1 Chapter 1: From finite differences to a spectral method for periodic functions

Here we discuss

1. Finite difference (FD) approximations to the derivative of periodic functions
2. How to construct FD formulas using interpolating polynomials
3. How to analyse the accuracy of FD approximations using Taylor's theorem
4. Differentiation matrices
5. An example of exponentially fast convergence of a spectral method

1.1 Finite difference approximations to the derivative

Throughout this chapter, we consider a function $f(x)$ that is 2π-periodic, i.e., $f(x+2\pi) = f(x)$ for $x \in \mathbb{R}$ and we suppose we have function values at equally spaced points on the interval $[0, 2\pi)$, i.e., we have $f(x_j)$, where

$$x_j = jh, \qquad h = \frac{2\pi}{n}, \qquad j = 0, \ldots, n-1.$$

How can we approximate $f'(x_j)$?

Recall that

$$f'(x_j) = \lim_{\epsilon \to 0} \frac{f(x_j + \epsilon) - f(x_j)}{\epsilon},$$

this suggests that for small h,

$$f'(x_j) \approx \frac{f(x_j + h) - f(x_j)}{h} = \frac{f(x_{j+1}) - f(x_j)}{h}.$$

This is an example of a **finite difference** approximation to a derivative of a function. This particular approximation is known as a *forward difference* approximation (or formula). Replacing h with $-h$, we obtain a *backward difference* formula

$$f'(x_j) \approx \frac{f(x_j) - f(x_{j-1})}{h}.$$

Finite difference formulae are used to design finite difference methods for PDEs, as we'll see later in this module.

Here is the *central difference* formula

$$f'(x_j) \approx \frac{f(x_{j+1}) - f(x_{j-1})}{2h}.$$

To analyse the accuracy of finite difference formulae, we recall Taylor's Theorem:

Theorem (Taylor's Theorem) Let $a < b$, n a positive integer, $x \in [a, b]$ and $f : [a, b] \to \mathbb{R}$ be a continuous function such that the derivatives up to order $n + 1$ (inclusive) exist at every point in $[a, b]$. Then, for every $x \in [a, b]$, there exists a ξ between x and $x + h$ with $x + h \in [a, b]$, such that

$$f(x+h) = \sum_{k=0}^{n} \frac{h^k}{k!} f^{(k)}(x) + \frac{h^{n+1}}{(n+1)!} f^{(n+1)}(\xi).$$

Corollary (central difference is 2nd order accurate) Suppose $f \in C^3[x_{j-1}, x_{j+1}]$ (i.e., the derivatives of orders $0, 1, 2, 3$ are continuous on $[x_{j-1}, x_{j+1}]$), then

$$\left| \frac{f(x_{j+1}) - f(x_{j-1})}{2h} - f'(x_j) \right| \leq \frac{M}{6} h^2,$$

where $M = \sup_{x \in [x_{j-1}, x_{j+1}]} |f'''(x)|$.

Proof

It follows from Taylor's theorem that there exists an $\xi_1 \in [x_j, x_{j+1}]$ such that

$$f(x_{j+1}) = f(x_{j+1}) = f(x_j + h) = f(x_j) + hf'(x_j) + \frac{h^2}{2} f''(x_j) + \frac{h^3}{6} f'''(\xi_1).$$

Similarly, there exists an $\xi_2 \in [x_{j-1}, x_j]$ such that

$$f(x_{j-1}) = f(x_j) - hf'(x_j) + \frac{h}{2} f''(x_j) - \frac{h^3}{6} f'''(\xi_2).$$

Therefore,

$$\frac{f(x_{j+1}) - f(x_{j-1})}{2h} = f'(x_j) + \frac{h^2}{12} \left(f'''(\xi_1) + f'''(\xi_2) \right),$$

from which the result follows. ■

Recalling the definition of big O notation, we can write

$$f'(x_j) = \frac{f(x_{j+1}) - f(x_{j-1})}{2h} + \mathcal{O}(h^2), \qquad h \to 0,$$

and we say that the central difference approximation is second order accurate. Similarly, we can show that the forward and backward difference approximations are first order approximations of the derivative.

1.2 Finite differences via Lagrange interpolating polynomials

How can we construct finite difference approximations? One approach is via interpolating polynomials (aka polynomial interpolants or interpolatory polynomials).

1.2.1 Interpolation

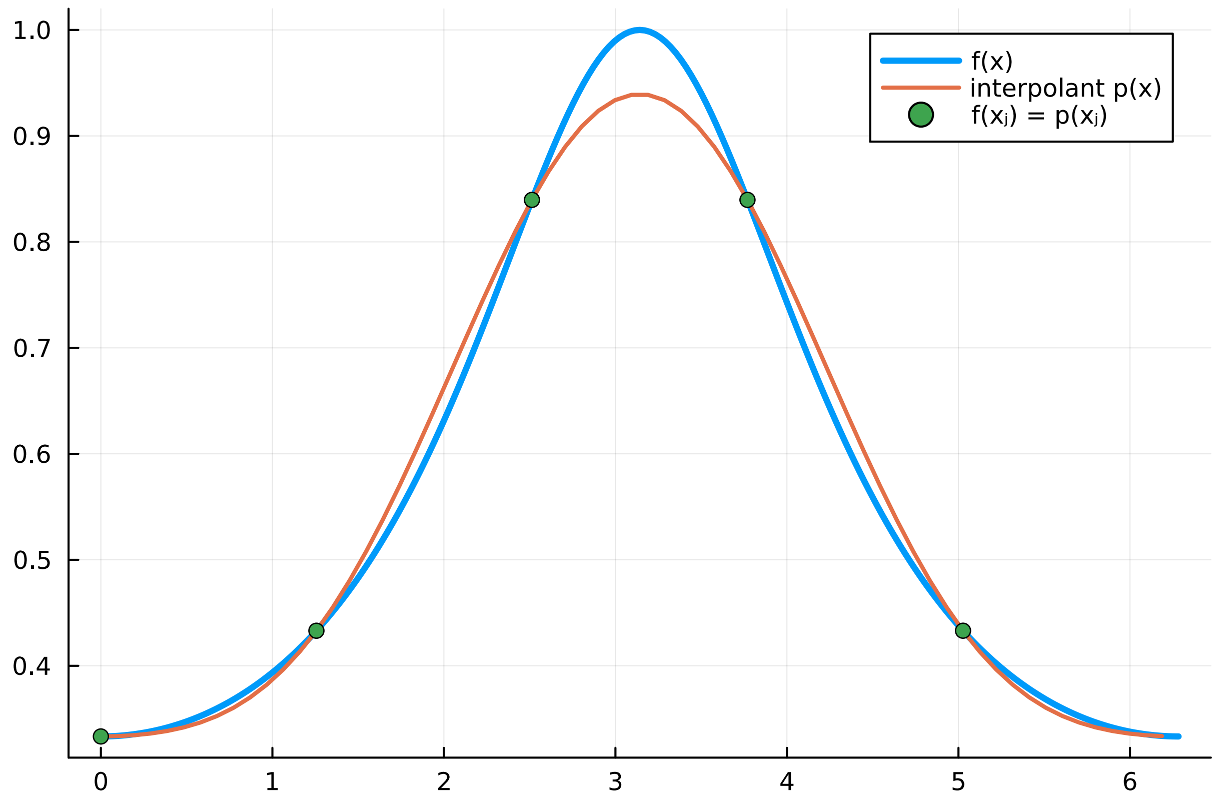
Definition (interpolation) We say that a function $p(x)$ interpolates a function $f(x)$ at the nodes / grid x_j, $j = 0, \ldots, n-1$ if

$$p(x_j) = f(x_j), \qquad j = 0, \ldots, n-1.$$

The function values $f(x_j)$ are sometimes referred to as samples.

Here we use ApproxFun.jl (or Chebfun in Matlab) to construct an interpolant of a 2π-periodic function. In the next lecture we'll learn that this interpolant is not a polynomial in x but rather it's a 'trigonometric interpolant' consisting of sines and cosines. (Later in this module, we'll learn more about the 'approximation technology' operating under the hood in ApproxFun and Chebfun.)

```julia
using ApproxFun, Plots, LinearAlgebra, SparseArrays, ToeplitzMatrices
# plot a 2π-periodic function f
f = Fun(x -> 1/(cos(x)+2),0..2π)
plot(f,lw=3,label="f(x)")
# construct an interpolant through n equally spaced points and plot it
n = 5
p = Fun(f,Fourier(0..2π),n) # interpolant
plot!(p,lw=2,label="interpolant p(x)")
# show the interpolant interpolates f(x) at the interpolation nodes
h = 2π/n;
xn = 0:h:2π-h # the interpolation nodes
scatter!(xn,p.(xn);label="f(x_j) = p(x_j)")
```



A central idea in approximation methods is to approximate a function (which could be a solution to a differential equation, for example) with an interpolant, which are typically constructed from simple functions (e.g., polynomials, sines, cosines). Hence, we approximate $f(x) \approx p(x)$ and here the idea is to construct approximations to $f'(x)$ by differentiating $p(x)$, then we set $f'(x) \approx p'(x)$.

1.2.2 Lagrange interpolation polynomials

We can construct interpolatory polynomials directly. We will use the following polynomials which are equal to 1 at one grid point and are zero at all the others:

Definition (Lagrange basis polynomial) The *Lagrange basis polynomial* on the grid (or at the nodes) x_j, $j = 0, \ldots, n-1$ is defined as

$$\ell_k(x) := \prod_{\substack{j=0 \\ j \neq k}}^{n-1} \frac{x - x_j}{x_k - x_j} = \frac{(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_{n-1})}{(x_k - x_1) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_{n-1})}$$

Plugging in the grid points verifies the following property:

Proposition (delta interpolation)

$$\ell_k(x_j) = \delta_{kj} = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{if } j \neq k \end{cases}$$

We can use these to construct the interpolatory polynomial:

Theorem (Lagrange interpolation) The unique polynomial of degree at most $n-1$ that interpolates f at the n distinct points x_j is:

$$p_n(x) = f(x_0)\ell_0(x) + \cdots + f(x_{n-1})\ell_{n-1}(x)$$

Proof Note that

$$p_n(x_j) = \sum_{j=0}^{n-1} f(x_j)\ell_k(x_j) = f(x_j), \qquad j = 0, \ldots, n-1,$$

and the fact that $p_n(x)$ is of degree $\leq n-1$ follows from the definition of the Lagrange basis polynomials, so we just need to show it is unique. Suppose $\tilde{p}_n(x)$ is a polynomial of degree at most $n-1$ that also interpolates f. Then $\tilde{p}_n - p_n$ vanishes at n distinct points. Thus by the fundamental theorem of algebra it must be zero.

■

Example Construct the central difference approximation as follows: (i) construct the polynomial $p_3(x)$ that interpolates $f(x)$ at the three equally spaced points x_{j-1}, x_j, x_{j+1}; (ii) use $p_3(x)$ to approximate f' at x_j.

Using the formula for the Lagrange interpolating polynomial given above, it follows that (verify!)

$$p_3(x) = \frac{(x - x_j)(x - x_{j+1})}{2h^2} f(x_{j-1}) - \frac{(x - x_{j-1})(x - x_{j+1})}{h^2} f(x_j) + \frac{(x - x_{j-1})(x - x_j)}{2h^2} f(x_{j+1})$$

and

$$p_3'(x_j) = \frac{f(x_{j+1}) - f(x_{j-1})}{2h} \approx f'(x_j),$$

which is the central difference approximation we've seen before.

1.3 Differentiation matrices

Collecting the approximations $p_3'(x_j) = \frac{f(x_{j+1})-f(x_{j-1})}{2h} \approx f'(x_j)$ for $j = 0, \ldots, n-1$ in a vector and using the fact that $f(x_{j+n}) = f(x_j + 2\pi) = f(x_j)$, we obtain

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

Note that omitted entries of the matrix are zero. In vector notation, we can express this as

$$\mathbf{f}' \approx \mathbf{p}' = D_n \mathbf{f}.$$

The matrix D_n is an example of a **differentiation matrix**; this differentiation matrix is skew-symmetric ($D_n^\top = -D_n$), Toeplitz (a matrix that is constant along its diagonals) and circulant.

We can use the following command in Julia to create a sparse version of the differentiation matrix:

```
n = 10
spdiagm(1=>fill(0.5,n-1),-1=>fill(-0.5,n-1),n-1=>fill(-0.5,1),1-n=>fill(0.5,1))
```

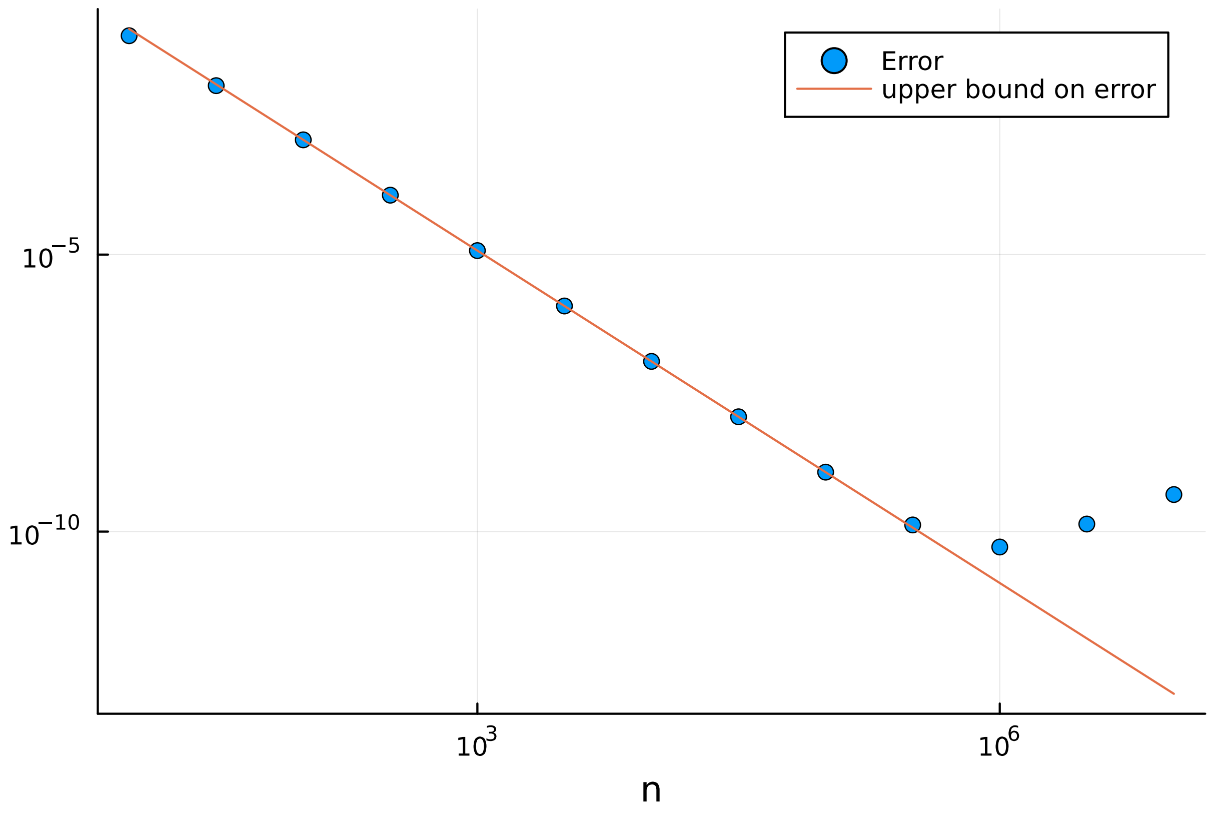
```
10×10 SparseArrays.SparseMatrixCSC{Float64, Int64} with 20 stored entries:
   ⋅    0.5    ⋅      ⋅      ⋅      ⋅      ⋅      ⋅      ⋅   -0.5
 -0.5    ⋅    0.5     ⋅      ⋅      ⋅      ⋅      ⋅      ⋅     ⋅
   ⋅   -0.5    ⋅     0.5     ⋅      ⋅      ⋅      ⋅      ⋅     ⋅
   ⋅     ⋅   -0.5     ⋅     0.5     ⋅      ⋅      ⋅      ⋅     ⋅
   ⋅     ⋅     ⋅    -0.5     ⋅     0.5     ⋅      ⋅      ⋅     ⋅
   ⋅     ⋅     ⋅      ⋅    -0.5     ⋅     0.5     ⋅      ⋅     ⋅
   ⋅     ⋅     ⋅      ⋅      ⋅    -0.5     ⋅     0.5     ⋅     ⋅
   ⋅     ⋅     ⋅      ⋅      ⋅      ⋅    -0.5     ⋅     0.5    ⋅
   ⋅     ⋅     ⋅      ⋅      ⋅      ⋅      ⋅    -0.5     ⋅    0.5
  0.5    ⋅     ⋅      ⋅      ⋅      ⋅      ⋅      ⋅    -0.5    ⋅
```

For various values of n, let's compute the maximum error of the central difference approximation to the derivative at the points $x_j = jh$ with $j = 0, \ldots, n-1$ and $h = 2\pi/n$ for the function $f(x) = 1/(2 + \cos(x))$:

```
f = x -> 1/(2 + cos(x))
Df = x -> sin(x)/(2 + cos(x))^2  # derivative
# compute the errors
@time begin
errs =
[( h = 2π/n;
   # differentiation matrix:
   Dn =
1/h*spdiagm(1=>fill(0.5,n-1),-1=>fill(-0.5,n-1),n-1=>fill(-0.5,1),1-n=>fill(0.5,1));
   x = range(0,2π;length=n+1)[1:end-1]; # the nodes x_j, j = 0, ..., n-1
   # compute the maximum error at the nodes
   norm(Dn*f.(x) - Df.(x),Inf) ) for n = Int64.(round.(10 .^(1:0.5:7)))]
end;
# plot the errors on a log-log scale
D3f = x -> (16*sin(2x) + 7*sin(x) - sin(3x))/(4*(2 + cos(x))^4) # third derivative
```

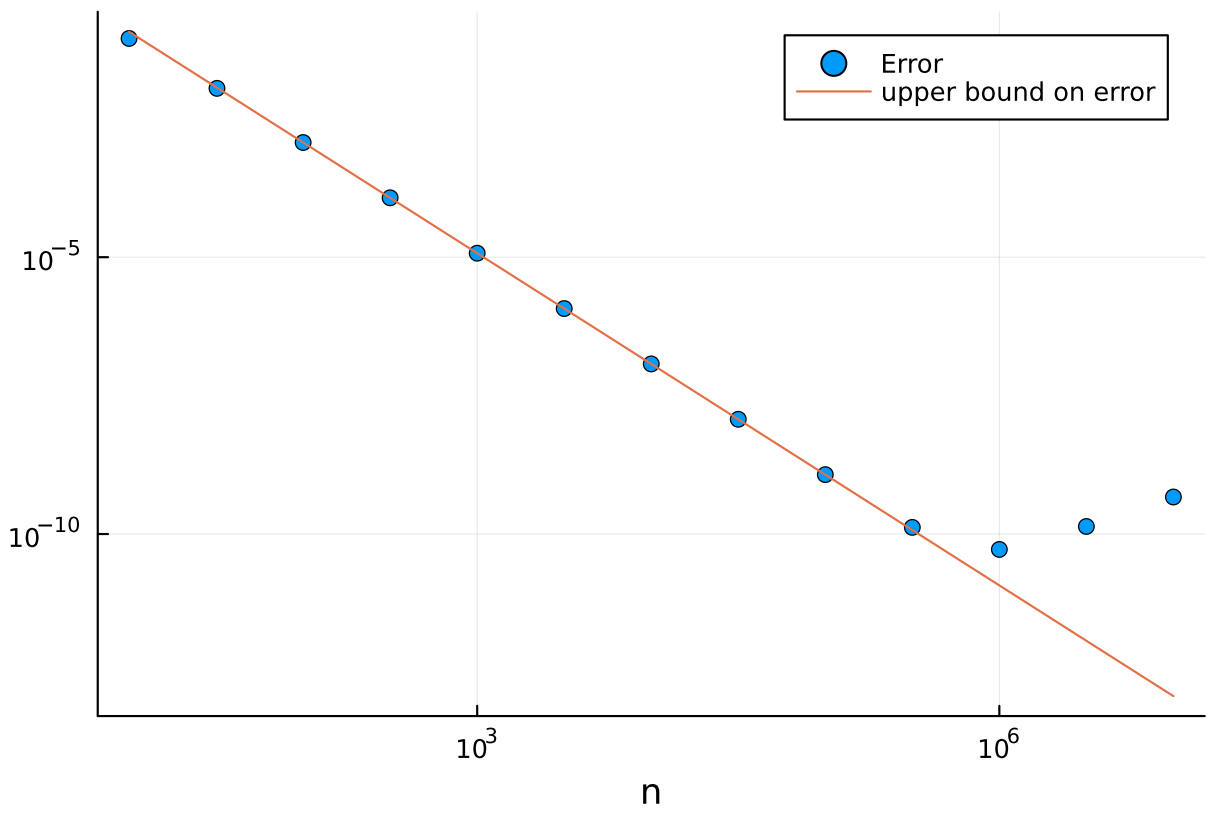
```
# estimate the maximum absolute value of the third derivative on [-π, π]
xx = range(0,2π,1001)
M = norm(D3f.(xx),Inf)
n = 10 .^(1:0.5:7)
scatter(n,errs;xscale=:log10,yscale=:log10,label="Error",xlabel="n")
plot!(n,M*(2π)^2/6*n.^(-2);label="upper bound on error")
```

```
4.506398 seconds (1.71 M allocations: 3.141 GiB, 21.40% gc time, 22.36% c
ompilation time)
```



```
f = x -> 1/(2 + cos(x))
Df = x -> sin(x)/(2 + cos(x))^2  # derivative
# compute the errors
nv = Int64.(round.(10 .^(1:0.5:7)))
@time begin
errs =
[( h = 2π/n;
   # differentiation matrix:
   Dn =
1/h*spdiagm(1=>fill(0.5,n-1),-1=>fill(-0.5,n-1),n-1=>fill(-0.5,1),1-n=>fill(0.5,1));
   x = range(0,2π;length=n+1)[1:end-1]; # the nodes x_j, j = 0, ..., n-1
   # compute the maximum error at the nodes
   norm(Dn*f.(x) - Df.(x),Inf) ) for n = nv]
end;
# plot the errors on a log-log scale
D3f = x -> (16*sin(2x) + 7*sin(x) - sin(3x))/(4*(2 + cos(x))^4) # third derivative
# estimate the maximum absolute value of the third derivative on [-π, π]
xx = range(0,2π,1001)
M = norm(D3f.(xx),Inf)
scatter(nv,errs;xscale=:log10,yscale=:log10,label="Error",xlabel="n")
plot!(nv,M*(2π)^2/6*nv.^(-2);label="upper bound on error")
```

```
3.493785 seconds (490.76 k allocations: 3.078 GiB, 17.46% gc time, 7.94%
compilation time)
```



The plot shows that the error decays as $\mathcal{O}(n^{-2})$, $n \to \infty$.

To derive a higher order finite difference approximation to the derivative, we can use a higher degree interpolating polynomial. For example, let m be even, then we can derive an m-th order central difference approximation to the derivative by

1. constructing an interpolating polynomial $p_{m+1}(x)$ such that p_{m+1} interpolates f at x_j, $x_{j\pm1}$, $x_{j\pm2}$, …, $x_{j\pm m/2}$ and
2. making the approximation $f'(x_j) \approx p'_{m+1}(x_j)$.

We have already carried out this procedure for $m = 2$. If we take $m = 4$, then we would obtain the following fourth-order central difference approximation to the derivative:

$$f'(x_j) \approx p_5'(x_j) = \frac{f(x_{j-2}) - 8f(x_{j-1}) + 8f(x_{j+1}) - f(x_{j-2})}{12h},$$

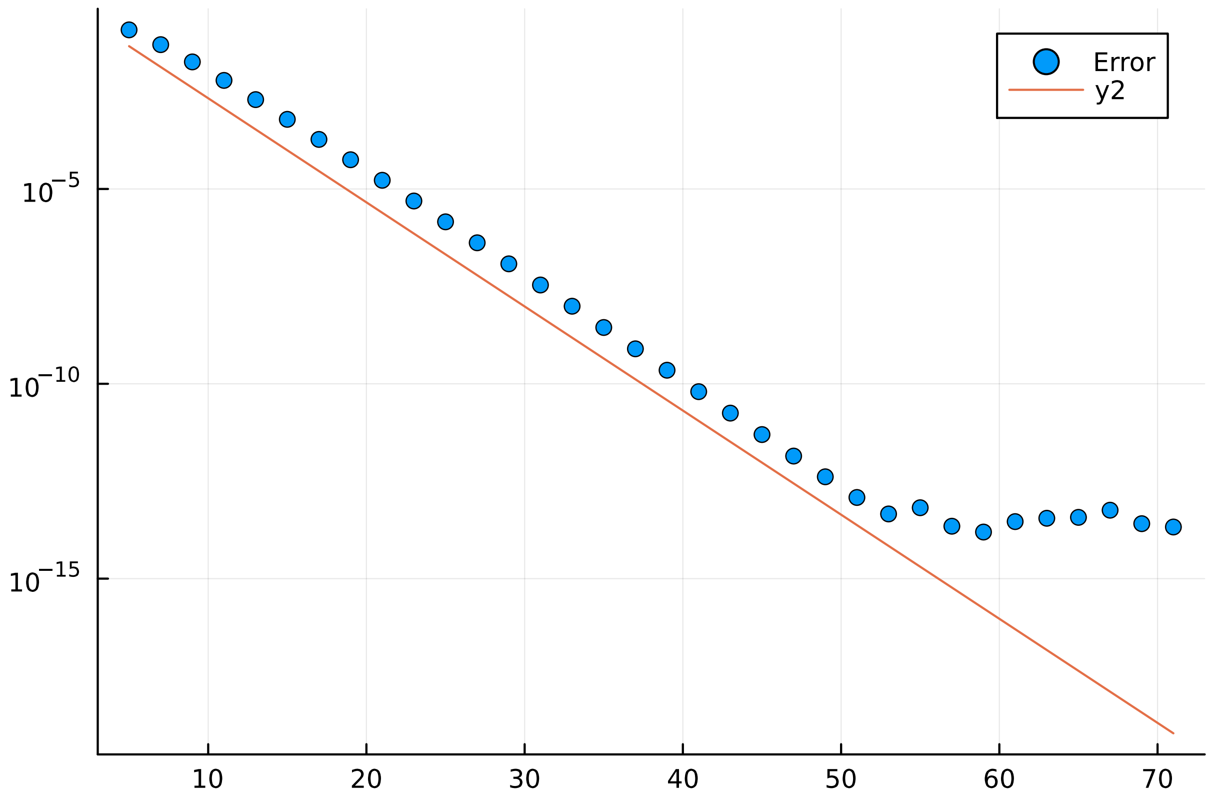
whose error behaves as $\mathcal{O}(h^4)$, $h \to 0$ (or, equivalently, as $\mathcal{O}(n^{-4})$, $n \to \infty$) if f is sufficiently smooth (e.g., if $f \in C^5[x_{j-2}, x_{j+2}]$).

What would the central difference approximation to the derivative be if we let $m \to \infty$? We won't prove this (see *A Practical Guide to Pseudospectral Methods* by B. Fornberg if you're interested in the details) but using the fact that f is 2π-periodic, it can be shown that the central difference approximation obtained by letting $m \to \infty$ is equivalent to applying the following $n \times n$ differentiation matrix

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

where $h = 2\pi/n$. In the next chapter we shall derive this differentiation matrix via trigonometric interpolants. Since this differentiation matrix is derived by letting $m \to \infty$, we expect the error to decay faster than $\mathcal{O}(n^{-m})$ for all $m > 0$, provided $f \in C^\infty[0, 2\pi]$. Let's check this numerically:

```
Nv = 5:2:71;
errs =
[( h = 2π/N;
    column = [0; 0.5*(-1).^(1:N-1).*csc.((1:N-1)*h/2)];
    # Differentiation matrix:
    D_m = Toeplitz(column,[column[1]; column[N:-1:2]]);
    # N equally spaced points on [-π, π)
    x = range(0,2π;length=N+1)[1:end-1];
    # maximum error at the set of N points
    norm(D_m*f.(x) - Df.(x),Inf) ) for N = Nv]
# plot the error on a semi-log (log-linear) scale
scatter(Nv,errs;yscale=:log10,label="Error")
plot!(Nv,(1.85).^(-Nv))
```



Note that the error decays linearly on a semi-logarithimc plot (or log-linear plot). This means that the error decays *exponentially fast* with n (we say that the approximation converges exponentially fast). That is, the error decays as $\mathcal{O}(c^{-n})$ for $n \to \infty$ with $c > 1$. Recall that the error of the second-order central difference formula decayed as $\mathcal{O}(n^{-2})$. This type of converge is called algebraic convergence. Note how much faster exponential convergence is compared to algebraic convergence.

To derive finite difference approximations to the derivative, we used low degree polynomials to approximate f' locally. The idea behind *spectral methods* for differential equations (including PDEs) is to use high degree polynomials (or high accuracy interpolants) to approximate solutions to differential equations.

1.4 Exercises

1. **(Second-order central difference approximation to the second derivative)** Show that if $f \in C^4[x_{j-1}, x_{j+1}]$, then

$$f''(x_j) = \frac{f(x_{j+1}) - 2f(x_j) + f(x_{j-1})}{h^2} + \mathcal{O}(h^2), \qquad h \to 0.$$

2. Run the Julia code in this chapter and / or the Matlab code for this chapter (see the Blackboard page of this module) on your own machine. To run the Julia code, see the document titled Julia on Blackboard. To run the Matlab code, you'll need to use Chebfun (see the instructions for how to do this here).

3. Construct, by hand, a polynomial that interpolates $f(x) = \mathrm{e}^x$ at $x = 0, 1, 2$.

4. One-sided FD formula?

5. Numerical experiment with the fourth order central difference formula?