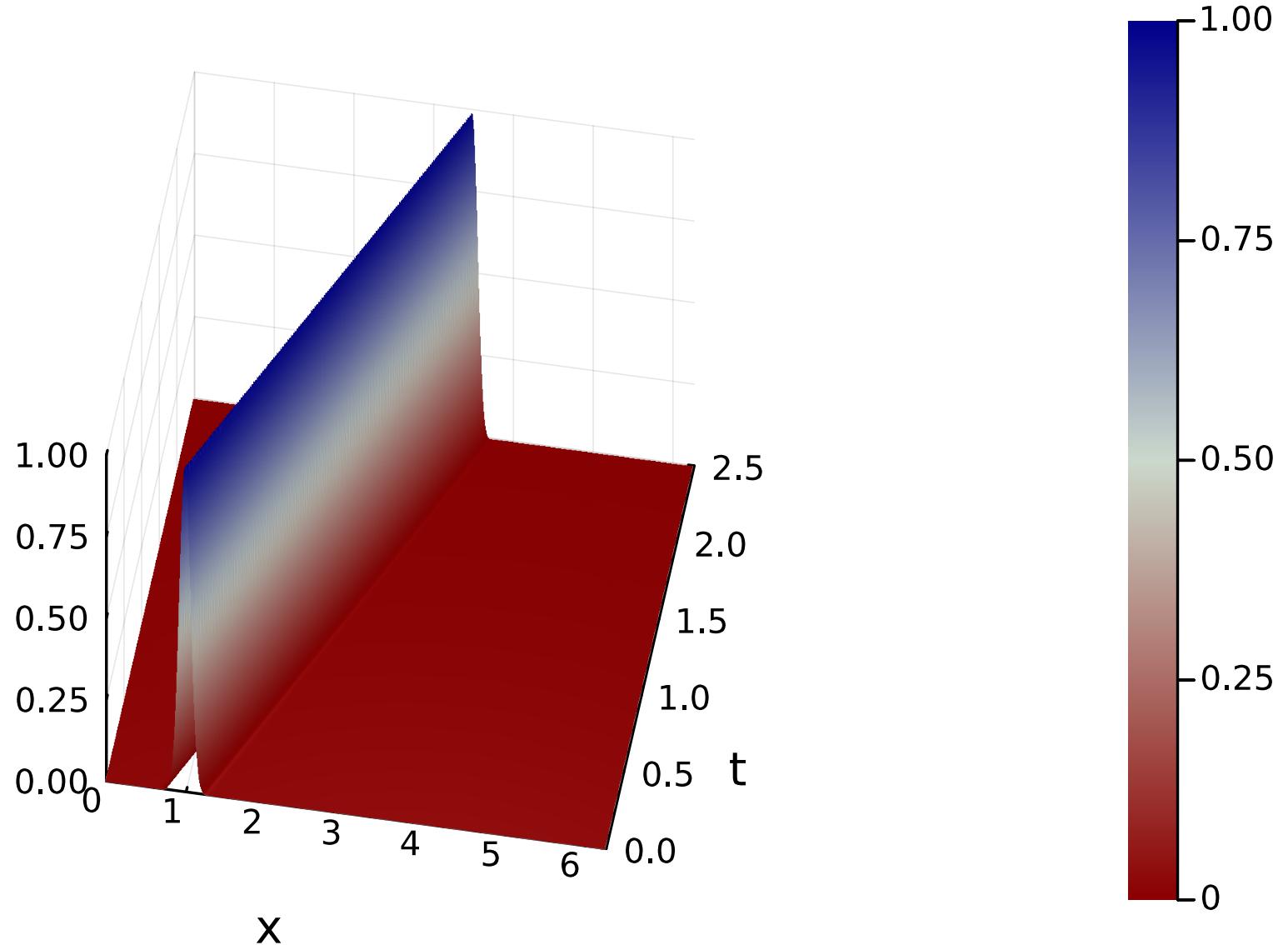
```

Leapfrog solution to the advection equation



```
surface(x,t,u;seriescolor=:redsblues, camera=(10,50),
xlabel="x",ylabel="t",title="Leapfrog solution to the advection
equation")
```

Leapfrog solution to the advection equation



Let's consider now the advection equation with a variable coefficient:

$$u_t + c(x)u_x = 0$$

with

$$c(x) = \frac{1}{5} + \sin^2(x - 1)$$

and the same initial data as before, $u(x, 0) = f(x) = e^{-100(x-1)^2}$. We can easily verify that the exact solution to this Cauchy problem (which can be derived using the method of characteristics) is

$$u(x, t) = f(t - C(x)),$$

where $C(x)$ is an antiderivative of $1/c(x)$, i.e., $C'(x) = 1/c(x)$. For example,

$$C(x) = -\frac{5}{\sqrt{6}} \arctan \left(\sqrt{6} \tan(1 - x) \right).$$

If we again use central differences to approximate the time derivative and FFTs to approximate the spatial derivative, then the leapfrog method is

$$\mathbf{u}^{i+1} = \mathbf{u}^{i-1} - 2\tau c(\mathbf{x}) \cdot \mathcal{F}^{-1} \left\{ i(-m:m) \cdot \mathcal{F}\{\mathbf{u}^i\} \right\}, \quad i = 0, \dots, n_t - 1, \quad \mathbf{u}^0 = \mathbf{f}.$$

As before, we'll use 10 steps of the forward difference method with a step size of $\tau/10$ to compute \mathbf{u}^1 to initialise the leapfrog method:

```

c = x -> 0.2 + sin(x - 1)^2
n_t = 2000
T = 8
τ = T/n_t
u = zeros(n_t + 1,n_x)
u[1,:] = f.(x)

```

First take ten steps using the forward difference formula

```

ut = zeros(11,n_x)
u[1,:] = ut[1,:] = f.(x) # initial data
for n = 1:10
    ut[n+1,:] = real.(ut[n,:] -
τ/10*c.(x).*ifft(ifftshift(im*(-m:m)).*fft(ut[n,:])))
end

```

Now use the leap frog method

```

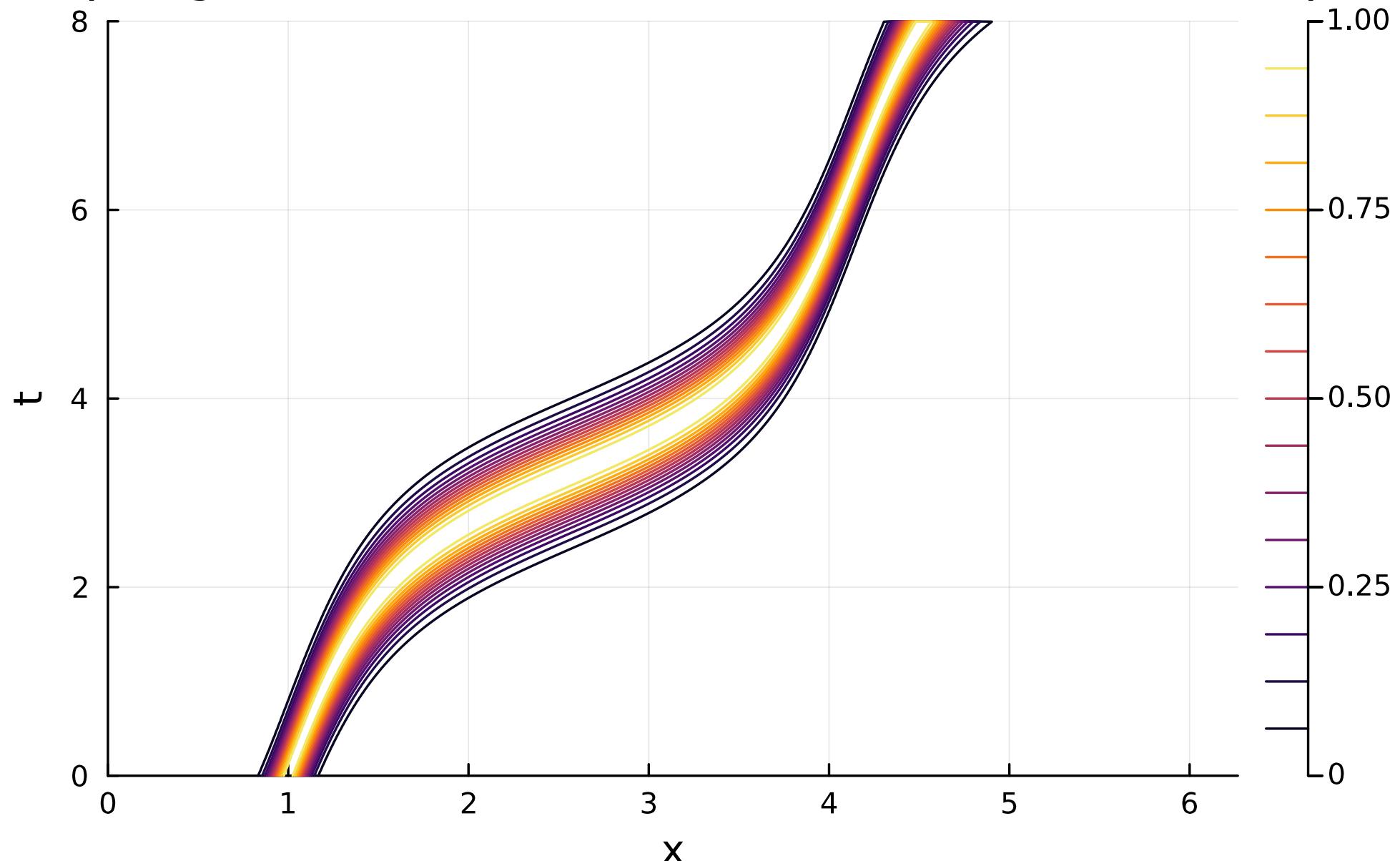
u[2,:] = ut[11,:]
for n = 2:n_t-1
    u[n+1,:] = real.(u[n-1,:] -
2τ*c.(x).*ifft(ifftshift(im*(-m:m)).*fft(u[n,:])))
end

```

```
t = range(0,T;length=n_t+1)
```

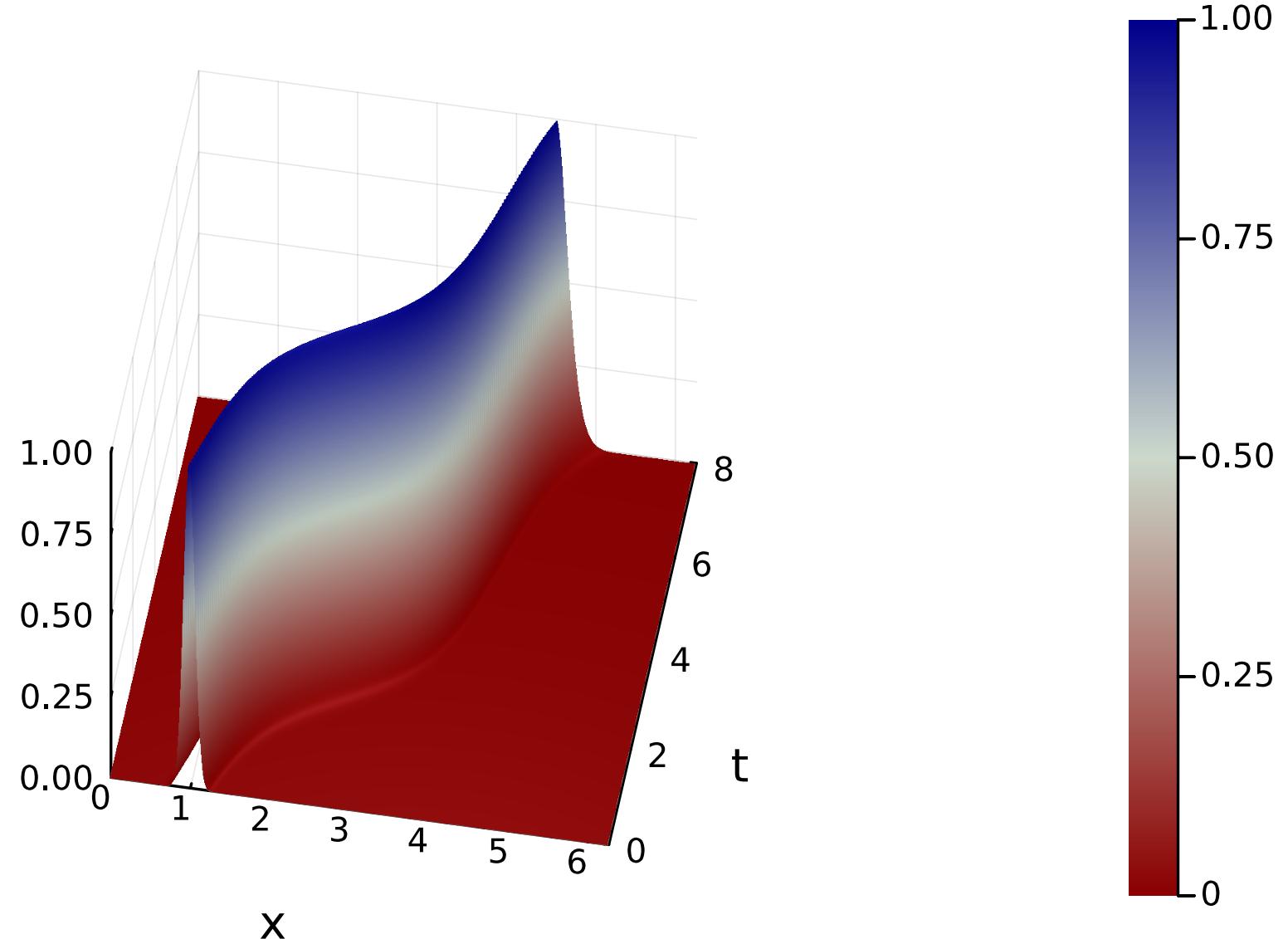
```
contour(x,t,u;xlabel="x",ylabel="t",title="Leapfrog sln to the  
variable coefficient advection eqn")
```

Leapfrog sln to the variable coefficient advection eqn



```
surface(x,t,u;seriescolor=:redsblues, camera=(10,50),  
xlabel="x",ylabel="t",title="Leapfrog sln to the variable  
coefficient advection eqn")
```

Leapfrog sln to the variable coefficient advection eqn



1.4 Smoothness of f and decay of Fourier coefficients

What happens if we approximate a function that is not periodic with the periodic interpolant

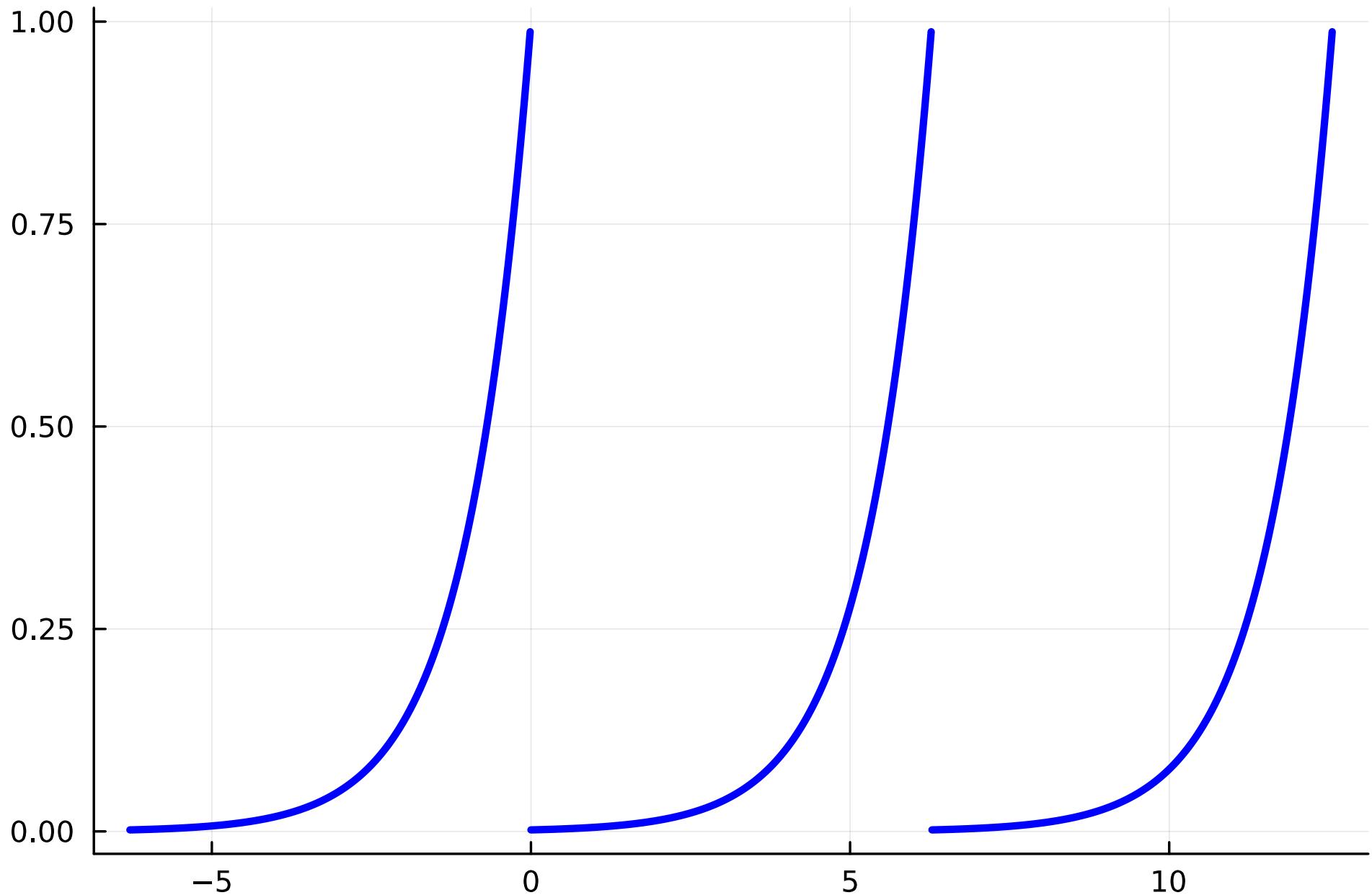
$$p(x) = \sum_{k=-m}^m \tilde{c}_k e^{ikx}$$

Example Consider the function $f(x) = e^{x-2\pi}$. This function is not 2π -periodic because $f(x + 2\pi) \neq f(x)$. However, for any function f , we can define its *periodic extension*

$$f(x + 2\pi p) = f(x), \quad x \in [0, 2\pi], \quad p \in \mathbb{Z},$$

which is a 2π -periodic function. Here is a plot of the periodic extension of f on $[-2\pi, 4\pi]$, however it extends infinitely far to the left and right.

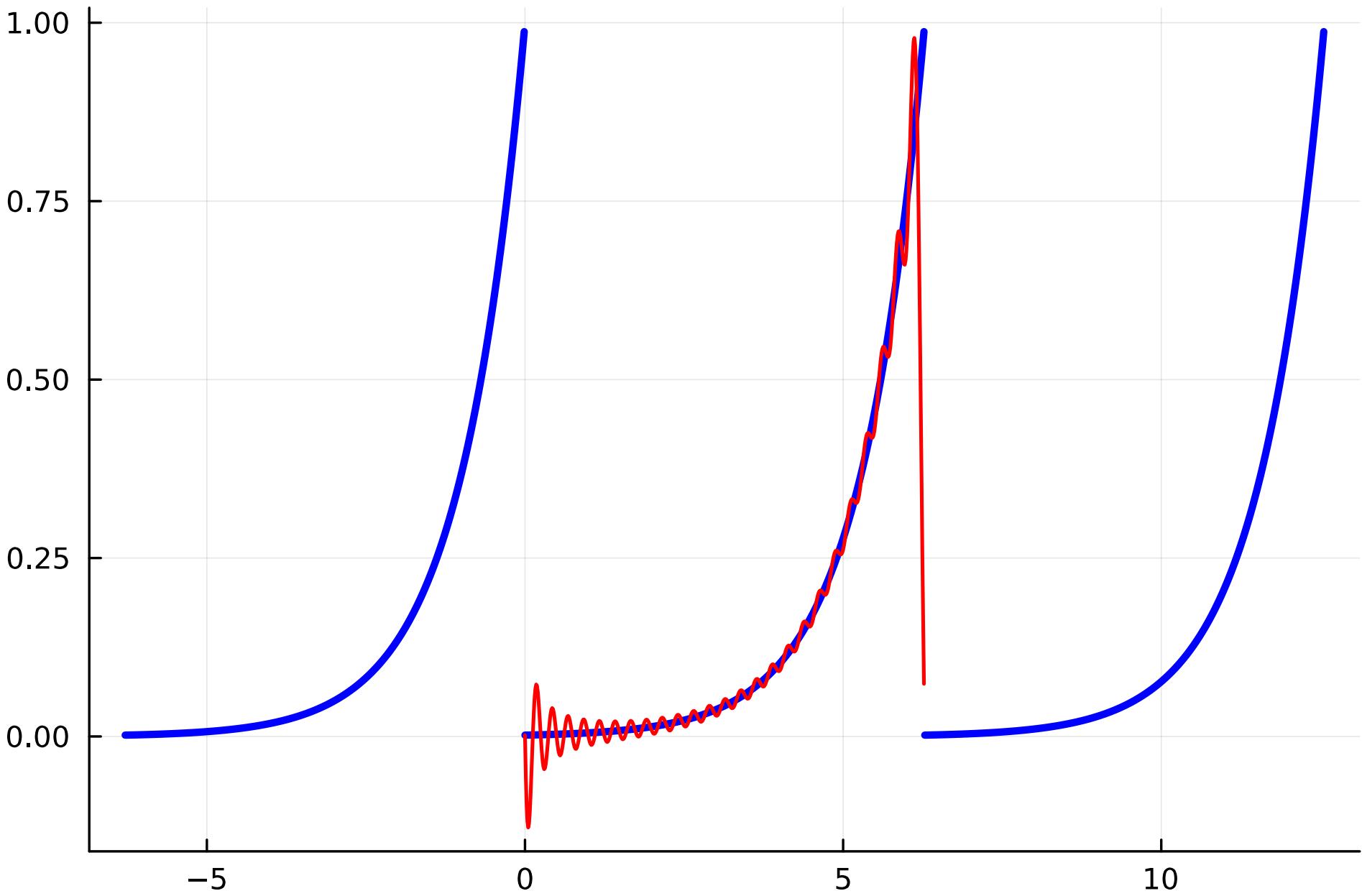
```
f_1 = x -> exp(x-2π)
xx = range(0,2π;length=501)[1:end-1] # plotting grid
plot(xx,f_1.(xx);lw=3,lc=:blue,legend=false)
plot!(xx .+ 2π,f_1.(xx);lw=3,lc=:blue,legend=false)
plot!(xx .- 2π,f_1.(xx);lw=3,lc=:blue,legend=false)
```



Notice that the periodic extension has jump discontinuities at $x = 2\pi p$, $p \in \mathbb{Z}$.
Here is the trigonometric interpolant of the periodic extension of f :

```
plot(xx,f_1.(xx);lw=3,lc=:blue,legend=false)
plot!(xx .+ 2π,f_1.(xx);lw=3,lc=:blue,legend=false)
plot!(xx .- 2π,f_1.(xx);lw=3,lc=:blue,legend=false)

# Trigonometric interpolant
n = 51
x = range(0,2π;length=n+1)[1:end-1]
c = fft(f_1.(x))/n
p = x -> triginterp(c, x)
plot!(xx,real.(p.(xx)),lw=1.5,lc=:red)
```



The periodic interpolant is clearly not a good approximation to the function. The oscillations at the endpoints is known as the *Gibbs phenomenon*. As $n \rightarrow \infty$, the amplitude of the oscillations decrease as $\mathcal{O}(n^{-1})$ (very slowly) and the interpolant at $x = 0$ and $x = 2\pi$

converges to the average of the function values at $x = 0$ and $x = 2\pi$:

$$\lim_{x \rightarrow 0, 2\pi} p(x) = \frac{1}{2} \left(\lim_{x \rightarrow 0^+} f(x) + \lim_{x \rightarrow 2\pi^-} f(x) \right), \quad n \rightarrow \infty.$$

The reason why p is not a good approximation to f is because it is not very smooth (it has jump discontinuities, so its 0-th derivative is discontinuous). Our aim in this section is to relate the smoothness of f to the accuracy of its approximation by the interpolant $p(x)$. Then we'll be able to relate the smoothness of derivatives of f to the accuracy of their approximation by derivatives of the interpolant, which is relevant to the analysis of the accuracy of spectral methods for PDEs.

We now show that the accuracy of the interpolant $p(x)$ and its derivatives depends on the magnitudes of the Fourier coefficients of the function f . It follows that

$$\begin{aligned} f^{(\nu)}(x) - p^{(\nu)}(x) &= \sum_{k=-\infty}^{\infty} (\mathrm{i}k)^{\nu} c_k e^{\mathrm{i}kx} - \sum_{k=-m}^m (\mathrm{i}k)^{\nu} \tilde{c}_k e^{\mathrm{i}kx} \\ &= \sum_{k=-m}^m (\mathrm{i}k)^{\nu} (c_k - \tilde{c}_k) e^{\mathrm{i}kx} + \sum_{|k|>m} (\mathrm{i}k)^{\nu} c_k e^{\mathrm{i}kx} \end{aligned}$$

Recall that the approximate Fourier coefficients \tilde{c}_k that we obtain from the trapezoidal rule and the exact Fourier coefficients are related as follows:

$$\tilde{c}_k = \cdots + c_{k-2n} + c_{k-n} + c_k + c_{k+n} + c_{k+2n} + \cdots$$

Therefore,

$$\left| f^{(\nu)}(x) - p^{(\nu)}(x) \right| \leq \sum_{k=-m}^m |k|^\nu |c_k - \tilde{c}_k| + \sum_{|k|>m} |k|^\nu |c_k| \leq 2 \sum_{|k|>m} |k|^\nu |c_k| \quad (1)$$

The infinity norm of a function $g(x)$ on $[0, 2\pi]$ is defined as $\|g\|_\infty = \sup_{x \in [0, 2\pi]} |g(x)|$, hence we have

$$\|f^{(\nu)} - p^{(\nu)}\|_\infty \leq 2 \sum_{|k|>m} |k|^\nu |c_k|$$

Now we show that the magnitudes of the Fourier coefficients depend on the smoothness of f . First, we define the one-norm of a function $g(x)$ on $[0, 2\pi)$ by

$$\|g\|_1 = \int_0^{2\pi} |g(x)| dx.$$

Also, we define $g(a^-)$ as follows:

$$g(a^-) := \lim_{x \rightarrow a^-} g(x).$$

Loosely speaking, the following result says that if the derivatives of order $0, \dots, \mu - 1$ are continuous and integrable (they are "nice") and the derivatives of order μ and $\mu + 1$ are absolutely integrable (but not necessarily periodic or continuous, i.e., not so nice but not too wild), then $c_k = \mathcal{O}(k^{-\mu-1})$, i.e., the Fourier coefficients decay as $\mathcal{O}(k^{-\mu-1})$, as $k \rightarrow \infty$.

Proposition (Smoothness of f and decay of Fourier coefficients) Suppose the periodic extension of a function f is in $C^{(\mu-1)}[0, 2\pi]$ (i.e., the order 0-th up to $\mu - 1$ -st derivatives of the periodic extension of f are continuous on $[0, 2\pi]$; note that continuity on $[0, 2\pi]$ implies 2π -periodicity) and suppose $\|f^{(k)}\|_1 < \infty$ for $k = 0, \dots, \mu + 1$ and $f^{(\mu)}(0), f^{(\mu)}(2\pi^-) < \infty$, then

$$|c_k| \leq \frac{M}{2\pi|k|^{\mu+1}},$$

where $M = |f^{(\mu)}(2\pi^-) - f^{(\mu)}(0)| + \|f^{(\mu+1)}\|_1$.

Proof Integrating by parts $\mu + 1$ times, it follows that

$$\begin{aligned}
c_k &= \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx \\
&= \frac{1}{2\pi(-ik)} \left\{ f(x) e^{-ikx} \Big|_0^{2\pi^-} - \int_0^{2\pi} f'(x) e^{-ikx} dx \right\} \\
&= \frac{1}{2\pi(ik)} \int_0^{2\pi} f'(x) e^{-ikx} dx \\
&\vdots \\
&= \frac{1}{2\pi(ik)^{\mu-1}(-ik)} \left\{ f^{(\mu-1)}(x) e^{-ikx} \Big|_0^{2\pi^-} - \int_0^{2\pi} f^{(\mu)}(x) e^{-ikx} dx \right\} \\
&= \frac{1}{2\pi(ik)^\mu} \int_0^{2\pi} f^{(\mu)}(x) e^{-ikx} dx \\
&= \frac{1}{2\pi(ik)^\mu(-ik)} \left\{ f^{(\mu)}(x) e^{-ikx} \Big|_0^{2\pi^-} - \int_0^{2\pi} f^{(\mu+1)}(x) e^{-ikx} dx \right\},
\end{aligned}$$

from which the result follows. ■

Corollary (Accuracy of the trigonometric interpolant and its derivatives) If the conditions of the above proposition hold and n is odd with $n = 2m + 1$, then

$$\|f^{(\nu)} - p_n^{(\nu)}\|_\infty = \mathcal{O}(n^{\nu-\mu}), \quad n \rightarrow \infty.$$

Proof sketch Let $M = |f^{(\mu)}(2\pi^-) - f^{(\mu)}(0)| + \|f^{(\mu+1)}\|_1$, then we have that

$$\begin{aligned}
\|f^{(\nu)} - p^{(\nu)}\|_\infty &\leq 2 \sum_{|k|>m} |k|^\nu |c_k| \\
&\leq \frac{M}{\pi} \sum_{|k|>m} |k|^{\nu-\mu-1} \\
&= \frac{2M}{\pi} \sum_{k=m+1}^{\infty} k^{\nu-\mu-1} \\
&\leq \frac{2M}{(\nu-\mu)\pi} m^{\nu-\mu}
\end{aligned}$$

where $n = 2m + 1$ and we have used the fact that

$$\sum_{k=m+1}^{\infty} \frac{1}{k^{p+1}} \leq \int_m^{\infty} \frac{1}{k^{p+1}} dk = \frac{1}{pk^p}.$$

Hence, the result follows upon recalling the definition of big-O notation. ■

Corollary (accuracy of the approximate Fourier coefficients computed via the trapezoidal rule) If the conditions of the above proposition hold, then

$$c_k - \tilde{c}_k = \mathcal{O}(n^{-\mu-1}), \quad n \rightarrow \infty,$$

where $k = -m, \dots, m$ and $n = 2m + 1$.

Proof sketch Since

$$|\tilde{c}_k - c_k| \leq \sum_{\substack{p=-\infty \\ p \neq 0}}^{\infty} |c_{k+pn}| \leq \frac{M}{2\pi} \sum_{\substack{p=-\infty \\ p \neq 0}}^{\infty} \frac{1}{|k+pn|^{\mu+1}}$$

where $M = |f^{(\mu)}(2\pi^-) - f^{(\mu)}(0)| + \|f^{(\mu+1)}\|_1$. The result follows by showing that

$$\sum_{\substack{p=-\infty \\ p \neq 0}}^{\infty} \frac{1}{|k+pn|^{\mu+1}} = \mathcal{O}(n^{-\mu-1}), \quad n \rightarrow \infty.$$

■

As an aside, the sum

$$\sum_{\substack{p=-\infty \\ p \neq 0}}^{\infty} \frac{1}{|k+pn|^{\mu+1}}$$

can be expressed in terms of a special function called the Hurwitz zeta function, which is a generalisation of the Riemann zeta function. For odd μ , the sum can be expressed in terms of trigonometric functions. For example, for $k \neq 0$, and $\mu = 1$

$$\sum_{p=-\infty}^{\infty} \frac{1}{(k+np)^2} = \frac{\pi^2 \csc^2(k\pi/n)}{n^2},$$

which can be proved using the residue theorem from complex analysis.

An immediate consequence of this result is that the approximate Fourier coefficients \tilde{c}_k decay at the same rate as the exact Fourier coefficients c_k .

Examples

- For the non-periodic function we considered before, $f(x) = e^{x-2\pi}$, we have $\mu = 0$ because none of the derivatives of its periodic extension (even the 0-th order derivative) are continuous. However, $\|f^{(k)}\|_1 < \infty$ for $k = 0, 1$ (verify) and $f(0), f(2\pi^-) < \infty$, therefore, we expect its Fourier coefficients to decay at the algebraic rate $c_k = \mathcal{O}(k^{-1})$. This is indeed the case because

$$c_k = \frac{e^{-2\pi}}{2\pi} \int_0^{2\pi} e^{x(1-ik)} dx = \frac{e^{-2\pi}}{2\pi} \left[\frac{e^{x(1-ik)}}{1-ik} \right]_0^{2\pi} = \frac{1 - e^{-2\pi}}{1 - ik}$$

- For the function $f(x) = |\sin(x)|$, the conditions of the above proposition are satisfied for $\mu = 1$ (verify) and therefore we expect its exact and approximate Fourier coefficients to decay at the rate $c_k = \mathcal{O}(k^{-2})$.
- For the function $f(x) = |\sin(x)|\sin(x)$, the conditions of the above proposition are satisfied for $\mu = 2$ and therefore we expect its exact and approximate Fourier coefficients to decay at the rate $c_k = \mathcal{O}(k^{-3})$.
- The function $f(x) = e^{-1/\sin^2(x/2)} \in C^\infty[0, 2\pi]$ (but it is not analytic on $[0, 2\pi]$) and therefore we expect its exact and approximate Fourier coefficients to decay faster than $c_k = \mathcal{O}(k^{-\mu-1})$ for all $\mu > 0$.

Let's check numerically that the approximate Fourier coefficients decay at their predicted rates for the functions above:

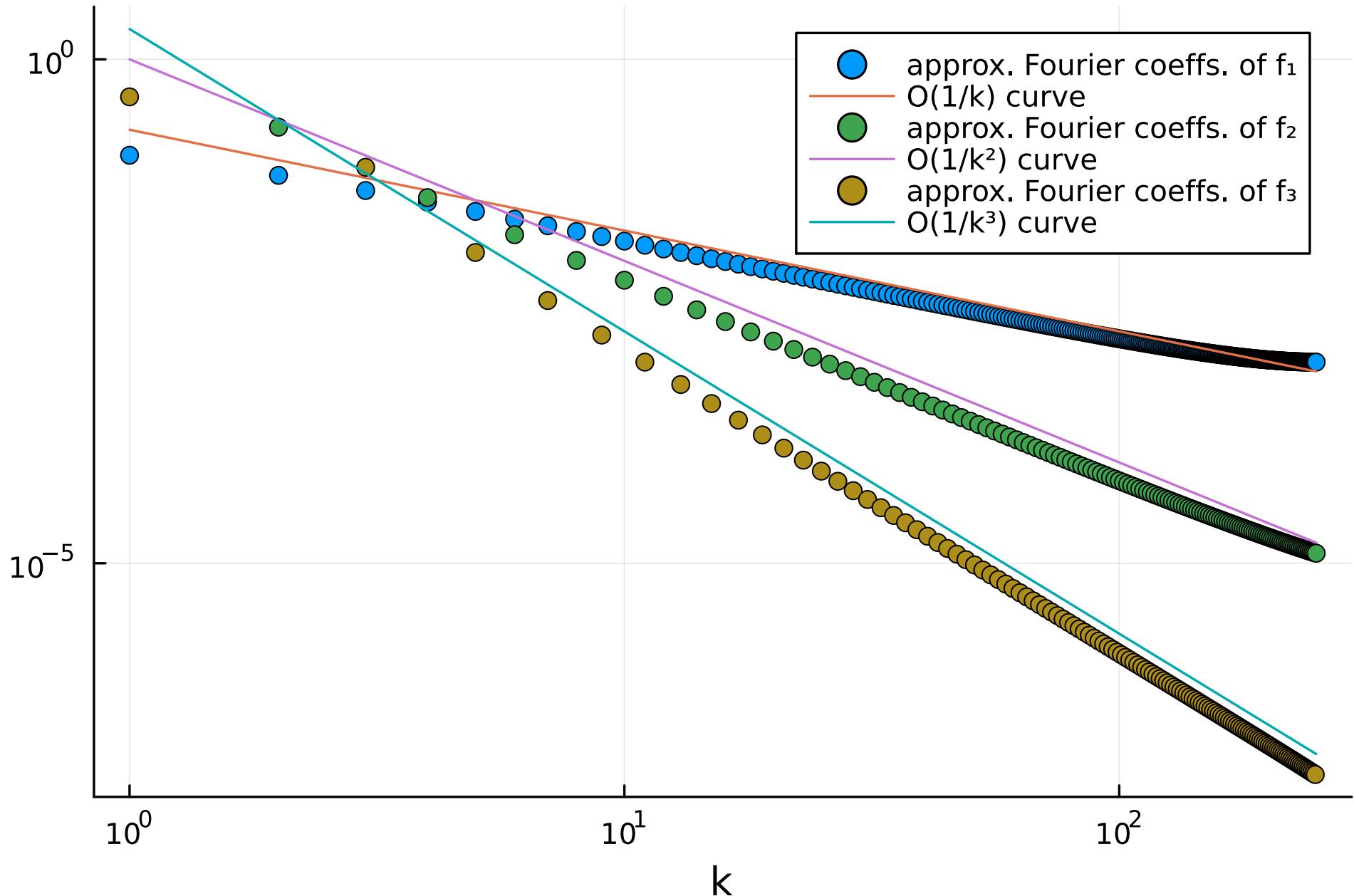
```
n = 501
m = (n-1)÷2
x = range(0,2π;length=n+1)[1:end-1]

f_1 = x -> exp(x-2π)
c = fftshift(fft(f_1.(x)))/n
scatter(1:m,abs.(c[m+2:end]);
yscale=:log10,xscale=:log10,xlabel="k",label="approx. Fourier
coeffs. of f_1")
plot!(1:m,0.2 ./(1:m),label="O(1/k) curve")
```

```
f_2 = x -> abs(sin(x))
c_2 = fftshift(fft(f_2.(x)))/n
scatter!(2:2:m,abs.(c_2[m+3:2:end]),label="approx. Fourier coeffs.
of f_2")
plot!(1:m,1 ./((1:m).^2),label="O(1/k^2) curve")
```

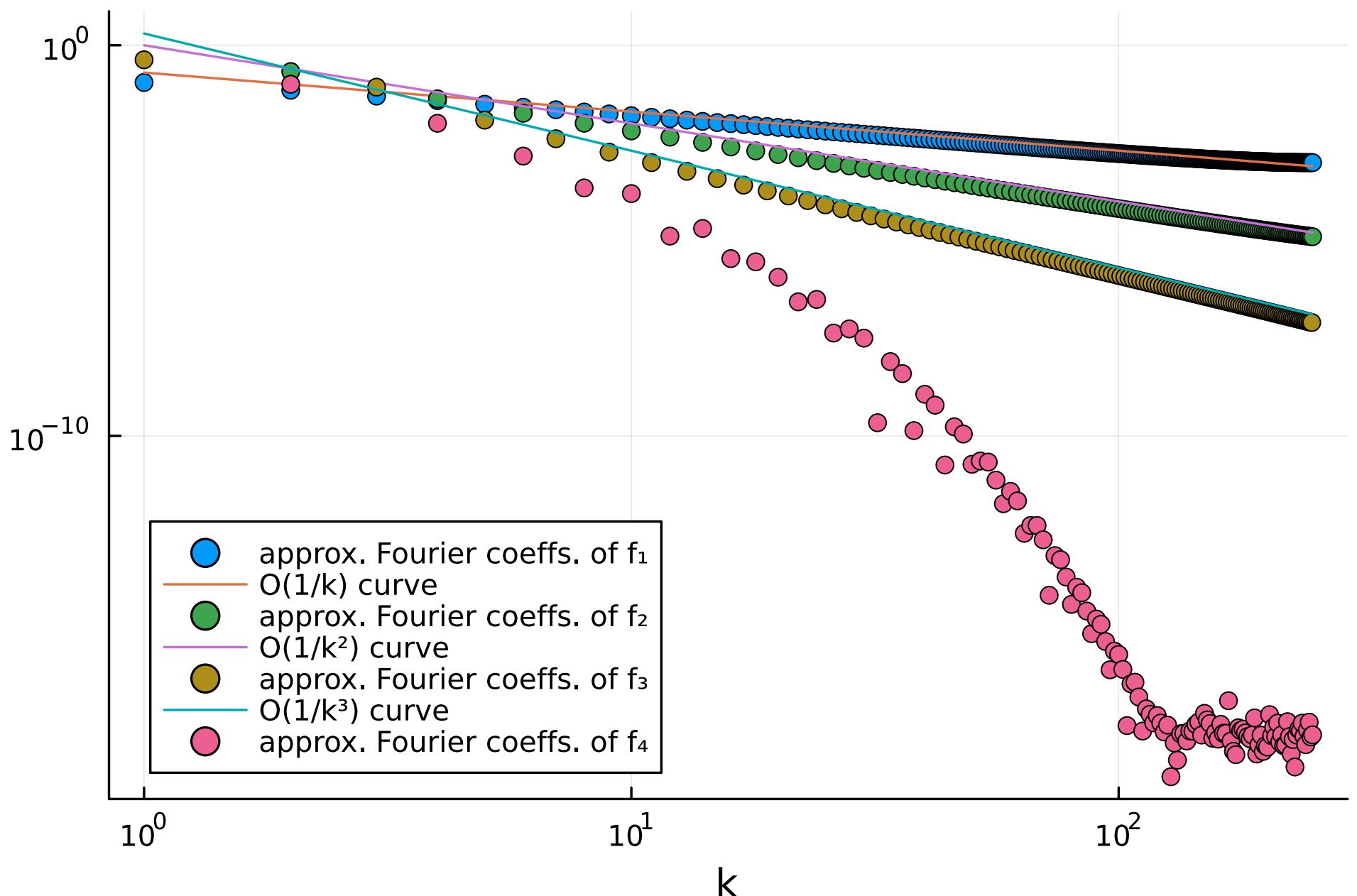
```
f_3 = x -> sin(x)*abs(sin(x))
c_3 = fftshift(fft(f_3.(x)))/n
scatter!(1:2:m,abs.(c_3[m+2:2:end]),label="approx. Fourier coeffs.
of f_3")
```

```
p = plot!(1:m, 2 ./((1:m).^3), label="O(1/k3) curve")
```



p

```
f_4 = x -> exp(-1/sin(x/2)^2)
c_4 = fftshift(fft(f_4.(x)))/n
scatter!(2:2:m,abs.(c_4[m+2:2:end]),label="approx. Fourier coeffs.
of f_4",legend=:bottomleft)
```



Functions that are analytic on $[0, 2\pi]$ are even smoother than functions in $C^\infty[0, 2\pi]$ and consequently their Fourier coefficients decay even faster.

Definition (analytic function) A function $f(x)$ is analytic on a region Ω (think of a real interval $[a, b] \in \mathbb{R}$ or a region in the complex plane for those who have studied complex analysis) if its Taylor series has a positive radius of convergence for every $x \in \Omega$.

The Fourier coefficients of functions that are analytic on a neighbourhood of $[0, 2\pi]$ decay *exponentially* fast. To prove this and to quantify the rate of convergence, one needs to use techniques from complex analysis. Since complex analysis is not a pre-requisite for this module, we won't prove this result. Another fact that is proven in complex analysis is that the derivatives of analytic functions are also analytic functions! Here are two consequences:

1. If f is analytic on $[0, 2\pi]$, then the approximate Fourier coefficients \tilde{c}_k (which we obtained via the trapezoidal rule) converge exponentially fast to the exact Fourier coefficients c_k of f as $n \rightarrow \infty$.
2. The trigonometric interpolant and its derivatives converge exponentially fast with n to f and its derivatives if f is analytic.

A function $f(x)$ is called an *entire function* if it is analytic on the entire complex plane; e^x , $\cos x$ and $\sin x$ are examples of entire functions. The Fourier coefficients of entire functions decay at a *super-exponential* rate.

A function $f(x)$ with $c_k = 0$ for $|k| > N$, where N is a non-negative integer, is called a *band-limited* function; $\cos x$ and $\sin x$ are examples of band-limited functions because $c_k = 0$ for $|k| > 1$ for these functions.

Here is an example of the decay of the approximate Fourier coefficients of a function in

$C^\infty[0, 2\pi]$, an analytic function and an entire function. Note the plot below is on a semi-logarithmic scale.

```
# A function in  $C^\infty[0, 2\pi]$ 
scatter(2:2:m,abs.(c_4[m+2:2:end]), xlabel="k",
yscale=:log10,label="approx. Fourier coeffs. of f_4")
# An analytic function
f_5 = x -> 1/(1.2 - sin(x))
c_5 = fftshift(fft(f_5.(x)))/n
scatter!(1:m,abs.(c_5[m+2:end]),label="approx. Fourier coeffs. of
f_5")
# An entire function
f_6 = x -> exp(cos(10x))
c_6 = fftshift(fft(f_6.(x)))/n
scatter!(0:m,abs.(c_6[m+1:end]),label="approx. Fourier coeffs. of
f_6")
```

