1 Chapter 5: Consistency, stability and convergence of methods for evolutionary PDEs

In this chapter we analyse and implement finite difference and spectral methods for the diffusion equation, the (variable coefficient) advection equation and the wave equation.

1.0.1 A finite difference method for the diffusion equation

Consider the $(1+1)$-dimensional diffusion equation,

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \qquad x \in (0,1), \qquad t \in (0,T]$$

subject to the initial and boundary data

$$u(x,0) = f(x), \qquad 0 \le x \le 1, \qquad u(0,t) = \varphi_0(t), \qquad u(1,t) = \varphi_1(t), \qquad t \ge 0.$$

We shall analyse a finite difference method for this PDE initial / boundary value problem.

We discretise the spatial variable as follows,

$$x_j = jh, \qquad h = \frac{1}{n_x + 1}, \qquad j = 0, \ldots, n_x + 1,$$

and the temporal variable,

$$t_i = i\tau, \qquad \tau = \frac{T}{n_t}, \qquad i = 0, 1, 2, \ldots, n_t.$$

Let

$$u(x_j, t_i) := \tilde{u}_j^i \approx u_j^i.$$

We set

$$u(x_j, 0) = \tilde{u}_j^0 = f(x_j) = u_j^0, \qquad j = 1, \ldots, n_x$$

and

$$u(0, t_i) = \tilde{u}_0^i = \varphi_0(t_i) = u_0^i, \qquad u(1, t_i) = \tilde{u}_{n_x+1}^i = \varphi_1(t_i) = u_{n_x+1}^i, \qquad i \ge 0.$$

1.0.2 Order and convergence

Let's replace the time derivative with a (first-order) forward difference and the second spatial derivative with a (second-order) central difference approximation: from Taylor's theorem (see Chapter 1), there exists a $\xi \in [t_i, t_{i+1}]$ such that

$$\frac{u(x_j, t_{i+1}) - u(x_j, t_i)}{\tau} = \frac{\tilde{u}_j^{i+1} - \tilde{u}_j^i}{\tau} = u_t(x_j, t_i) + \frac{\tau}{2} u_{tt}(x_j, \xi).$$

Let's assume that $u_{tt}(x,t)$ exists and is bounded on $(x,t) \in [0,1] \times [0,T]$, then we have

$$\frac{\tilde{u}_j^{i+1} - \tilde{u}_j^i}{\tau} = u_t(x_j, t_i) + \mathcal{O}(\tau), \qquad \tau \to 0.$$

Similarly, if we assume that $u_{xxxx}(x,t)$ exists and is bounded on $(x,t) \in [0,1] \times [0,T]$, then we have (see Chapter 1 Exercises)

$$\frac{\tilde{u}_{j+1}^i - 2\tilde{u}_j^i + \tilde{u}_{j-1}^i}{h^2} = u_{xx}(x_j, t_i) + \mathcal{O}(h^2), \qquad h \to 0.$$

We define

$$\mu = \frac{\tau}{h^2},$$

which is known as the Courant number and specify that μ be held constant as $\tau, h \to 0$. Hence, we have that

$$\frac{\tilde{u}_j^{i+1} - \tilde{u}_j^i}{\tau} - \frac{\tilde{u}_{j+1}^i - 2\tilde{u}_j^i + \tilde{u}_{j-1}^i}{h^2} = u_t(x_j, t_i) - u_{xx}(x_j, t_i) + \mathcal{O}(h^2) = \mathcal{O}(h^2), \qquad h \to 0.$$

We shall approximate the solution to the diffusion equation with the finite difference method

$$\frac{u_j^{i+1} - u_j^i}{\tau} - \frac{u_{j+1}^i - 2u_j^i + u_{j-1}^i}{h^2} = 0$$

or

$$u_j^{i+1} = u_j^i + \mu\left(u_{j+1}^i - 2u_j^i + u_{j-1}^i\right).$$

This method is known as **Euler's method**. Notice we have shown that if the exact solution is substituted into the finite difference method, then the PDE is recovered exactly and the local error tends to zero as the discretisation parameters tend to zero. The method is **second order** since the local error approaches zero at the rate $\mathcal{O}(h^2)$ as $h \to 0$. A finite difference method of order $p \geq 1$ is said to be **consistent**. Notice that for consistency, we only require the error at (x_j, t_i) to go to zero as the discretisation parameters tend to zero, hence the order of a method measures the local error. A method is said to be **convergent** if the *global* error tends to zero as the discretisation parameters tend to zero, i.e., if

$$\lim_{h\to 0}\left[\lim_{j\to x/h}\left(\lim_{i\to t/\tau} u_j^i\right)\right] = u(x,t), \qquad x \in [0,1], \qquad t \in [0,T],$$

where $\mu = \tau/h^2$ is kept constant as $h \to 0$.

1.0.3 Numerical experiments with Euler's method

Before we analyse the convergence or otherwise of Euler's method, let's perform some numerical experiments. First, we note that we can express Euler's method as

$$\underbrace{\begin{bmatrix} u_1^{i+1} \\ \vdots \\ \vdots \\ \vdots \\ u_{n_x}^{i+1} \end{bmatrix}}_{\mathbf{u}^{i+1}} = \underbrace{\begin{bmatrix} 1-2\mu & \mu & & & \\ \mu & 1-2\mu & \mu & & \\ & \ddots & \ddots & \ddots & \\ & & \mu & 1-2\mu & \mu \\ & & & \mu & 1-2\mu \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u_1^{i} \\ \vdots \\ \vdots \\ \vdots \\ u_{n_x}^{i} \end{bmatrix}}_{\mathbf{u}^{i}} + \underbrace{\begin{bmatrix} \mu\varphi_0(t_i) \\ 0 \\ \vdots \\ 0 \\ \mu\varphi_1(t_i) \end{bmatrix}}_{\mathbf{k}^{i}}$$

i.e.,

$$\mathbf{u}^{i+1} = A\mathbf{u}^i + \mathbf{k}^i, \qquad i = 0, \ldots, n_t - 1.$$

```
using LinearAlgebra, Plots, SparseArrays, OrdinaryDiffEq, Printf, SparseArrays, FFTW,
ApproxFun

function Euler(f,ϕ0,ϕ1,nx,μ,T)

    x = range(0,1,nx + 2)
    h = 1/(nx+1)
    τ = μ*h^2
    t = 0:τ:T
    nt = length(t)-1
    A = SymTridiagonal(fill((1 - 2μ),nx),fill(μ,nx-1))
    u = zeros(nt+1,nx+2)
    u[:,1] = ϕ0.(t)
    u[:,nx+2] = ϕ1.(t)
    u[1,2:nx+1] = f.(x[2:nx+1])

    for i = 1:nt
        u[i+1,2:nx+1] = A*u[i,2:nx+1]
        u[i+1,2] += μ*ϕ0(t[i])
        u[i+1,nx+1] += μ*ϕ1(t[i])
    end

    u, x, t
end
```

```
Euler (generic function with 1 method)
```

The diffusion initial / boundary value problem with

$$u(x,0) = f(x) = \sin\frac{1}{2}\pi x + \frac{1}{2}\sin 2\pi x, \qquad 0 \le x \le 1,$$

and

$$u(0,t) = \varphi_0(t) = 0, \qquad u(1,t) = \varphi_1(t) = \mathrm{e}^{-\pi^2 t/4}, \qquad t \ge 0,$$

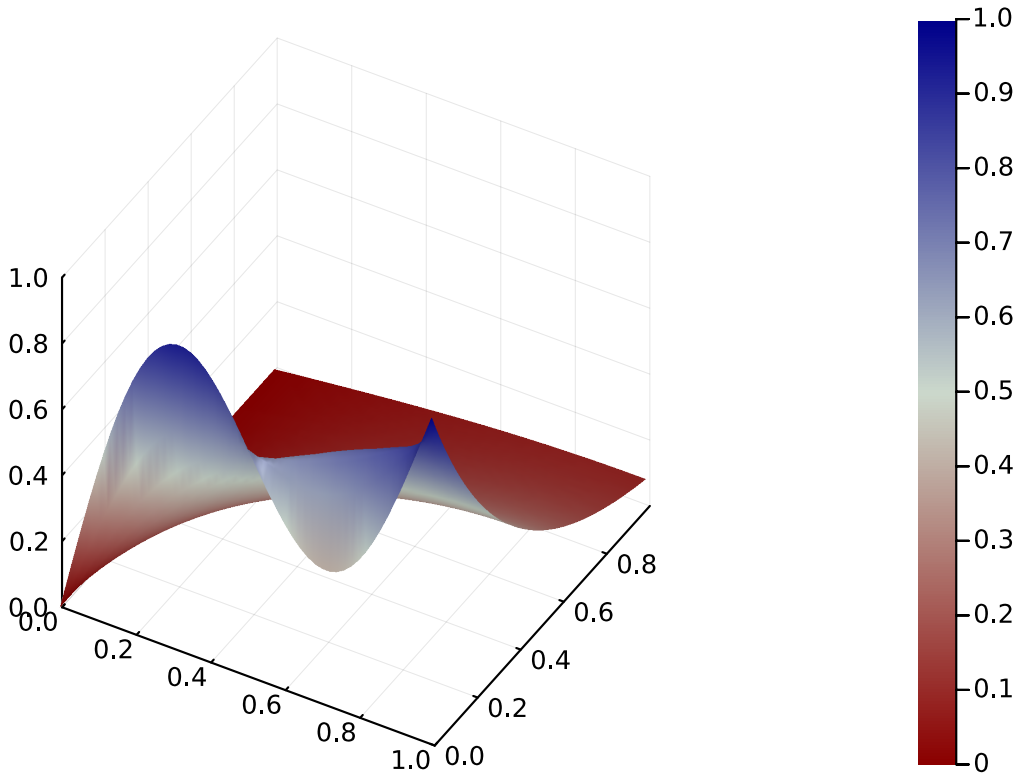
has the exact solution

$$u(x,t) = \mathrm{e}^{-\pi^2 t/4}\sin\frac{1}{2}\pi x + \frac{1}{2}\mathrm{e}^{-4\pi^2 t}\sin 2\pi x, \qquad 0 \le x \le 1, \qquad t \ge 0.$$

```
f = x -> sin(π*x/2) + 0.5*sin(2π*x)
ϕ1 = t -> exp(-π^2*t/4)
ϕ0 = t -> 0
nx = 50
μ = 0.50
T = 1
u,x,t = Euler(f,ϕ0,ϕ1,nx,μ,T)
nt = length(t) -1
```

```
5202
```

```
inc = 100
surface(x,t[1:inc:end],u[1:inc:end,:],seriescolor=:redsblues, camera=(30,40))
```




```
@gif for i = 2:50:nt+1
    plot(x, u[1,:], size = (500, 150), label = "u0")
    plot!(x, u[i,:], label = "t = " * (@sprintf("%.2f", t[i])), legend = :outertopright)
end
```

```
Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_f9tRm4Aqoo.gi
f")
```

```
xx = x' .* ones(nt+1)
tt = ones(nx+2)' .* t
ue = (x,t) -> exp(-π^2*t/4)*sin(π*x/2) + 0.5*exp(-4*π^2*t)*sin(2π*x)
error = ue.(xx,tt) - u
e1 = maximum(error)
```

```
0.00047001331601359553
```

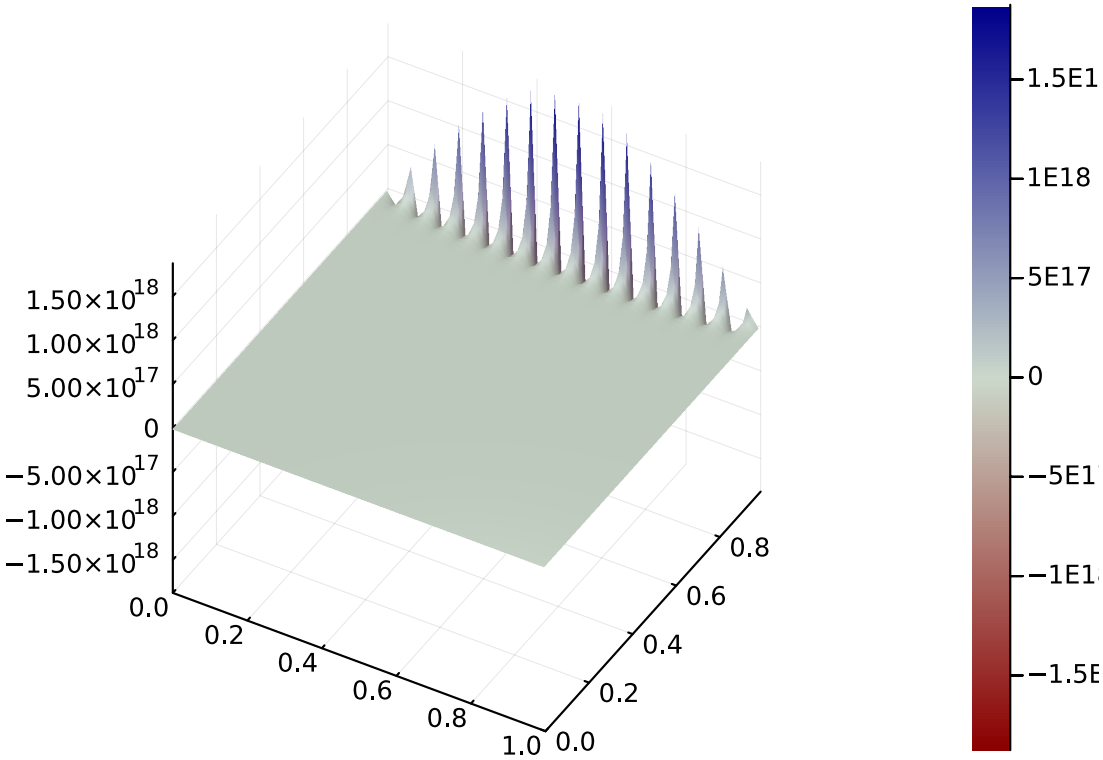
If we double n_x, the the error decreases by roughly a factor of 4.

```
nx *= 2
u,x,t = Euler(f,ϕ0,ϕ1,nx,μ,T)
@show nt = length(t) - 1
xx = x' .* ones(nt+1)
tt = ones(nx+2)' .* t
error = ue.(xx,tt) - u
e2 = maximum(error)
e1/e2
```

```
nt = length(t) - 1 = 20402
3.929469183388294
```

This suggests the error decays as $\mathcal{O}(n_x^{-2})$, $n_x \to \infty$.

```
nx = 30
μ = 0.51
u,x,t = Euler(f,ϕ0,ϕ1,nx,μ,T);

surface(x,t[1:30:end],u[1:30:end,:],seriescolor=:redsblues, camera=(30,40))
```




```
nt = length(t)-1
@gif for i = 2:4:nt÷2
    plot(x, u[1,:], size = (500, 150), label = "u0")
    plot!(x, u[i,:], label = "t = " * (@sprintf("%.2f", t[i])), legend = :outertopright)
end
```

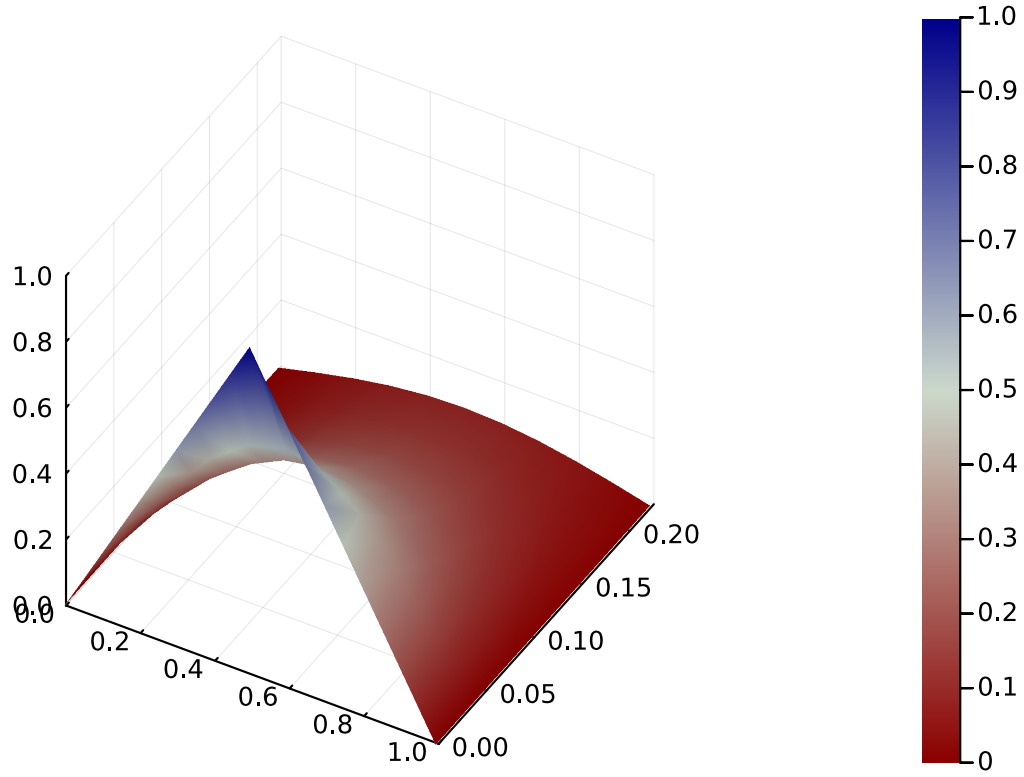
```
Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_ishCEcc355.gi
f")
```

Note: The gif animations don't appear in the PDF version of this chapter, however they can be viewed in the html version, which is available on Blackboard.

Based on these experiments, we conjecture that Euler's method converges if $\mu \leq \frac{1}{2}$.

The initial condition is "smoothed out" by the diffusion equation. Here is another example:

```
f = x -> x <= 0.5 ? 2*x : 2-2*x
ϕ1 = t -> 0
ϕ0 = t -> 0
nx = 19
μ = 0.50
T = 0.25
u,x,t = Euler(f,ϕ0,ϕ1,nx,μ,T);

surface(x,t[1:30:end],u[1:30:end,:],seriescolor=:redsblues, camera=(30,40))
```




```
nt = length(t)-1
@gif for i = 2:nt
    plot(x, u[1,:], size = (500, 150), label = "u0")
    plot!(x, u[i,:], label = "t = " * (@sprintf("%.2f", t[i])), legend = :outertopright)
end

Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_kHffoKkCsZ.gi
f")
```

1.0.4 Convergence of Euler's method

Proposition If $\mu \leq \frac{1}{2}$, then Euler's method converges.

Proof Recall that

$$\frac{\tilde{u}_j^{i+1} - \tilde{u}_j^i}{\tau} - \frac{\tilde{u}_{j+1}^i - 2\tilde{u}_j^i + \tilde{u}_{j-1}^i}{h^2} = \mathcal{O}(h^2), \qquad h \to 0,$$

which implies

$$\tilde{u}_j^{i+1} = \tilde{u}_j^i + \mu\left(\tilde{u}_{j+1}^i - 2\tilde{u}_j^i + \tilde{u}_{j-1}^i\right) + \mathcal{O}(h^4), \qquad h \to 0.$$

Let $e_j^i = \tilde{u}_j^i - u_j^i$, then since

$$u_j^{i+1} = u_j^i + \mu\left(u_{j+1}^i - 2u_j^i + u_{j-1}^i\right),$$

we have

$$e_j^{i+1} = e_j^i + \mu\left(e_{j+1}^i - 2e_j^i + e_{j-1}^i\right) + \mathcal{O}(h^4), \qquad h \to 0.$$

This means that for sufficiently small $h > 0$, there exists a constant c such that

$$\left|e_j^{i+1} - e_j^i - \mu\left(e_{j+1}^i - 2e_j^i + e_{j-1}^i\right)\right| \leq ch^4.$$

Hence, for sufficiently small h, we have

$$\begin{aligned} |e_j^{i+1}| &\leq \left|e_j^i + \mu(e_{j+1}^i - 2e_j^i + e_{j-1}^i)\right| + ch^4 \\ &\leq \mu|e_{j-1}^i| + |1-2\mu||e_j^i| + \mu|e_{j+1}^i| + ch^4 \\ &\leq (2\mu + |1-2\mu|)\,\eta^i + ch^4 \end{aligned}$$

where

$$\eta^i = \max_{j=0,\ldots,n_x+1} |e_j^i|.$$

The above inequality holds for $|e_j^{i+1}|$, with $j = 0, \ldots, n_x + 1$ and therefore it also holds for $|\eta^{i+1}|$. Since $0 < \mu \leq 1/2$ and $u_j^0 = \tilde{u}_j^0$, which implies that $\eta^0 = 0$, we have that

$$\eta^{i+1} \leq \eta^i + ch^4 \leq \eta^{i-1} + 2ch^4 \leq \cdots \leq \eta^0 + (i+1)ch^4 = (i+1)ch^4.$$

Therefore

$$\eta^{n_t} \leq cn_t h^4 = \frac{c}{\mu} n_t \mu h^4 = \frac{c}{\mu} n_t \tau h^2 = \frac{c}{\mu} T h^2$$

We conclude that $\eta^i \to 0$ for $i = 0, \ldots, n_t$ as $h \to 0$ and therefore $e_j^i \to 0$ as $h \to 0$ for $j = 0, \ldots, n_x + 1$ and $i = 0, \ldots, n_t$ as $h \to 0$, which implies that the method converges. ■

Remark: Notice that the maximum error of the Euler method (assuming all relevant partial derivatives exist and are bounded) is bounded by $\frac{c}{\mu} T h^2$ for sufficiently small h. This confirms our numerical observation above that the method converges at the rate $\mathcal{O}(h^2)$ as $h \to 0$, or equivalently, at the rate $\mathcal{O}(n_x^{-2})$ as $n_x \to \infty$ (i.e., the method is second order).

1.0.5 Well-posedness and ill-posedness of PDE problems

A PDE intial and / or boundary value problem is **well-posed** if there exists a unique solution that depends continuously on the initial and boundary data. Otherwise, the PDE problem is said to be **ill-posed**.

Example: Suppose $\varphi_0(t) = \varphi_1(t) = 0$ (i.e., zero Dirichlet boundary conditions), then using the method of separation of variables, it can be shown that if

$$u(x,0) = f(x) = \sum_{m=1}^{\infty} \alpha_m \sin \pi m x, \qquad 0 \le x \le 1,$$

then the solution to the diffusion equation is

$$u(x,t) = \sum_{m=1}^{\infty} \alpha_m \mathrm{e}^{-\pi^2 m^2 t} \sin \pi m x, \qquad 0 \le x \le 1, \qquad t \ge 0.$$

The 2-norm of a function g on $[0,1]$, or L^2 norm, is defined as

$$\|g\|_2 = \left(\int_0^1 [g(x)]^2 \, \mathrm{d}x \right)^{1/2}$$

Hence,

$$\begin{aligned} \|u(x,t)\|_2^2 &= \int_0^1 [u(x,t)]^2 \, \mathrm{d}x = \int_0^1 \left(\sum_{m=1}^{\infty} \alpha_m \mathrm{e}^{-\pi^2 m^2 t} \sin \pi m x \right)^2 \mathrm{d}x \\ &= \sum_{m=1}^{\infty} \sum_{j=1}^{\infty} \alpha_m \alpha_j \mathrm{e}^{-\pi^2 (m^2 + j^2) t} \int_0^1 \sin \pi m x \sin \pi j x \, \mathrm{d}x \end{aligned}$$

and since

$$\int_0^1 \sin \pi m x \sin \pi j x \, \mathrm{d}x = \begin{cases} \frac{1}{2}, & m = j, \\ 0, & \text{otherwise} \end{cases}$$

we have

$$\|u(x,t)\|_2^2 = \frac{1}{2} \sum_{m=1}^{\infty} \alpha_m^2 \mathrm{e}^{-2\pi^2 m^2 t} \le \frac{1}{2} \sum_{m=1}^{\infty} \alpha_m^2 = \|u(x,0)\|_2^2.$$

Suppose that $\tilde{u}(x,t)$ is the solution to the diffusion equation with zero boundary conditions and a slightly perturbed initial condition, i.e.,

$$\tilde{u}(x,0) = \tilde{f}(x) = \sum_{m=1}^{\infty} \tilde{\alpha}_m \sin \pi m x, \qquad 0 \le x \le 1,$$

where

$$\|\tilde{u}(x,0) - u(x,0)\|_2^2 = \|\tilde{f}(x) - f(x)\|_2^2 = \frac{1}{2} \sum_{m=1}^{\infty} (\tilde{\alpha}_m - \alpha_m)^2 \le \epsilon^2 \ll 1$$

then

$$\tilde{u}(x,t) = \sum_{m=1}^{\infty} \tilde{\alpha}_m \mathrm{e}^{-\pi^2 m^2 t} \sin \pi m x, \qquad 0 \le x \le 1, \qquad t \ge 0$$

and therefore

$$\|\tilde{u}(x,t) - u(x,t)\|_2^2 \leq \|\tilde{u}(x,0) - u(x,0)\|_2^2 = \epsilon^2.$$

This shows that if $\|\tilde{u}(x,0) - u(x,0)\|_2$ is small, then $\|\tilde{u}(x,t) - u(x,t)\|_2$ is also small, which shows that the solution depends continuously on the initial data, i.e., the diffusion equation with zero Dirichlet boundary conditions is well-posed.

As an example of an ill-posed problem, consider the reversed-time diffusion equation:

$$\frac{\partial u}{\partial t} = -\frac{\partial^2 u}{\partial x^2}, \qquad x \in (0,1), \qquad t \in (0,T],$$

subject to zero Dirichlet boundary conditions. If

$$u(x,0) = f(x) = \sum_{m=1}^{\infty} \alpha_m \sin \pi m x, \qquad 0 \leq x \leq 1,$$

then

$$u(x,t) = \sum_{m=1}^{\infty} \alpha_m \mathrm{e}^{\pi^2 m^2 t} \sin \pi m x, \qquad 0 \leq x \leq 1, \qquad t \geq 0.$$

Now suppose we have initial data such that

$$\alpha_m, \tilde{\alpha}_m = 0, \qquad m > N$$

and

$$\tilde{\alpha}_m - \alpha_m = \epsilon \sqrt{\frac{2}{N}}$$

so that

$$\|\tilde{u}(x,0) - u(x,0)\|_2^2 = \frac{1}{2} \sum_{m=1}^{N} (\tilde{\alpha}_m - \alpha_m)^2 = \epsilon^2$$

then for $t > 0$

$$\|\tilde{u}(x,t) - u(x,t)\|_2^2 = \frac{1}{2} \sum_{m=1}^{N} (\tilde{\alpha}_m - \alpha_m)^2 \, \mathrm{e}^{2\pi^2 m^2 t} = \frac{\epsilon^2}{N} \sum_{m=1}^{N} \mathrm{e}^{2\pi^2 m^2 t} \to \infty, \qquad N \to \infty.$$

This shows that for an arbitrarily small perturbation of the initial data ($\|\tilde{u}(x,0) - u(x,0)\|_2 \leq \epsilon$) and for any $t > 0$, the difference between the solutions u and $\tilde{u}$ at time t can be made arbitrarily large by taking N large enough. This shows that reversed-time diffusion equation is ill-posed because the solution does not depend continuously on the initial data. In physics, the ill-posedness of the reversed-time diffusion equation is related to the impossibility of deducing the thermal history of an object from it present temperature distribution, which is studied in thermodynamics.

1.0.6 Stability of numerical methods for PDEs

A method is **stable** if, as $h \to 0$ and with μ constant,

$$|u_j^i| < \infty, \qquad j = 0, \ldots, n_x + 1, \qquad i = 0, \ldots, n_t.$$

The "Fundamental theorem of numerical analysis of differential equations" relates consistency, stability and convergence.

Theorem (Lax equivalence theorem) If a PDE problem is well posed, then a numerical method is convergent if and only if it is stable and consistent.

The von Neumann method for stability analysis The von Neumann method for the stability analysis of numerical methods for PDEs is applicable to linear PDEs and in its simplest form, it ignores boundary conditions. The method is essentially a discretised Fourier analysis. The method proceeds as follows: given a numerical method for a PDE (e.g., a finite difference method), use the ansatz

$$u_j^i = \lambda^i \mathrm{e}^{\mathrm{i}kx_j},$$

where $k \in \mathbb{Z}$, $x_j = jh$, $j \in \mathbb{Z}$ and $h > 0$.

Note: For the symbol u_j^i, i is an index (a superscript), whereas λ^i means λ raised to the i-th power, where i is a non-negative integer.

Then for stability, we require

$$|\lambda| \leq 1, \qquad k \in \mathbb{Z}, \qquad h > 0$$

because

$$|u_j^i| = |\lambda^i \mathrm{e}^{\mathrm{i}kx_j}| = |\lambda^i| = |\lambda|^i.$$

We only need to consider the one wave $u_j^i = \lambda^i \mathrm{e}^{\mathrm{i}kx_j}$ because we assume the PDE and thus also the discretised equations are linear and hence the principle of superposition holds.

Example Here we illustrate the von Neumann method (or von Neumann stability analysis) for Euler's method:

$$u_j^{i+1} = u_j^i + \mu \left(u_{j+1}^i - 2u_j^i + u_{j-1}^i \right).$$

Setting $u_j^i = \lambda^i \mathrm{e}^{\mathrm{i}kx_j}$, it follows that

$$\begin{aligned} \lambda &= 1 + \mu \left(\mathrm{e}^{\mathrm{i}kh} - 2 + \mathrm{e}^{-\mathrm{i}kh} \right) \\ &= 1 + \mu \left(\mathrm{e}^{\mathrm{i}kh/2} - \mathrm{e}^{-\mathrm{i}kh/2} \right)^2 \\ &= 1 - 4\mu \sin^2(kh/2). \end{aligned}$$

For stability, we require

$$|\lambda| \leq 1 \qquad \Leftrightarrow \qquad -1 \leq 1 - 4\mu \sin^2(kh/2) \leq 1, \qquad h > 0, \qquad k \in \mathbb{Z}.$$

Since $0 \leq \sin^2(kh/2) \leq 1$ and $\mu > 0$, we conclude that Euler's method is stable if and only if $\mu \leq \frac{1}{2}$.

Earlier we proved from first principles that if $\mu \leq \frac{1}{2}$, then Euler's method converges. An alternative proof relies on the Lax equivalence theorem: first we show that Euler's method is consistent (which we've done already), then we show (using the von Neumann method) that the method is stable iff $\mu \leq \frac{1}{2}$ and then we can conclude that the method is convergent if $\mu \leq \frac{1}{2}$.

Stability analysis via matrix analysis We saw earlier that Euler's method can be formulated as

$$\mathbf{u}^{i+1} = A\mathbf{u}^i + \mathbf{k}^i, \qquad i = 0, \ldots, n_t - 1.$$

where

$$\mathbf{u}^i = \begin{bmatrix} u_1^i \\ \vdots \\ u_{n_x}^i \end{bmatrix}, \quad A = \begin{bmatrix} 1-2\mu & \mu & & & \\ \mu & 1-2\mu & \mu & & \\ & \ddots & \ddots & \ddots & \\ & & \mu & 1-2\mu & \mu \\ & & & \mu & 1-2\mu \end{bmatrix} \in \mathbb{R}^{n_x \times n_x}, \quad \mathbf{k}^i = \begin{bmatrix} \mu\varphi_0(t_i) \\ 0 \\ \vdots \\ 0 \\ \mu\varphi_1(t_i) \end{bmatrix} \in \mathbb{R}^{n_x}.$$

A large class of methods for the diffusion equation (and other linear PDEs) can be put into this form. Before we analyse the stability of these methods using matrix methods, we recall some facts from linear algebra.

Recall the definition of the Euclidean inner product: let

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_{n_x} \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_{n_x} \end{bmatrix} \in \mathbb{R}^{n_x}$$

then

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y}$$

If $\mathbf{x}, \mathbf{y} \in \mathbb{C}^{n_x}$, then

$$\langle \mathbf{x}, \mathbf{y} \rangle = \overline{\mathbf{x}}^\top \mathbf{y} = \mathbf{x}^* \mathbf{y},$$

i.e., $\mathbf{x}^*$ denotes the conjugate transpose of $\mathbf{x}$. This inner product induces the Euclidean or ℓ_2 norm:

$$\|\mathbf{x}\| = \left(|x_1|^2 + \cdots + |x_{n_x}|^2 \right)^{1/2} = \left(\langle \mathbf{x}, \mathbf{x} \rangle \right)^{1/2}.$$

We shall also use a scaled Euclidean inner product and ℓ_2 norm:

$$\langle \mathbf{x}, \mathbf{y} \rangle_h := h \langle \mathbf{x}, \mathbf{y} \rangle$$

and hence

$$\|\mathbf{x}\|_h = (h\langle \mathbf{x}, \mathbf{x}\rangle)^{1/2} = \sqrt{h}\,\|\mathbf{x}\|.$$

where $h = \frac{1}{n_x+1}$ and $\mathbf{x}, \mathbf{y} \in \mathbb{C}^{n_x}$ or $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{n_x}$. One reason for using this inner product is the following: suppose the entries of $\mathbf{f} \in \mathbb{R}^{n_x}$ are the function values $f(x_j)$, $x_j = jh$, $j = 1, \ldots, n_x$, then from the Riemann sum definition of the integral of a function, it follows that as $h \to 0$

$$\|\mathbf{f}\|_h = \left(h \sum_{j=1}^{n_x} [f(x_j)]^2 \right)^{1/2} \to \left(\int_0^1 [f(x)]^2 \, \mathrm{d}x \right)^{1/2} = \|f\|_2.$$

i.e., the scaled ℓ_2 vector norm $\|\cdot\|_h$ tends to the L^2 function norm. There is another reason for using the scaled norm $\|\cdot\|_h$: suppose $\mathbf{x} \in \mathbb{R}^{n_x}$ is a vector such that each entry x_j satisfies $x_j = \mathcal{O}(h^p)$, $h \to 0$, i.e., for sufficiently small h (say $0 < h < h_0 \ll 1$), there's a constant $c > 0$ such that $|x_j| \le ch^p$ for $j = 1, \ldots, n_x$, then for sufficiently small h,

$$\|\mathbf{x}\|_h = \left(h \sum_{j=1}^{n_x} x_j^2 \right)^{1/2} \le \left(h n_x c^2 h^{2p} \right)^{1/2} \le ch^p$$

i.e., $\|\mathbf{x}\|_h = \mathcal{O}(h^p)$, as $h \to 0$.

Let $A \in \mathbb{R}^{n_x \times n_x}$, then the matrix 2-norm or matrix Euclidean norm induced by the Euclidean or ℓ_2 vector norm is

$$\|A\| = \max_{\mathbf{x} \in \mathbb{R}^n, \mathbf{x} \neq \mathbf{0}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\|.$$

Since

$$\frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \frac{\sqrt{h}\|A\mathbf{x}\|}{\sqrt{h}\|\mathbf{x}\|} = \frac{\|A\mathbf{x}\|_h}{\|\mathbf{x}\|_h},$$

if follows that

$$\|A\|_h = \|A\|.$$

It follows from the definition of the matrix norm that

$$\|A\mathbf{x}\| \le \|A\|\|\mathbf{x}\|, \qquad \|AB\| \le \|A\|\|B\|$$

and the same is true in the norm $\|\cdot\|_h$.

The eigenvalues or *spectrum* of a square matrix $A \in \mathbb{C}^{n_x \times n_x}$ is

$$\sigma(A) = \{\lambda \in \mathbb{C} : \det(A - \lambda I) = 0\}.$$

The *spectral radius* of a square matrix $A \in \mathbb{C}^{n_x \times n_x}$ is

$$\rho(A) = \max\{|\lambda| : \lambda \in \sigma(A)\}$$

A square matrix $A \in \mathbb{C}^{n_x \times n_x}$ is **normal** if $AA^* = A^*A$. Recall that A^* is the conjugate transpose of A, hence for $A \in \mathbb{R}^{n_x \times n_x}$, $A^* = A^\top$ and thus a real matrix is normal if $AA^\top = A^\top A$.

If a matrix is normal, then

$$\|A\|_h = \|A\| = \rho(A).$$

A numerical method for the diffusion equation on $(x,t) \in [0,1] \times [0,T]$ that takes the form

$$\mathbf{u}^{i+1} = A\mathbf{u}^i + \mathbf{k}^i, \qquad i = 0, \ldots, n_t - 1.$$

where $\mathbf{u}^i \in \mathbb{R}^{n_x}$ and $A \in \mathbb{R}^{n_x \times n_x}$ is **stable** if there exists a $0 < c < \infty$ such that

$$\lim_{h \to 0} \left(\max_{n=0,1,\ldots,n_t} \|A^n\|_h \right) \leq c$$

where, as before, $n_t \tau = T$, $h = 1/(n_x + 1)$, $\mu = \tau/h^2$ and μ is held constant.

Theorem (stability for methods with normal matrices) Suppose A is normal, where $A \in \mathbb{R}^{n_x \times n_x}$ or $A \in \mathbb{C}^{n_x \times n_x}$, then the method

$$\mathbf{u}^{i+1} = A\mathbf{u}^i + \mathbf{k}^i, \qquad i = 0, \ldots, n_t - 1.$$

is stable if there exists a $\nu \geq 0$ such that

$$\rho(A) \leq \mathrm{e}^{\nu\tau}, \qquad h \to 0,$$

where $\tau = \mu h^2$.

Proof For simplicity, let us assume that $A \in \mathbb{R}^{n_x \times n_x}$. Using the definition of the scaled inner product $\langle \cdot, \cdot \rangle_h$, the normalcy of A and the Cauchy-Schwarz inequality, it follows that for a $\mathbf{w} \in \mathbb{R}^{n_x}$, $\mathbf{w} \neq \mathbf{0}$,

$$\begin{aligned}
\|A^n \mathbf{w}\|_h^2 &= \langle A^n \mathbf{w}, A^n \mathbf{w} \rangle_h \\
&= \langle \mathbf{w}, (A^n)^\top A^n \mathbf{w} \rangle_h \\
&= \langle \mathbf{w}, \left(A^\top A\right)^n \mathbf{w} \rangle_h \\
&\leq \|\mathbf{w}\|_h \| \left(A^\top A\right)^n \mathbf{w}\|_h \\
&\leq \|\mathbf{w}\|_h^2 \| \left(A^\top A\right)^n \|_h \\
&\leq \|\mathbf{w}\|_h^2 \|A\|_h^{2n}.
\end{aligned}$$

Therefore, since A is normal and $0 \leq n \leq n_t$,

$$\frac{\|A^n \mathbf{w}\|_h}{\|\mathbf{w}\|_h} \leq \|A\|_h^n = [\rho(A)]^n \leq \mathrm{e}^{\nu n \tau} \leq \mathrm{e}^{\nu n_t \tau} = \mathrm{e}^{\nu T}.$$

Since

$$\|A^n\|_h = \max_{\mathbf{w} \in \mathbb{R}^{n_x}, \mathbf{w} \neq 0} \frac{\|A^n \mathbf{w}\|_h}{\|\mathbf{w}\|_h},$$

the result follows since we've shown that $\|A^n\|_h \leq c$ for $n = 0, \ldots, n_t$ with $c = \mathrm{e}^{\nu T}$ as $h \to 0$. ■

Example (Stability of Euler's method via matrix analysis) For Euler's method,

$$A = \begin{bmatrix} 1-2\mu & \mu & & & \\ \mu & 1-2\mu & \mu & & \\ & \ddots & \ddots & \ddots & \\ & & \mu & 1-2\mu & \mu \\ & & & \mu & 1-2\mu \end{bmatrix} \in \mathbb{R}^{n_x \times n_x}.$$

Since A is symmetric, it is a normal matrix. However, A is not just symmetric, it is a TST matrix (tridiagonal, symmetric and Toeplitz) and the eigenvalues of TST matrices are known explicitly:

Lemma (eigenvalues of TST matrices) The eigenvalues of an $n_x \times n_x$ TST matrix with α on the main diagonal and β on the first super- and sub-diagonal are

$$\lambda_j = \alpha + 2\beta \cos\left(\frac{\pi j}{n_x + 1}\right), \qquad j = 1, \ldots, n_x.$$

Proof See *A First Course in the Numerical Analysis of Differential Equations* by A. Iserles, 2nd Edition, p. 264.

Returning to Euler's method, since $h = 1/(n_x+1)$ and $x_j = jh$, the eigenvalues of the matrix A are

$$\lambda_j = 1 - 2\mu + 2\mu\cos(\pi x_j) = 1 - 2\mu + 2\mu(1 - 2\sin^2(\pi x_j/2)) = 1 - 4\mu\sin^2(\pi x_j/2)$$

for $j = 1, \ldots, n_x$. If $0 < \mu \leq 1/2$, then $\rho(A) \leq 1$ as $h \to 0$ and hence the above theorem holds with $\nu = 0$ and we conclude that Euler's method is stable, whereas if $\mu > 1/2$, then $\rho(A) > 1$ as $h \to 0$ and there isn't a $\nu > 0$ such that the above theorem holds.

```
nx = 11
μ = 0.5
A = SymTridiagonal(fill((1 - 2μ),nx),fill(μ,nx-1))
h = 1/(nx+1)
x = h:h:1-h
[eigvals(A) 1 .- 4μ*sin.(π*x[end:-1:1]/2).^2]
```

```
11×2 Matrix{Float64}:
 -0.965926     -0.965926
 -0.866025     -0.866025
 -0.707107     -0.707107
 -0.5          -0.5
 -0.258819     -0.258819
 -2.68965e-16   2.22045e-16
  0.258819      0.258819
  0.5           0.5
  0.707107      0.707107
  0.866025      0.866025
  0.965926      0.965926
```

1.1 Implicit Euler method

Recall that we arrived at Euler's method for the diffusion equation by combining a (first-order) forward difference approximation to the time derivative with a (second-order) central difference approximation to the second spatial derivative. Suppose now that we approximate the time derivative with a (first-order) *backward* difference approximation

$$u_t(x_j, t_i) \approx \frac{u_j^i - u_j^{i-1}}{\tau}$$

and again use a second-order central difference approximation to u_{xx}, then we arrive at the method

$$\frac{u_j^i - u_j^{i-1}}{\tau} - \frac{u_{j+1}^i - 2u_j^i + u_{j-1}^i}{h^2} = 0$$

or

$$u_j^i = u_j^{i+1} - \mu\left(u_{j+1}^{i+1} - 2u_j^{i+1} + u_{j-1}^{i+1}\right), \qquad j = 1, \ldots, n_x,$$

which is known is the backward Euler method, or the implicit Euler method.

In linear algebra notation, the method becomes

$$B\mathbf{u}^{i+1} = \mathbf{u}^i + \mathbf{k}^{i+1}$$

where

$$\mathbf{u}^i = \begin{bmatrix} u_1^i \\ \vdots \\ u_{n_x}^i \end{bmatrix}, \quad B = \begin{bmatrix} 1+2\mu & -\mu & & & \\ -\mu & 1+2\mu & -\mu & & \\ & \ddots & \ddots & \ddots & \\ & & -\mu & 1+2\mu & -\mu \\ & & & -\mu & 1+2\mu \end{bmatrix} \in \mathbb{R}^{n_x \times n_x}, \quad \mathbf{k}^i = \begin{bmatrix} \mu\varphi_0(t_i) \\ 0 \\ \vdots \\ 0 \\ \mu\varphi_1(t_i) \end{bmatrix} \in \mathbb{R}^{n_x}.$$

As with the (explicit) Euler method, the local truncation error is second order

$$\frac{\tilde{u}_j^i - \tilde{u}_j^{i-1}}{\tau} - \frac{\tilde{u}_{j+1}^i - 2\tilde{u}_j^i + \tilde{u}_{j-1}^i}{h^2} = \mathcal{O}(h^2), \qquad h \to 0,$$

and therefore the method is consistent. There appears to be no reason to prefer the implicit Euler method over the explicit Euler method because it has the same order but it is computationally more expensive. Now, let's consider the stability of the implicit Euler method.

1.1.1 Von Neumann stability analysis

Setting

$$u_j^i = \lambda^i \mathrm{e}^{\mathrm{i}kx_j}$$

in the implicit Euler method, we deduce that

$$\begin{aligned} 1 &= \lambda\left[1-\mu\left(\mathrm{e}^{\mathrm{i}kh}-2+\mathrm{e}^{-\mathrm{i}kh}\right)\right] \\ &= \lambda\left[1-\mu\left(\mathrm{e}^{\mathrm{i}kh/2}-\mathrm{e}^{-\mathrm{i}kh/2}\right)^2\right] \\ &= \lambda\left[1+4\mu\sin^2(kh/2)\right]. \end{aligned}$$

For stability, we require

$$|\lambda| \leq 1 \qquad \Leftrightarrow \qquad 1+4\mu\sin^2(kh/2) \geq 1, \qquad h>0, \qquad k \in \mathbb{Z}.$$

We conclude that the backward Euler method is stable for all $\mu > 0$. We say that the backward Euler method is unconditionally stable. Hence, compared to the explicit Euler method, we can take much larger step sizes while maintaining stability. However, larger step sizes lead to larger errors. The ideal method would be high order and unconditionally stable.

1.1.2 Stability analysis via matrix analysis

The implicit Euler method can be formulated as

$$\mathbf{u}^{i+1} = A\mathbf{u}^i + \mathbf{b}^i$$

where $A = B^{-1}$ and $\mathbf{b}^i = B^{-1}\mathbf{k}^{i+1}$.

The matrix A is normal because

$$AA^\top = B^{-1}B^{-\top} = \left(B^\top B\right)^{-1} = \left(BB^T\right)^{-1} = B^{-\top}B^{-1} = A^\top A.$$

Since B is a TST matrix, its eigenvalues are

$$\lambda_j = 1+2\mu-2\mu\cos(\pi x_j) = 1+2\mu-2\mu(1-2\sin^2(\pi x_j/2)) = 1+4\mu\sin^2(\pi x_j/2).$$

Since $\lambda \in \sigma(B)$ iff $\lambda^{-1} \in \sigma(B^{-1})$, we conclude that for any $\mu > 0$ and $h > 0$

$$\rho(A) = \max_{j=1,\ldots,n_x} \frac{1}{1+4\mu\sin^2(\pi x_j/2)} < 1,$$

and therefore the implicit Euler method is stable for any $\mu > 0$.

```
nx = 11
μ = 0.5
B = SymTridiagonal(fill((1 + 2μ),nx),fill(-μ,nx-1))
h = 1/(nx+1)
x = h:h:1-h
norm(eigvals(B) - (1 .+ 4μ*sin.(π*x/2).^2),Inf)
```

```
1.3322676295501878e-15
```

1.1.3 Numerical experiment

```
function BackwardEuler(f,ϕ0,ϕ1,nx,μ,T)

    x = range(0,1,nx + 2)
    h = 1/(nx+1)
    τ = μ*h^2
    t = 0:τ:T
    nt = length(t)-1
    B = SymTridiagonal(fill((1 + 2μ),nx),fill(-μ,nx-1))
    u = zeros(nt+1,nx+2)
    u[:,1] = ϕ0.(t)
    u[:,nx+2] = ϕ1.(t)
    u[1,2:nx+1] = f.(x[2:nx+1])

    for i = 1:nt
        b = u[i,2:nx+1]
        b[1] += μ*ϕ0(t[i+1])
        b[nx] += μ*ϕ1(t[i+1])
        u[i+1,2:nx+1] = B\b
    end

    u, x, t
end
```

```
BackwardEuler (generic function with 1 method)
```

```
f = x -> sin(π*x/2) + 0.5*sin(2π*x)
ϕ1 = t -> exp(-π^2*t/4)
ϕ0 = t -> 0
nx = 50
μ = 0.50
T = 1
Euler(f,ϕ0,ϕ1,nx,μ,T)
@time Euler(f,ϕ0,ϕ1,nx,μ,T)
BackwardEuler(f,ϕ0,ϕ1,nx,μ,T)
u,x,t = @time BackwardEuler(f,ϕ0,ϕ1,nx,μ,T)
nt = length(t) -1
```

```
0.019761 seconds (10.42 k allocations: 7.067 MiB)
  0.032318 seconds (20.82 k allocations: 11.750 MiB)
5202
```

```
xx = x' .* ones(nt+1)
tt = ones(nx+2)' .* t
error = ue.(xx,tt) - u
e1 = maximum(error)
```

```
0.00091223286366926
```

```
μ = 0.50*nx
u,x,t = @time BackwardEuler(f,ϕ0,ϕ1,nx,μ,T)
nt = length(t) -1
```

```
0.000832 seconds (429 allocations: 242.906 KiB)
104
```

```
xx = x' .* ones(nt+1)
tt = ones(nx+2)' .* t
error = ue.(xx,tt) - u
e1 = maximum(error)
```

0.029907965385205237

1.2 Semi-discretisation / Method of lines

We have seen one approach to designing numerical methods for the diffusion equation: replacing derivatives with finite difference approximations. Another approach, as the name semi-discretisation implies, is to discretise only the spatial variable. For example, let

$$u(x_j, t) \approx v_j(t), \qquad j = 0, \ldots, n_x + 1, \qquad x_j = jh, \qquad h = \frac{1}{n_x + 1},$$

and suppose we approximate u_{xx} with a second-order central difference:

$$u_{xx}(x_j, t) \approx \frac{1}{h^2} \left(v_{j+1} - 2v_j + v_{j-1} \right).$$

The temporal derivative is approximated by setting $u_t(x_j, t) \approx v_j'(t)$, hence we approximate the solution to the diffusion equation by solving

$$v_j'(t) = \frac{1}{h^2} \left(v_{j+1} - 2v_j + v_{j-1} \right), \qquad j = 1, \ldots, n_x.$$

That is, we approximate the solution to the diffusion equation (a linear PDE) by the solution to a linear system of ODEs. This semi-discretisation approach is also known as the *method of lines* because we are approximating the solution by functions $v_j(t)$ on the lines $(x, t) = (x_j, t)$, $t \geq 0$, $j = 1, \ldots, n_x$.

Setting $\tilde{v}_j(t) = u(x_j, t)$ and assuming $\tilde{v}_j(t) = v_j(t)$, it follows that if $u_{xxxx}(x, t)$ is bounded on $(x, t) \in [0, 1] \times [0, T]$, then

$$\tilde{v}_j'(t) = u_t(x_j, t) = \frac{1}{h^2} \left(\tilde{v}_{j+1} - 2\tilde{v}_j + \tilde{v}_{j-1} \right) = u_{xx}(x_j, t) + \mathcal{O}(h^2), \qquad h \to 0,$$

and therefore

$$\tilde{v}_j'(t) - \frac{1}{h^2} \left(\tilde{v}_{j+1} - 2\tilde{v}_j + \tilde{v}_{j-1} \right) = \mathcal{O}(h^2), \qquad h \to 0,$$

hence the semi-discrete method is second order and consistent.

1.2.1 Von Neumann stability analysis of semi-discrete methods

The von Neumann method for the stability analysis of semi-discrete methods is completely analogous to the method for fully discretised methods: we use the ansatz

$$v_j(t) = \lambda(t) \mathrm{e}^{\mathrm{i}kx_j}$$

ignore boundary conditions and since $|v_j| = |\lambda(t)|$, for stability we require that

$$\lim_{h \to 0} |\lambda(t)| \leq c < \infty, \qquad t \in [0, T], \qquad k \in \mathbb{Z}.$$

Example For the semi-discrete method above,

$$v_j'(t) = \frac{1}{h^2}\left(v_{j+1} - 2v_j + v_{j-1}\right),$$

if we set $v_j(t) = \lambda(t)\mathrm{e}^{\mathrm{i}kx_j}$, then we obtain

$$\lambda'(t) = \frac{1}{h^2}\left(\mathrm{e}^{\mathrm{i}kh} - 2 + \left(\mathrm{e}^{-\mathrm{i}kh}\right)\right)\lambda(t) = -\frac{4\sin^2(kh/2)}{h^2}\lambda(t)$$

and therefore

$$\lambda(t) = \mathrm{e}^{-4t\sin^2(kh/2)/h^2}\lambda(0).$$

As $h \to 0$,

$$\frac{4\sin^2(kh/2)}{h^2} = k^2 + \mathcal{O}(h^2).$$

Hence the method is stable because for $k \in \mathbb{Z}$,

$$\lim_{h\to 0}|\lambda(t)| = \mathrm{e}^{-k^2t}|\lambda(0)| \le |\lambda(0)| < \infty, \qquad t \in [0,T].$$

By the Lax equivalence theorem, we conclude the method is convergent because we have showed that it is consistent and stable.

1.2.2 Stability analysis via matrix analysis for semi-discrete methods

All semi-discrete methods for the diffusion equation can be cast in the form

$$\mathbf{v}'(t) = A\mathbf{v}(t) + \mathbf{h}(t), \qquad \mathbf{v}(0) = \mathbf{f},$$

where $\mathbf{v}(t), \mathbf{h}(t) \in \mathbb{R}^{n_x}$ and $A \in \mathbb{R}^{n_x \times n_x}$. For example, the method derived above,

$$v_j'(t) = \frac{1}{h^2}\left(v_{j+1} - 2v_j + v_{j-1}\right), \qquad j = 1, \ldots, n_x,$$

can be expressed as

$$\underbrace{\begin{bmatrix} v_1' \\ \vdots \\ \vdots \\ \vdots \\ v_{n_x}' \end{bmatrix}}_{\mathbf{v}'} = \underbrace{\frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} v_1 \\ \vdots \\ \vdots \\ \vdots \\ v_{n_x} \end{bmatrix}}_{\mathbf{v}} + \underbrace{\frac{1}{h^2}\begin{bmatrix} \varphi_0(t) \\ 0 \\ \vdots \\ 0 \\ \varphi_1(t) \end{bmatrix}}_{\mathbf{h}(t)}$$

subject to the initial condition

$$v_j(0) = u(x_j, 0) = f(x_j), \qquad j = 1, \ldots, n_x.$$

The exact solution to the coupled ODE system $\mathbf{v}'(t) = A\mathbf{v}(t) + \mathbf{h}(t)$ is

$$\mathbf{v}(t) = \mathrm{e}^{tA}\mathbf{v}(0) + \int_0^t \mathrm{e}^{(t-\tau)A}\mathbf{h}(\tau)\,\mathrm{d}\tau$$

where the matrix exponential for a square matrix $B \in \mathbb{R}^{n_x \times n_x}$ or $B \in \mathbb{C}^{n_x \times n_x}$ is defined as

$$\mathrm{e}^B = \sum_{k=0}^{\infty} \frac{B^k}{k!}.$$

Therefore we have that a semi-discrete method for the diffusion equation is **stable** if there exists a constant $0 < c < \infty$ such that

$$\lim_{h \to 0} \left(\max_{t \in [0,T]} \|\mathrm{e}^{tA}\|_h \right) \leq c.$$

Example Let's consider the method above, for which

$$A = \frac{1}{h^2} \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \in \mathbb{R}^{n_x \times n_x}.$$

Since A is a TST matrix, its eigenvalues are

$$\lambda_j = \frac{1}{h^2}\left[-2 + 2\cos\left(\pi x_j\right)\right] = \frac{1}{h^2}\left[-2 + 2\left(1 - 2\sin^2\left(\pi x_j/2\right)\right)\right] = -\frac{4}{h^2}\sin^2(\pi x_j/2), \qquad j = 1, \ldots, n_x.$$

Let's check this:

```
nx = 11
h = 1/(nx+1)
A = SymTridiagonal(fill(-2,nx),fill(1,nx-1))/h^2
x = h:h:1-h
[eigvals(A) -4/h^2*sin.(π*x[end:-1:1]/2).^2]
```

```
11×2 Matrix{Float64}:
 -566.187    -566.187
 -537.415    -537.415
 -491.647    -491.647
 -432.0      -432.0
 -362.54     -362.54
 -288.0      -288.0
 -213.46     -213.46
 -144.0      -144.0
  -84.3532    -84.3532
  -38.5847    -38.5847
   -9.81336    -9.81336
```

To make more sense of the meaning of the matrix exponential we'll need the following.

Lemma (eigenvectors of TST matrices) Let λ_j, $j = 1, \ldots, n_x$ be the eigenvalues of an $n_x \times n_x$ TST matrix with α on the main diagonal and β on the first super- and sub-diagonal. The entries of the corresponding orthogonal eigenvector $\mathbf{q}_j \in \mathbb{R}^{n_x}$ are

$$q_{j,\ell} = \sqrt{\frac{2}{n_x+1}} \sin\left(\frac{\pi j \ell}{n_x+1}\right), \qquad \ell = 1, \ldots, n_x,$$

for $j = 1, \ldots, n_x$.

Proof See *A First Course in the Numerical Analysis of Differential Equations* by A. Iserles, 2nd Edition, p. 264.

Hence, we have

$$A\mathbf{q}_j = \lambda_j \mathbf{q}_j, \qquad j = 1, \ldots, n_x.$$

Let

$$Q = [\mathbf{q}_1 | \mathbf{q}_2 | \cdots | \mathbf{q}_{n_x}] \in \mathbb{R}^{n_x \times n_x},$$

then

$$AQ = Q\Lambda$$

where

$$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_{n_x} \end{bmatrix},$$

and since the vectors $\mathbf{q}_j$ are orthogonal, the matrix Q is orthogonal, i.e., $Q^\top Q = I$. Therefore, the spectral factorisation (eigendecomposition) of A is

$$A = Q\Lambda Q^{-1} = Q\Lambda Q^\top.$$

```
Q = [sqrt(2/(nx+1))*sin(π*j*l/(nx+1)) for l = 1:nx, j = 1:nx]
Q'*Q ≈ I
```

```
true
```

```
Λ = diagm(eigvals(A)[end:-1:1])
Q*Λ*Q' ≈ A
```

```
true
```

We have that

$$
\begin{aligned}
\mathrm{e}^{tA} &= \sum_{k=0}^{\infty} \frac{(tA)^k}{k!} \\
&= \sum_{k=0}^{\infty} \frac{t^k}{k!} Q\Lambda^k Q^\top \\
&= \sum_{k=0}^{\infty} Q \frac{t^k}{k!} \Lambda^k Q^\top \\
&= Q \left(\sum_{k=0}^{\infty} \frac{t^k}{k!} \Lambda^k \right) Q^\top \\
&= Q \begin{bmatrix} \sum_{k=0}^{\infty} \frac{t^k \lambda_1^k}{k!} & & & \\ & \sum_{k=0}^{\infty} \frac{t^k \lambda_2^k}{k!} & & \\ & & \ddots & \\ & & & \sum_{k=0}^{\infty} \frac{t^k \lambda_{n_x}^k}{k!} \end{bmatrix} Q^\top, \\
&= Q \begin{bmatrix} \mathrm{e}^{t\lambda_1} & & & \\ & \mathrm{e}^{t\lambda_2} & & \\ & & \ddots & \\ & & & \mathrm{e}^{t\lambda_{n_x}} \end{bmatrix} Q^\top \\
&= Q \mathrm{e}^{t\Lambda} Q^\top
\end{aligned}
$$

Notice that

$$
\sigma(\mathrm{e}^{tA}) = \left\{ \mathrm{e}^{t\lambda_j} : \lambda_j \in \sigma(A) \right\}
$$

The matrix e^{tA} is symmetric

$$
\left(\mathrm{e}^{tA} \right)^\top = \left(Q \mathrm{e}^{t\Lambda} Q^\top \right)^\top = Q \mathrm{e}^{t\Lambda} Q^\top = \mathrm{e}^{tA}
$$

and therefore it's a normal matrix and thus

$$
\|\mathrm{e}^{tA}\|_h = \|\mathrm{e}^{tA}\| = \rho(\mathrm{e}^{tA}) = \max \left\{ \left| \mathrm{e}^{t\lambda_j} \right| : \lambda_j \in \sigma(A) \right\}.
$$

Since $\lambda_j < 0$ for $j = 1, \ldots, n_x$, it follows that

$$
\lim_{h \to 0} \left(\max_{t \in [0,T]} \|\mathrm{e}^{tA}\|_h \right) \leq 1
$$

and therefore we have proved that the method is stable.

In the above example, we have proven the stability of a method by finding an upper bound on the spectral radius of A, which implied an upper bound on the norm of e^{tA}. This method of proving stability can be generalised to all semi-discrete methods with a normal matrix A because normal matrices are *unitarily diagonalisable*, meaning that if $A \in \mathbb{C}^{n_x \times n_x}$ is a normal matrix then,

$$
A = U \Lambda U^*
$$

where, $U, \Lambda \in \mathbb{C}^{n_x \times n_x}$, Λ is a diagonal matrix containing the eigenvalues of A and U is unitary, i.e., $UU^* = I$.

Theorem (stability of semidiscrete methods for the diffusion equation with normal matrices) The semi-discrete method for the diffusion equation given by

$$\mathbf{v}'(t) = A\mathbf{v}(t) + \mathbf{h}(t), \qquad \mathbf{v}(0) = \mathbf{f},$$

where $\mathbf{v}(t), \mathbf{h}(t) \in \mathbb{R}^{n_x}$ and $A \in \mathbb{R}^{n_x \times n_x}$ is a normal matrix is stable if there exists an $\eta \in \mathbb{R}$ such that as $h \to 0$,

$$\text{Re } \lambda \leq \eta \quad \text{for every} \quad \lambda \in \sigma(A).$$

Proof See *A First Course in the Numerical Analysis of Differential Equations* by A. Iserles, p. 370.

Remark In the example above, the conditions of this theorem hold for $\eta = 0$.

In the example above, we showed that A had the factorisation

$$A = Q\Lambda Q^\top.$$

This can be used to 'diagonalise' the ODE system

$$\mathbf{v}' = A\mathbf{v} + \mathbf{h}(t),$$

because if we set $\mathbf{w} = Q^\top \mathbf{v}$ and $\mathbf{b}(t) = Q^\top \mathbf{h}(t)$, then

$$\mathbf{w}' = \Lambda \mathbf{w} + \mathbf{b}(t),$$

i.e., if we let $w_j(t)$, $b_j(t)$ be the entries of $\mathbf{w}$ and $\mathbf{b}(t)$ for $j = 1, \ldots, n_x$, then

$$w_j' = \lambda_j w_j + b_j(t), \qquad j = 1, \ldots, n_x.$$

Hence, the coupled system of ODEs is reduced to n_x decoupled scalar ODEs which we know how to solve exactly:

$$w_j(t) = \mathrm{e}^{t\lambda_j} w_j(0) + \int_0^t \mathrm{e}^{(t-\tau)\lambda_j} b_j(\tau)\, \mathrm{d}\tau.$$

Suppose for simplicity that $b_j(t) = 0$ and we approximate the solution to

$$w_j' = \lambda_j w_j, \qquad j = 1, \ldots, n_x,$$

with Euler's method, i.e., we set

$$w_j(t_i) \approx w_j^i, \qquad t_i = i\tau, \qquad \tau = \frac{T}{n_t}, \qquad i = 0, \ldots, n_t,$$

and approximate the derivative with a forward difference

$$w_j'(t_i) \approx \frac{w_j^{i+1} - w_j^i}{\tau},$$

then Euler's method for the ODE problems is

$$w_j^{i+1} = (1 + \tau\lambda_j)\, w_j^i = (1 + \tau\lambda_j)^2\, w_j^{i-1} = \cdots = (1 + \tau\lambda_j)^{i+1}\, w_j^0.$$

We require that the w_j^i be bounded as $\tau \to 0$ (or, equivalently, as $n_t \to \infty$), which will occur if and only if

$$|1 + \tau\lambda_j| \leq 1.$$

Recall that for $j = 1, \ldots, n_x$

$$\lambda_j = -\frac{4}{h^2} \sin^2\left(\frac{\pi x_j}{2}\right) = -4(n_x + 1)^2 \sin^2\left(\frac{\pi j}{2(n_x + 1)}\right)$$

i.e., the eigenvalues are real and strictly negative, hence the above condition is equivalent to

$$\tau \leq -\frac{2}{\lambda_j}, \qquad j = 1, \ldots, n_x,$$

that is, we require that

$$\tau \leq \min_{j=1,\ldots,n_x} \left\{-\frac{2}{\lambda_j}\right\} = \frac{2}{\rho(A)} = \frac{1}{2(n_x + 1)^2 \sin^2\left(\frac{\pi n_x}{2(n_x+1)}\right)}$$

This shows that the larger the spectral radius, the smaller the step size τ has to be to ensure that Euler's method is stable.

This observation generalises to a large class of methods: the step size restriction is determined by the spectral radius of the matrix representing the (linearised) system of ODEs. It is also generally true that, compared to explicit methods, implicit methods allow for larger step sizes, as we saw earlier in this chapter when we analysed the stability of the (explicit) Euler method and the implicit Euler method.

1.2.3 Time discretisation of semi-discrete methods

In general, we won't know the exact solution of the semi-discretised system of equations, so we approximate the solution by discretising time. For example, consider consider again the semi-discretised method

$$v_j'(t) = \frac{1}{h^2}\left(v_{j+1} - 2v_j + v_{j-1}\right).$$

If we approximate the derivative with a (first-order) forward difference by letting $v_j(t_i) \approx u_j^i$, $t_i = i\tau$, $\tau = t/n_t$ and

$$v_j'(t_i) \approx \frac{v_j(t_{i+1}) - v_j(t_i)}{\tau} \approx \frac{u_j^{i+1} - u_j^i}{\tau}, \qquad i = 0, \ldots, n_t,$$

then we obtain the fully discretised method

$$u_j^{i+1} = u_j^i + \mu\left(u_{j+1}^i - 2u_j^i + u_{j-1}^i\right), \qquad \mu = \frac{\tau}{h^2},$$

which is just the (explicit) Euler method. Similarly, if we approximate the time derivative with a first order backward difference, then we obtain the backward, or implicit Euler method.

Alternatively, we can solve the semidiscretised system (the system of ODEs) using a 'black box' ODE integrator / solver.

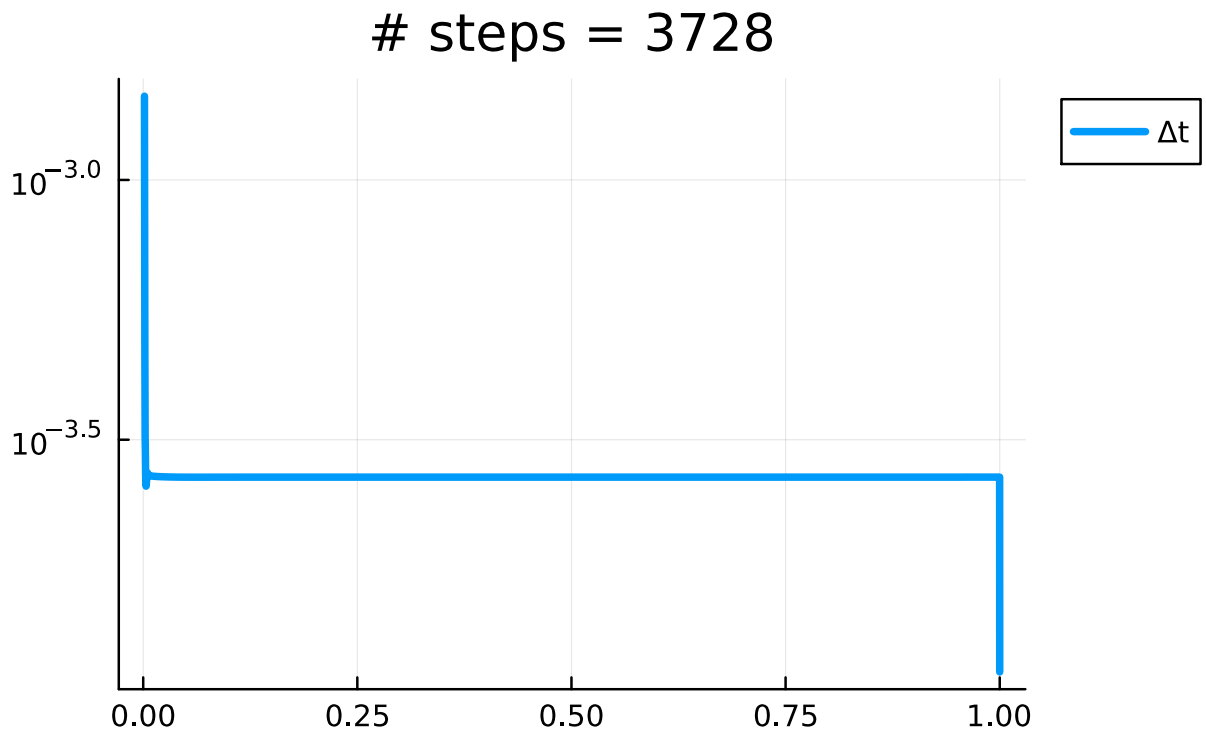
```
nx = 50
h = 1/(nx+1)
f = x -> sin(π*x/2) + 0.5*sin(2π*x)
ϕ1 = t -> exp(-π^2*t/4)
ϕ0 = t -> 0
T = 1
A = SymTridiagonal(fill(-2,nx),fill(1,nx-1))/h^2
F = (v,p,t) -> A*v + [ϕ0(t);zeros(nx-2);ϕ1(t)]/h^2
x = range(0,1,nx + 2)
prob = ODEProblem(F, f.(x[2:end-1]), (0.0, T));
soln =  solve(prob, RK4(),abstol=1e-4);
tv = soln.t
nt = length(tv)
```

```
3728
```

```
errs = zeros(nt-1)
for i = 2:nt
    errs[i-1] = norm(ue.(x[2:nx+1],tv[i]) - soln.u[i],Inf)
end
norm(errs,Inf)
```

```
0.0002743087401239075
```

```
# Let's analyze the numerical solution a bit...
dt = diff(soln.t)
plt1 = plot(soln.t[2:end], dt, yscale = :log10, size = (500, 300), lw = 3, label = "Δt",
     legend = :outertopright, title = "# steps = $(length(soln.t))")
```



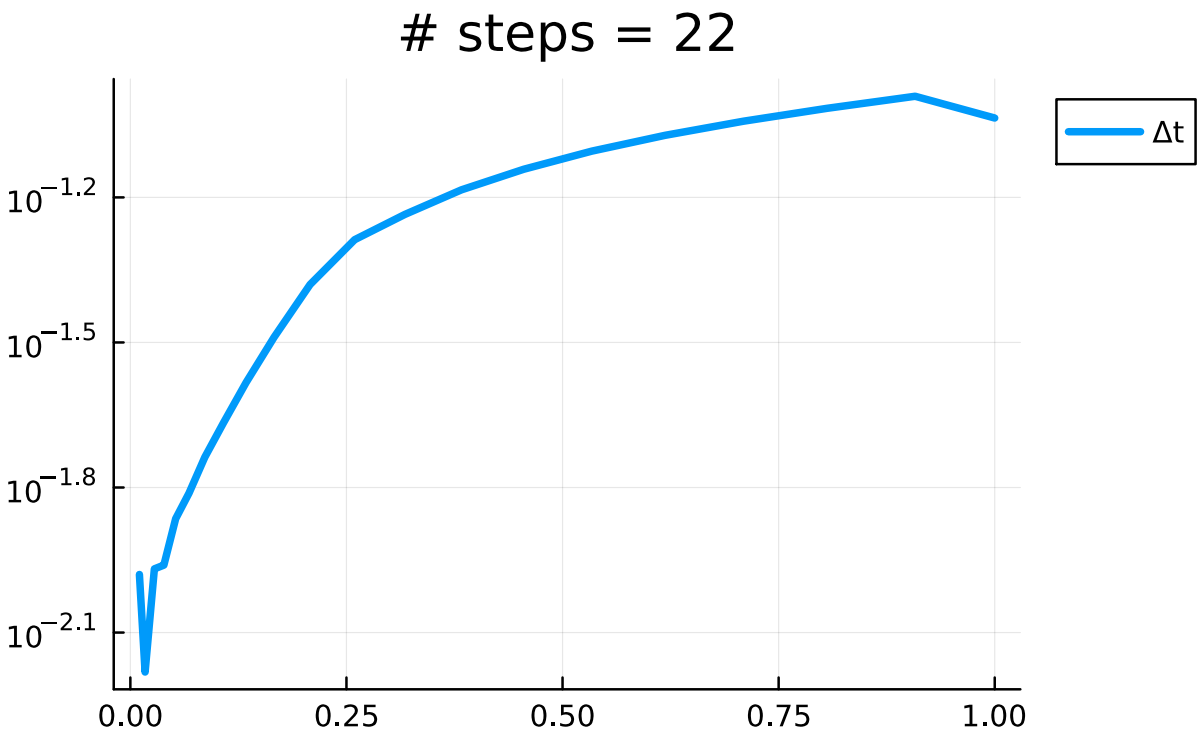
```
soln =  solve(prob, Rodas4(),abstol=1e-4);
tv = soln.t
nt = length(tv)
```

```
22
```

```
errs = zeros(nt-1)
for i = 2:nt
    errs[i-1] = norm(ue.(x[2:nx+1],tv[i]) - soln.u[i],Inf)
end
norm(errs,Inf)
```

```
0.00023693854401607428
```

```
# Let's analyze the numerical solution a bit...
dt = diff(soln.t)
plt1 = plot(soln.t[2:end], dt, yscale = :log10, size = (500, 300), lw = 3, label = "Δt",
     legend = :outertopright, title = "# steps = $(length(soln.t))")
```



1.3 Spectral methods for the diffusion equation

In the previous sections, we used finite differences (studied in Chapter 1) to approximate the solution to the diffusion equation. Here we use the approximation methods we studied in Chapters 2-4 (Fourier methods, Chebyshev methods, orthogonal polynomial methods) to design spectral methods for the diffusion equation.

1.3.1 Fourier spectral method

Let's briefly recall some facts from Chapter 2: the trigonometric interpolant

$$f(s) \approx p_n(s) = \sum_{k=-m}^{m} \tilde{c}_k^n \mathrm{e}^{\mathrm{i}ks}, \qquad n = 2m+1,$$

interpolates f at the following evenly spaced points on the interval $[0, 2\pi)$:

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

and the approximate Fourier coefficients

$$\tilde{c}_k^n = \frac{1}{n} \sum_{j=0}^{n-1} f(s_j) \mathrm{e}^{-\mathrm{i}ks_j}$$

can be computed with the FFT.

Let's return to the diffusion equation $u_t = u_{xx}$ for $(x,t) \in (0,1) \times (0,T]$ subject to the following initial and boundary data:

$$u(x,0) = f(x), \qquad x \in [0,1], \qquad u(0,t) = \varphi_0(t), \qquad u(1,t) = \varphi_1(t).$$

Since $x \in [0,1]$ and $s \in [0, 2\pi]$, the variables are related according to $s = 2\pi x$.

Let's approximate the solution to the diffusion equation for each $t \in [0,T]$ by a trigonometric interpolant in x:

$$u(x,t) \approx u_n(x,t) = \sum_{k=-m}^{m} \tilde{u}_k(t) \mathrm{e}^{2\pi \mathrm{i}kx}.$$

We have that

$$\frac{\partial}{\partial t} u(x,t) \approx \frac{\partial}{\partial t} u_n(x,t) = \sum_{k=-m}^{m} \tilde{u}_k'(t) \mathrm{e}^{2\pi \mathrm{i}kx}$$

and

$$\frac{\partial^2}{\partial x^2} u(x,t) \approx \frac{\partial^2}{\partial x^2} u_n(x,t) = -4\pi^2 \sum_{k=-m}^{m} k^2 \tilde{u}_k(t) \mathrm{e}^{2\pi \mathrm{i}kx}.$$

We require that $u_n(x,t)$ satisfy the diffusion equation, which implies that

$$\tilde{u}_k'(t) = -4\pi^2 k^2 \tilde{u}_k(t), \qquad k = -m, \ldots, m.$$

This is a diagonal system of ODEs, which has the solution

$$\tilde{u}_k(t) = \mathrm{e}^{-4\pi^2 k^2 t} \tilde{u}_k(0), \qquad k = -m, \ldots, m.$$

We can compute $\tilde{u}_k(0)$ with the FFT by requiring that $u_n(x,0)$ interpolates $u(x,0) = f(x)$ at equally spaced points:

$$u(x_j, 0) = f(x_j) = u_n(x_j, 0) = \sum_{k=-m}^{m} \tilde{u}_k(0) \mathrm{e}^{2\pi \mathrm{i}x_j}, \qquad x_j = \frac{j}{n}, \qquad j = 0, \ldots, n-1.$$

We also require that the boundary conditions be satisfied:

$$u(0,t) = \varphi_0(t) = u_n(0,t) = \sum_{k=-m}^{m} \tilde{u}_k(t) = \sum_{k=-m}^{m} \mathrm{e}^{-4\pi^2 k^2 t} \tilde{u}_k(0)$$

and

$$u(1,t) = \varphi_1(t) = u_n(1,t) = \sum_{k=-m}^{m} \tilde{u}_k(t) = \sum_{k=-m}^{m} \mathrm{e}^{-4\pi^2 k^2 t} \tilde{u}_k(0).$$

These conditions cannot be satsfied unless $u(0,t) = u(1,t)$, i.e., unless periodic boundary conditions are specified, in which case the approximate solution is

$$u_n(x,t) = \sum_{k=-m}^{m} \mathrm{e}^{2\pi \mathrm{i} k x - 4\pi^2 k^2 t} \tilde{u}_k(0)$$

and the exact solution is

$$u(x,t) = \sum_{k=-\infty}^{\infty} \mathrm{e}^{2\pi \mathrm{i} k x - 4\pi^2 k^2 t} u_k(0)$$

where

$$u(x,0) = f(x) = \sum_{k=-\infty}^{\infty} u_k(0) \mathrm{e}^{2\pi \mathrm{i} k x},$$

that is,

$$u_k(0) = \int_0^1 f(x) \mathrm{e}^{-2\pi \mathrm{i} k x} \, \mathrm{d}x.$$

Notice that the Fourier spectral method is a "coefficient space" method because we reformulated the PDE problem as a system of ODEs satisfied by the (approximate) Fourier coefficients of the solution. We'll now consider a "value space" method.

1.3.2 Fourier collocation method (Fourier pseudospectral method)

Let's again recall some facts from Chapter 2: the value space representation of the 2π-periodic trigonometric interpolant is

$$p_n(s) = \sum_{k=0}^{n-1} \ell_0(s - s_k) f(s_k),$$

where n is odd with $n = 2m + 1$, the s_k are evenly spaced points on the interval $[0, 2\pi)$ and

$$\ell_0(s) = \begin{cases} \dfrac{1}{n} \dfrac{\sin((m + 1/2)s)}{\sin(s/2)} & \text{if } s \neq 0, \pm 2\pi, \pm 4\pi, \ldots \\ 1 & \text{otherwise} \end{cases}.$$

Since the basis functions $\ell_0(s - s_k)$ have the periodic delta interpolation property, namely

$$\ell_0(s_j - s_k) = \begin{cases} 1 & \text{if } j = k, k \pm n, k \pm 2n, \ldots \\ 0 & \text{otherwise} \end{cases}$$

it follows that $p_n(s_j) = f(s_j)$, $j = 0, \ldots, n-1$, where the points s_j are evenly spaced on the interval $[0, 2\pi)$.

We can use this interpolant to approximate the solution to the diffusion equation on $(x,t) \in (0,1) \times (0,T]$ as follows:

$$u(x,t) \approx u_n(x,t) = \sum_{k=0}^{n-1} \ell_0(2\pi(x - x_k))v_k(t), \qquad x_k = \frac{k}{n},$$

where, from the periodic delta interpolation property,

$$u(x_j,t) \approx u_n(x_j,t) = v_j(t), \qquad j = 0, \ldots, n-1.$$

We require that $u_n(x,t)$ satisfies the diffusion equation at the points x_j (known as the collocation points). We have that

$$\frac{\partial}{\partial t} u_n(x_j,t) = v_j'(t), \qquad j = 0, \ldots, n-1,$$

and

$$\frac{\partial^2}{\partial x^2} u_n(x_j,t) = 4\pi^2 \sum_{k=0}^{n-1} \ell_0''(2\pi(x_j - x_k))v_k(t).$$

From the initial data, we have that

$$u(x_j,0) = f(x_j) = u_n(x_j,0) = v_j(0), \qquad j = 0, \ldots, n-1,$$

and from the boundary data,

$$u(0,t) = \varphi_0(t) = u_n(0,t) = v_0(t)$$

and because $u_n(x,t)$ is 1-periodic in x,

$$u(1,t) = \varphi_1(t) = u_n(1,t) = v_0(t),$$

which can only be satisfied if

$$u(1,t) = u(0,t)$$

i.e., if periodic boundary conditions are imposed.

Hence a Fourier pseudospectral / Fourier collocation method for the diffusion equation with periodic boundary conditions is

$$\mathbf{v}'(t) = D_2 \mathbf{v}(t)$$

where

$$\mathbf{v}(t) = \begin{bmatrix} v_0(t) \\ \vdots \\ v_{n-1}(t) \end{bmatrix}$$

and the entries of the second-order differentiation matrix D_2 are

$$\begin{aligned}(D_2)_{j,k} &= 4\pi^2 \ell_0''(2\pi(x_j - x_k)) \\ &= \begin{cases} -\frac{\pi^2}{3}(n^2-1) & \text{if } j = k \\ 2\pi^2(-1)^{j-k+1}\dfrac{\cos[\pi(j-k)/n]}{\sin^2[\pi(j-k)/n]} & \text{if } j \neq k \end{cases}\end{aligned}$$

for $j, k = 0, \ldots, n-1$.

Let's check this:

```
n = 29
D2m(n) = [j == k ?
    -π^2/3*(n^2 - 1) :
    2π^2*(-1)^(j-k+1)*cos(π*(j-k)/n)/(sin(π*(j-k)/n))^2
    for j = 0:n-1, k = 0:n-1]
f = x -> exp(sin(2π*x))
df = x -> f(x)*2π*cos(2π*x)
d2f = x -> f(x)*(2π*cos(2π*x))^2 - 4π^2*f(x)*sin(2π*x)
xx = range(0,1,n+1)[1:end-1]
norm(d2f.(xx) - D2m(n)*f.(xx), Inf)
```

```
4.078515303262975e-12
```

The differentiation matrix D_2 is a normal matrix because it is symmetric:

```
D2m(5)
```

```
5×5 Matrix{Float64}:
 -78.9568    46.2221    -6.74372   -6.74372   46.2221
  46.2221   -78.9568    46.2221    -6.74372   -6.74372
  -6.74372   46.2221   -78.9568    46.2221    -6.74372
  -6.74372   -6.74372   46.2221   -78.9568    46.2221
  46.2221    -6.74372   -6.74372   46.2221   -78.9568
```

From the results we proved above, to show that the semi-discrete method $\mathbf{v}'(t) = D_2\mathbf{v}(t)$ is stable, we need to prove that the real part of the eigenvalues of $D_2 \in \mathbb{R}^{n\times n}$ are bounded above as $n \to \infty$. To find the eigenvalues of D_2, we'll use some facts we learned in Chapter 2 to find its eigendecomposition (spectral factorisation) using the FFT.

The matrix D_2 is also circulant and all circulant matrices can be diagonalised with the FFT matrix, as we'll see shortly, which also implies that matrix-vector multiplication with a circulant marix can be computed with the FFT.

Recall that we showed that derivates can be computed with the FFT instead of differentiation matrices:

```
m = (n-1)÷2
norm(d2f.(xx) - ifft(ifftshift(-4π^2*(-m:m).^2) .*fft(f.(xx))), Inf)
```

```
1.5347723092418164e-12
```

We also learned that the FFT is equivalent to a certain matrix-vector multiplication. That is, the FFT applied to a vector $\mathbf{f}$ performs the matrix-vector multiplication $Q_n\mathbf{f}$, where Q_n is an $n \times n$ unitary matrix that was defined in Chapter 2. Here is a reminder of the matrix Q_n:

```
Qm(n) = [exp(-2π*im*(k-1)*j/n) for k = 1:n, j=1:n]/sqrt(n)
n = 5
Qm(5)'*Qm(5) ≈ I
```

```
true
```

We also learned that the $ifftshift$ command is equivalent to multiplication by a permutation matrix P, where

$$P = \begin{pmatrix} & I_m \\ I_{m+1} & \end{pmatrix}.$$

From the results of Chapter 2, it follows that the command $ifft(ifftshift(-(-m:m).^2)fft(f(xx)))$ used above for approximating the second derivative on an equispaced grid is equivalent to multipilication by the matrix

$$Q_n^* P^* \Lambda P Q_n, \qquad \Lambda = -4\pi^2 \mathrm{diag}(m^2, (m-1)^2, \ldots, 0, 1^2, \ldots, m^2),$$

and moreover

$$D_2 = Q_n^* P^* \Lambda P Q_n.$$

Let's check this:

```
n = 11
m = (n-1)÷2
P = sparse(1:n, [m+2:n;1:m+1], fill(1,n))
Q = Qm(n)
D2 = D2m(n)
-4π^2*Q'*P'diagm((-m:m).^2)*P*Q ≈ D2
```

```
true
```

This is the spectral factorisation of D_2 and it shows that its eigenvalues are $-4\pi^2 k^2$ for $k = 0, \ldots, m$. Since the eigenvalues are bounded above by 0, we conclude that the Fourier pseudospectral / Fourier collocation semidiscrete method $\mathbf{v}' = D_2\mathbf{v}$ is stable. If the solution to the diffusion equation is continuous and 1-periodic, then we know from Chapter 2 that the trigonometric interpolant will converge (for a fixed time t) to the exact solution as $n \to \infty$, hence the method is consistent and therefore (by the Lax equivalence theorem) convergent.

If we were to use Euler's method for time stepping, then the step size restriction would be

$$\tau \le \frac{2}{\rho(A)} = \frac{1}{2\pi^2 m^2} = \frac{2}{\pi^2 (n-1)^2}.$$

```
n = 91
m = (n-1)÷2
f = x -> sin(π*x/2) + 0.5*sin(2π*x)
T = 0.01
F = (v,p,t) -> real(ifft(ifftshift(-4π^2*(-m:m).^2) .*fft(v)))
x = range(0,1,n+1)[1:end-1]
prob = ODEProblem(F, f.(x), (0.0, T))
soln =  solve(prob,RK4(),abstol=1e-2);
```

In the code above, we used an explicit fourth-order Runge-Kutta method as time stepper / integrator. Let's compare the maximum step size of the Runge-Kutta method with the step size restriction of Euler's method:

```
[maximum(diff(soln.t)) 2/(π^2*(n-1)^2)]
```

```
1×2 Matrix{Float64}:
 4.35304e-5  2.50176e-5
```

```
t = soln.t
u = soln.u
nt = length(soln.t)
@gif for i = 1:nt
    plot(x, u[1], size = (700, 150), label = "u0")
    plot!(x, u[i], label = "Fourier collocation,t = " * (@sprintf("%.4f", t[i])), legend
= :outertopright)
    plot!(x, ue.(x,t[i]), label = "exact solution")
end
```

```
Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_4vCuBuGsCr.gi
f")
```

Of course, for a problem that doesn't have periodic boundary conditions, Fourier methods won't be accurate. We saw this in Chapter 2 when we approximated a non-periodic function with a trigonometric interpolant and observed the Gibbs phenomenon.

1.3.3 Chebyshev collocation method

Recall from Chapter 3 the formula for the Lagrange interpolating polynomial at the Chebyshev points:

$$p_n(s) = \sum_{k=0}^{n} \ell_k(s) f(s_k), \qquad \ell_k(s) = \prod_{\substack{j=0 \\ j \neq k}}^{n} \frac{s - s_j}{s_k - s_j}$$

where the Chebyshev points are

$$s_k = \cos\left(\frac{\pi k}{n}\right), \qquad k = 0, \ldots, n.$$

Since we let $s \in [-1, 1]$ and in the diffusion equation we let $x \in [0, 1]$, the variables are related according to

$$s = 2x - 1,$$

and we set

$$x_k = \frac{s_k + 1}{2}, \qquad k = 0, \ldots, n.$$

We approximate the solution to the diffusion equation using an interpolant at the Chebyshev points:

$$u(x, t) \approx u_n(x, t) = \sum_{k=0}^{n} \ell_k(2x - 1) v_k(t)$$

The Lagrange basis polynomials $\ell_k(s)$ have the delta interpolation property,

$$\ell_k(s_j) = \delta_{j,k} = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{otherwise} \end{cases}$$

and thus for $j = 0, \ldots, n$

$$u(x_j, t) \approx u_n(x_j, t) = \sum_{k=0}^{n} \ell_k(2x_j - 1)v_k(t) = \sum_{k=0}^{n} \ell_k(s_j)v_k(t) = v_j(t).$$

The boundary data imply that

$$u(0, t) = \varphi_0(t) = u_n(0, t) = v_n(t)$$

and

$$u(1, t) = \varphi_1(t) = u_n(1, t) = v_0(t)$$

while the initial data imply

$$u(x_j, 0) = u_n(x_j, 0) = f(x_j) = v_j(0), \qquad j = 0, \ldots, n.$$

We approximate the solution to the diffusion equation by requiring that $u_n(x, t)$ satisfy the diffusion equation at the Chebyshev points (the collocation points). Since

$$\frac{\partial}{\partial t}u(x_j, t) \approx \frac{\partial}{\partial t}u_n(x_j, t) = v_j'(t)$$

and

$$\frac{\partial^2}{\partial x^2}u(x_j, t) \approx \frac{\partial^2}{\partial x^2}u_n(x_j, t) = 4\sum_{k=0}^{n} \ell_k''(2x_j - 1)v_k(t) = 4\sum_{k=0}^{n} \ell_k''(s_j)v_k(t)$$

we approximate the solution to the diffusion equation as follows

$$\begin{aligned}
\begin{pmatrix} v_1'(t) \\ v_2'(t) \\ \vdots \\ v_{n-1}'(t) \end{pmatrix} &= 4 \begin{pmatrix} \ell_0''(s_1) & \ell_1''(s_1) & \cdots & \ell_{n-1}''(s_1) & \ell_n''(s_1) \\ \ell_0''(s_2) & \ell_1''(s_2) & \cdots & \ell_{n-1}''(s_2) & \ell_n''(s_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \ell_0''(s_{n-1}) & \ell_1''(s_{n-1}) & \cdots & \ell_{n-1}''(s_{n-1}) & \ell_n''(s_{n-1}) \end{pmatrix} \begin{pmatrix} v_0(t) \\ v_1(t) \\ \vdots \\ v_{n-1}(t) \\ v_n(t) \end{pmatrix} \\
&= 4 \begin{pmatrix} \ell_1''(s_1) & \cdots & \ell_{n-1}''(s_1) \\ \ell_1''(s_2) & \cdots & \ell_{n-1}''(s_2) \\ \vdots & \vdots & \vdots \\ \ell_1''(s_{n-1}) & \cdots & \ell_{n-1}''(s_{n-1}) \end{pmatrix} \begin{pmatrix} v_1(t) \\ \vdots \\ v_{n-1}(t) \end{pmatrix} \\
&\quad + 4\varphi_1(t) \begin{pmatrix} \ell_0''(s_1) \\ \ell_0''(s_2) \\ \vdots \\ \ell_0''(s_{n-1}) \end{pmatrix} + 4\varphi_0(t) \begin{pmatrix} \ell_n''(s_1) \\ \ell_n''(s_2) \\ \vdots \\ \ell_n''(s_{n-1}) \end{pmatrix} \\
&:= A\mathbf{v}(t) + \mathbf{h}(t).
\end{aligned}$$

Spectral factorisation of the second-order Chebyshev differentiation matrix Recall from Chapter 3 that the following function builds the (first-order) Chebyshev differentiation matrix, i.e., the $(n+1) \times (n+1)$ matrix given by

$$D_1 = \begin{pmatrix} \ell_0'(s_1) & \ell_1'(s_1) & \cdots & \ell_n'(s_1) \\ \ell_0'(s_2) & \ell_1'(s_2) & \cdots & \ell_n'(s_2) \\ \vdots & \vdots & \vdots & \vdots \\ \ell_0'(s_{n-1}) & \ell_1'(s_{n-1}) & \cdots & \ell_n'(s_{n-1}) \end{pmatrix}$$

```
function cheb(n)

    x = cos.(π*(0:n)/n)
    c = [2; ones(n-1,1); 2].*(-1).^(0:n)
    X = repeat(x,1,n+1)
    dX = X-X'
    D = (c*(1 ./c)')./( dX + I ) # off-diagonal entries
    D = D - diagm(vec(sum(D';dims=1)))# diagonal entries

    D, x

end
```

```
cheb (generic function with 1 method)
```

We can obtain the second-order differentiation matrix by squaring the first order differentiation matrix. Let's consider the matrix A defined above, which consists of the second to n-th rows and columns of the second order differentiation matrix:

```
n = 7
D,~ = cheb(n)
D2 = D^2
A = D2[2:n,2:n]
```

```
6×6 Matrix{Float64}:
 -113.208     43.2236   -11.3993     5.84434   -4.0         3.27193
   22.2999   -28.8518    14.9835    -4.0        2.10419    -1.52969
   -4.0       11.8558   -17.9404    10.6239    -3.07106     1.79288
    1.79288   -3.07106   10.6239   -17.9404    11.8558     -4.0
   -1.52969    2.10419   -4.0       14.9835   -28.8518     22.2999
    3.27193   -4.0        5.84434  -11.3993    43.2236   -113.208
```

Unfortunately, A is not a normal matrix:

```
A*A' ≈ A'*A
```

```
false
```

However, A has a spectral factorisation, i.e., there exists a nonsingular matrix $W \in \mathbb{R}^{(n-1)\times(n-1)}$ and a diagonal matrix $\Lambda \in \mathbb{R}^{(n-1)\times(n-1)}$ such that

$$A = W\Lambda W^{-1}.$$

Let's see an example:

```
fact = eigen(A)
Λ = diagm(fact.values)
```

```
6×6 Matrix{Float64}:
 -130.298     0.0      0.0      0.0      0.0       0.0
    0.0    -119.139    0.0      0.0      0.0       0.0
    0.0       0.0    -35.833    0.0      0.0       0.0
    0.0       0.0      0.0    -22.3933   0.0       0.0
    0.0       0.0      0.0      0.0     -9.86945   0.0
    0.0       0.0      0.0      0.0      0.0      -2.46741
```

```
W = fact.vectors
@show A ≈ W*Λ/W
@show A ≈  W*Λ*inv(W);
```

```
A ≈ (W * Λ) / W = true
A ≈ W * Λ * inv(W) = true
```

To prove the stability of the method, we need to show that

$$\lim_{n\to\infty} \|\mathrm{e}^{tA}\| < \infty, \qquad t \in [0, T].$$

We have that

$$\mathrm{e}^{tA} = W\mathrm{e}^{t\Lambda}W^{-1}$$

and thus, since we assume the spectrum / eigenvalues of A are real,

$$\|\mathrm{e}^{tA}\| \leq \|W\|\|W^{-1}\|\|\mathrm{e}^{t\Lambda}\| = \kappa(W)\max\{\mathrm{e}^{t\lambda} : \lambda \in \sigma(A)\}$$

where

$$\kappa(W) = \|W\|\|W^{-1}\|$$

is known as the (2-norm) spectral condition number of A.

If A were a normal matrix, then the spectral factorisation would be

$$A = U\Lambda U^*$$

where U is unitary and thus

$$\mathrm{e}^{tA} = U\mathrm{e}^{t\Lambda}U^*.$$

We have that

$$\kappa(U) = \|U\|\|U^{-1}\| = \|U\|\|U^*\| = 1$$

because

$$\|U\mathbf{x}\|^2 = \langle U\mathbf{x}, U\mathbf{x}\rangle = (U\mathbf{x})^* U\mathbf{x} = \mathbf{x}^*U^*U\mathbf{x} = \mathbf{x}^*\mathbf{x} = \|\mathbf{x}\|^2,$$

it follows from the definition of the matrix 2-norm that $\|U\| = 1$ and similarly it follows that $\|U^*\| = 1$. This shows that for normal matrices, the spectral condition number is one. This is not true of non-normal matrices and therefore we need to take into account how the spectral condition number grows as the dimension of the matrix grows.

Let's return to the non-normal matrix A (consisting of the second to n-th columns of the second order Chebyshev differentiation matrix) and investigate the eigenvalues of A and the spectral condition number $\kappa(W)$ and see what happens when n becomes large.

First we consider a continuous eigenvalue problem and then show that the eigenvalues of the matrix A approximate those of the continuous problem. The solutions to the following ODE eigenvalue problem

$$u''(s) = \lambda u(s), \qquad s \in (-1,1), \qquad u(-1) = u(1) = 0,$$

are

$$u(s) = \sin\left(\frac{n\pi(s+1)}{2}\right), \qquad \lambda = -\frac{\pi^2 n^2}{4}, \qquad n = 1, 2, \ldots$$

Suppose now that we approximate the solutions to the problem with an interpolant at the Chebyshev points, i.e., we set

$$u(s) \approx u_n(s) = \sum_{k=0}^{n} \ell_k(s) w_k, \qquad \ell_k(s) = \prod_{\substack{j=0 \\ j \neq k}}^{n} \frac{s - s_j}{s_k - s_j}$$

where the Chebyshev points are

$$s_k = \cos\left(\frac{\pi k}{n}\right), \qquad k = 0, \ldots, n,$$

and $w_k \approx u(s_k)$. We set

$$u_n(-1) = w_n = u(-1) = 0$$

and

$$u_n(1) = w_0 = u(1) = 0$$

and specify that $u_n(s)$ satisfies the ODE at the collocation points, i.e., we require

$$u_n''(s_j) = \lambda u_n(s_j), \qquad j = 1, \ldots, n-1.$$

Since $u_n(s_j) = w_j$, these equations are equivalent to

$$\begin{pmatrix} \ell_0''(s_1) & \ell_1''(s_1) & \cdots & \ell_{n-1}''(s_1) & \ell_n''(s_1) \\ \ell_0''(s_2) & \ell_1''(s_2) & \cdots & \ell_{n-1}''(s_2) & \ell_n''(s_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \ell_0''(s_{n-1}) & \ell_1''(s_{n-1}) & \cdots & \ell_{n-1}''(s_{n-1}) & \ell_n''(s_{n-1}) \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{n-1} \\ w_n \end{pmatrix} = \lambda \begin{pmatrix} w_1 \\ \vdots \\ w_{n-1} \end{pmatrix}.$$

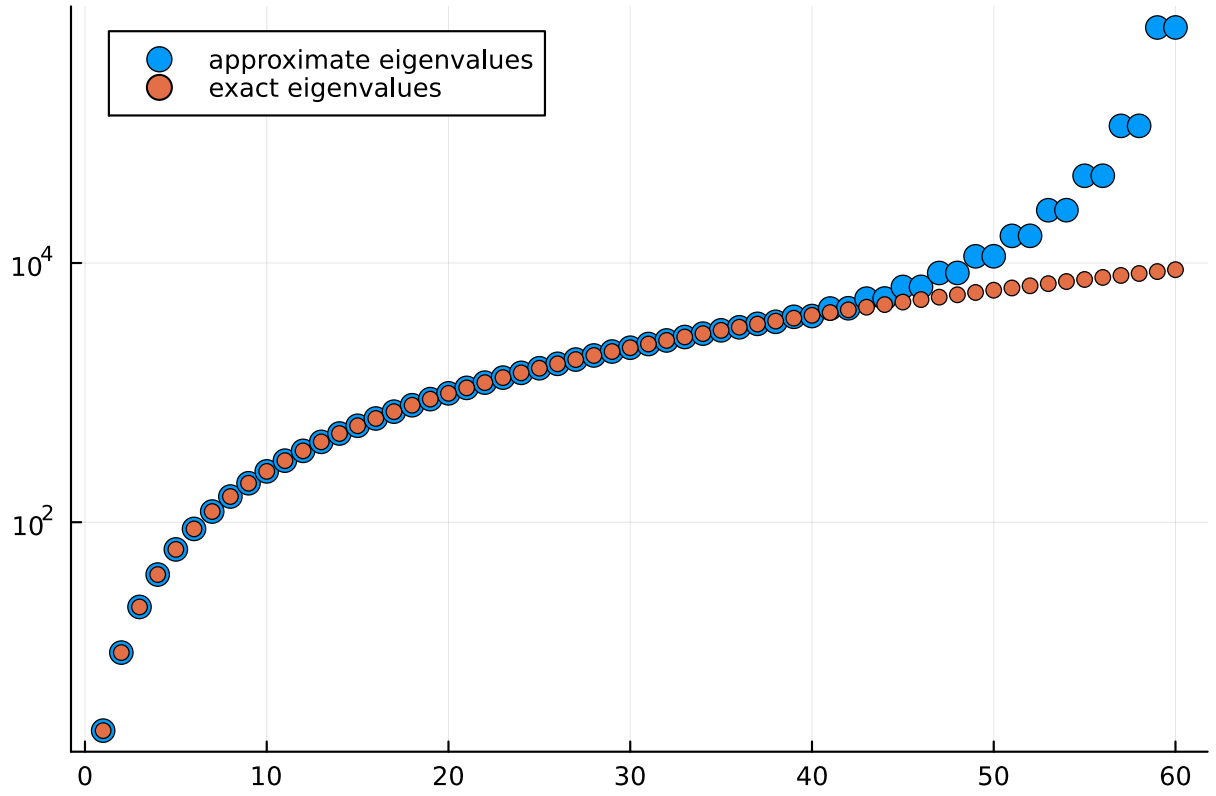
Since $w_0 = w_n = 0$, this can be reduced to

$$\begin{pmatrix} \ell_1''(s_1) & \cdots & \ell_{n-1}''(s_1) \\ \ell_1''(s_2) & \cdots & \ell_{n-1}''(s_2) \\ \vdots & \vdots & \vdots \\ \ell_1''(s_{n-1}) & \cdots & \ell_{n-1}''(s_{n-1}) \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_{n-1} \end{pmatrix} = \lambda \begin{pmatrix} w_1 \\ \vdots \\ w_{n-1} \end{pmatrix},$$

i.e., we have the matrix eigenvalue problem $A\mathbf{w} = \lambda\mathbf{w}$, where A is the matrix we encountered above when we applied the Chebyshev collocation method to the diffusion equation. We'll see that A has $n-1$ distinct eigenvalues and eigenvectors, say $A\mathbf{w}_j = \lambda_j \mathbf{w}_j$, $j = 1, \ldots, n-1$ and λ_j approximates the j-th eigenvalue of the ODE eigenvalue problem (namely, $-\pi^2 j^2/4$) and the eigenvector $\mathbf{w}_j$ approximates the j-th eigenfunction of the ODE eigenvalue problem (namely, $\sin(\pi n(s+1)/2)$ at the Chebyshev points. The eigenvectors $\mathbf{w}_j$ form the columns of the matrix W in the eigendecomposition of A, i.e., $A = W\Lambda W^{-1}$.

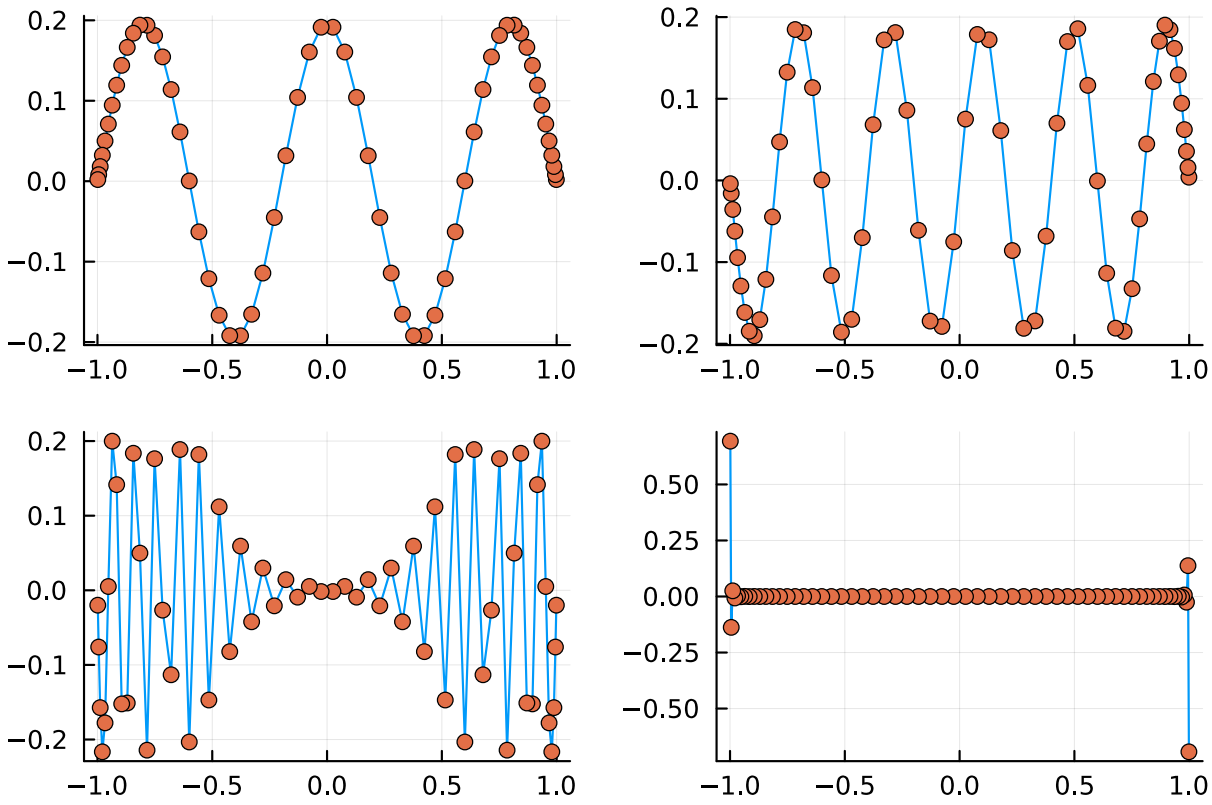
Let's compare the eigenvalues of A to those of the ODE eigenvalue problem:

```
n = 61
D,s = cheb(n)
D2 = D^2
A = D2[2:n,2:n]
λ = eigvals(A)
scatter(1:n-1,abs.(λ[end:-1:1]);yscale=:log10,ms=6,
label="approximate eigenvalues",legend=:topleft)
scatter!(1:n-1,π^2*(1:n-1).^2/4,label="exact eigenvalues")
```



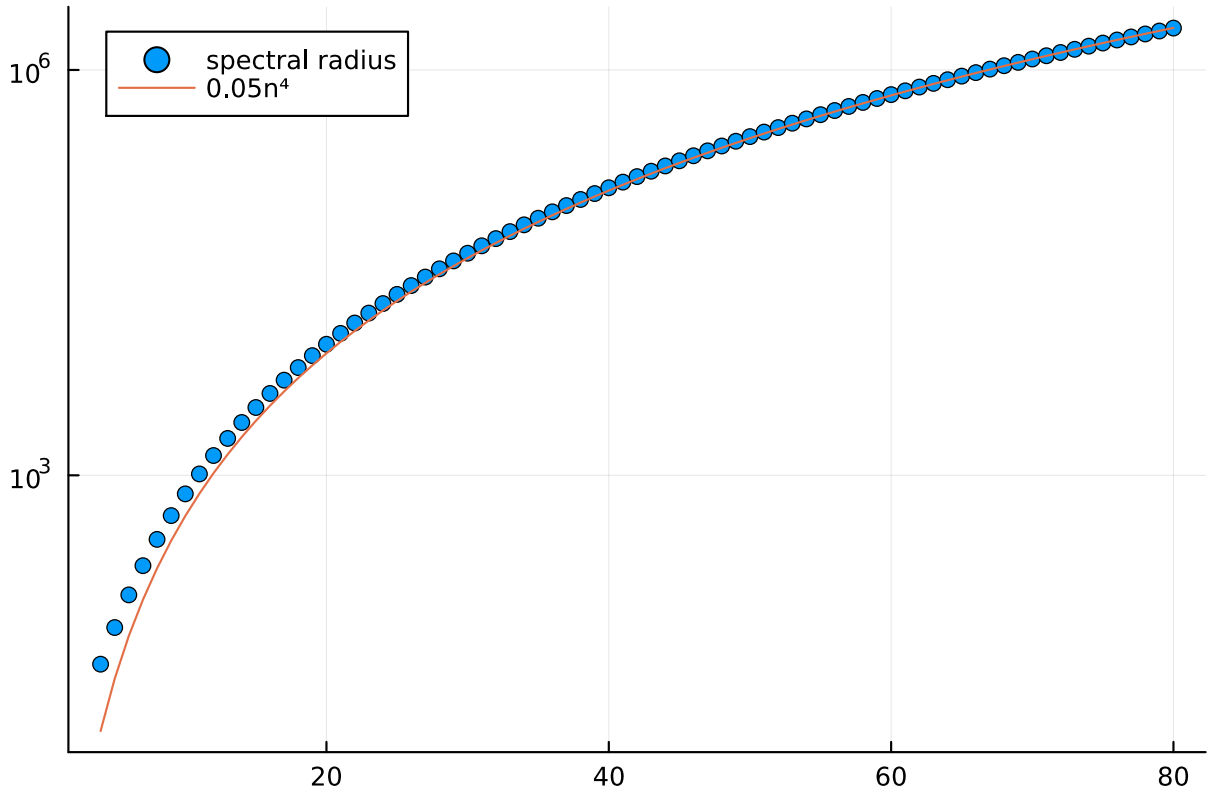
Here are some of the eigenvectors of A:

```
fact = eigen(A)
W = fact.vectors
p1 = plot(s[2:n],W[:,n-5];legend=false)
scatter!(s[2:n],W[:,n-5])
p2 = plot(s[2:n],W[:,n-10];legend=false)
scatter!(s[2:n],W[:,n-10])
p3 = plot(s[2:n],W[:,20];legend=false)
scatter!(s[2:n],W[:,20])
p4 = plot(s[2:n],W[:,1];legend=false)
scatter!(s[2:n],W[:,1])
plot(p1,p2,p3,p4)
```



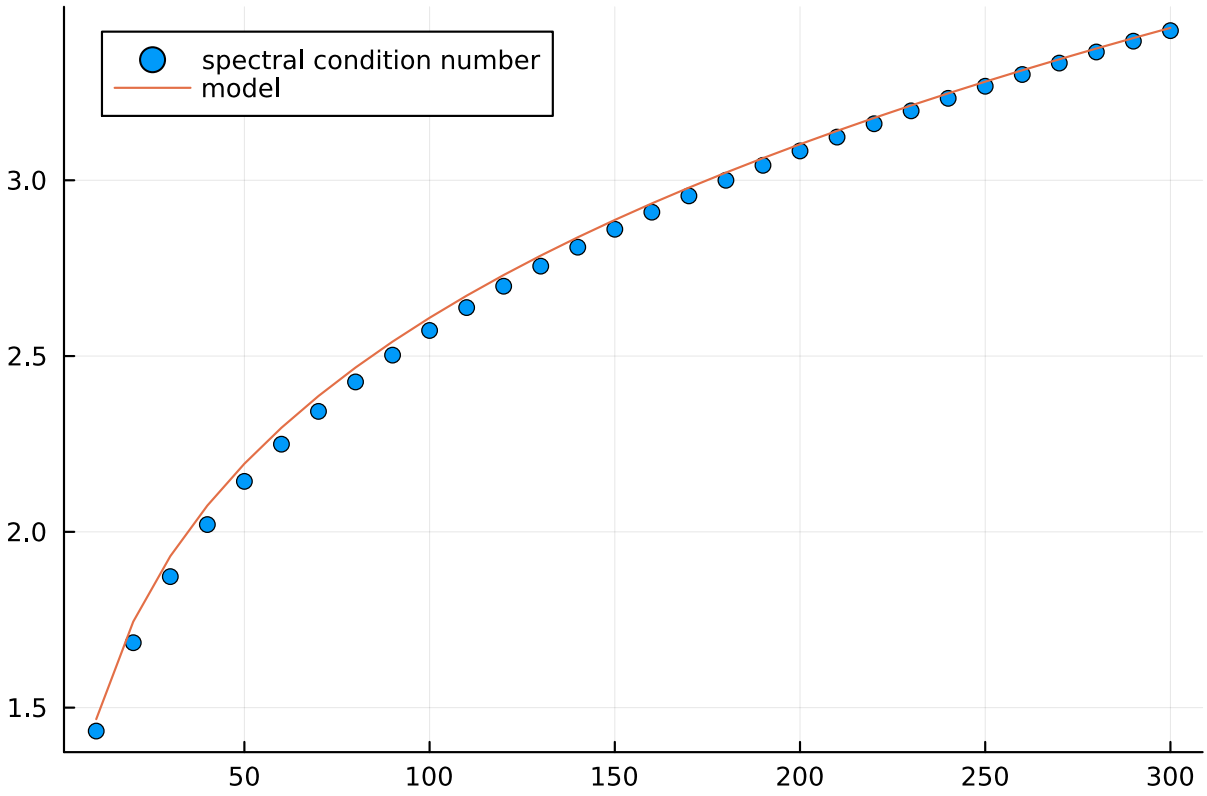
Let's investigate how the spectral radius grows with n:

```
nmax = 81; nvc = 5:nmax
ρc = [( (D,~) = cheb(n); D2 = D^2; A = D2[2:n,2:n]; λ = eigvals(A);
    norm(λ,Inf)) for n = nvc];

scatter(nvc.-1,ρc;yscale=:log10,legend=:topleft,label="spectral radius")
plot!(nvc.-1,0.05*(nvc.-1).^4;label="0.05n4")
```



Here's how the spectral condition number $\kappa(W)$ grows with n:

```
nmax = 300
nv = 10:10:nmax
κ = [( (D,~) = cheb(n); D2 = D^2; A = D2[2:n,2:n]; fact = eigen(A);
        cond(fact.vectors)) for n = nv];

scatter(nv,κ,label="spectral condition number")
plot!(nv,0.825*nv.^(1/4);label="model")
```



Returning to the bound we derived above,

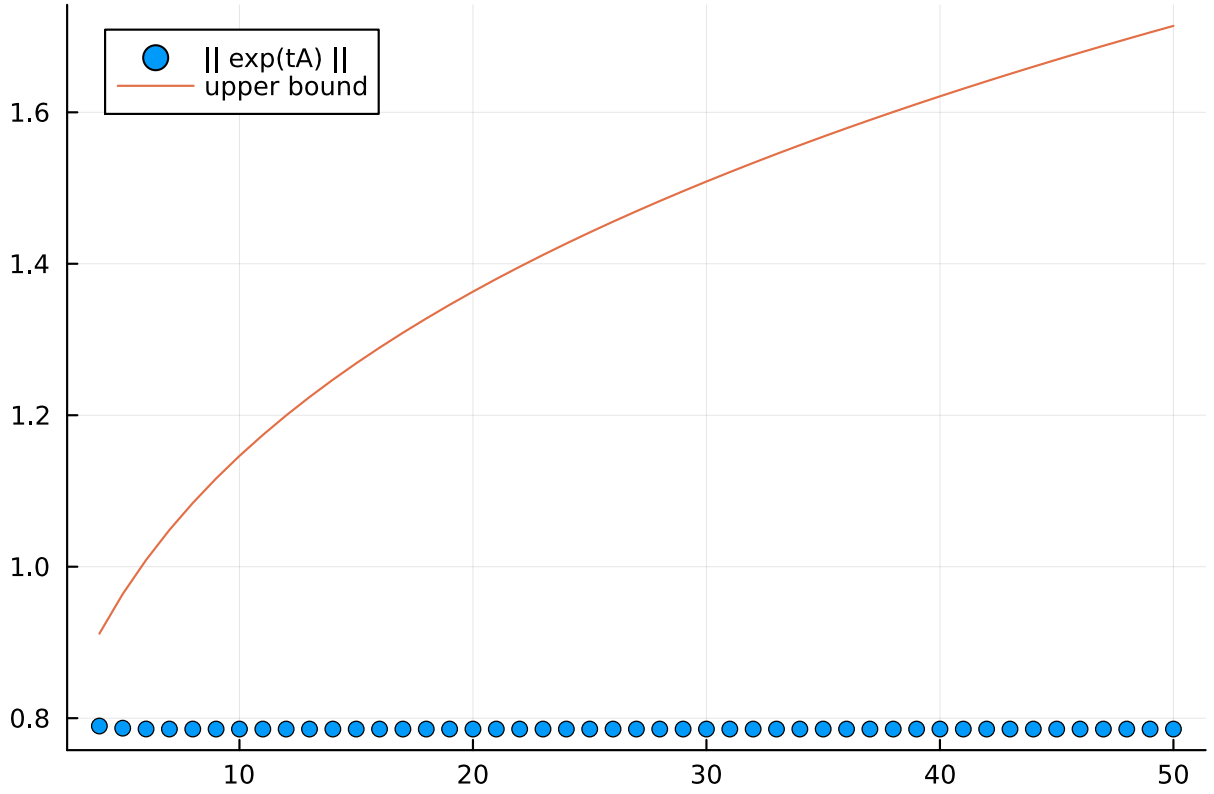
$$\|\mathrm{e}^{tA}\| \leq \|W\|\|W^{-1}\|\|\mathrm{e}^{t\Lambda}\| = \kappa(W)\max\{\mathrm{e}^{t\lambda} : \lambda \in \sigma(A)\},$$

from our numerical results we have the following estimate for an upper bound on $\|\mathrm{e}^{tA}\|$, namely,

$$0.825n^{1/4}\mathrm{e}^{-\pi^2 t/4}.$$

It turns out this bound is completely too pessimistic:

```
t = 0.1
nmax = 50
nv = 4:nmax
mexpb = [( (D,~) = cheb(n); D2 = D^2; A = D2[2:n,2:n];
           opnorm(exp(t*A),2))  for n = nv]
scatter(nv,mexpb;label="|| exp(tA) ||")
plot!(nv,0.825*nv.^(1/4)*exp(-π^2*t/4);label="upper bound")
```



Let's return to the linear ODE system we derived for the diffusion equation with the Chebyshev collocation method:

$$\mathbf{v}'(t) = A\mathbf{v}(t) + \mathbf{h}(t), \qquad \mathbf{v}(t) \in \mathbb{R}^{n-1}, \qquad A \in \mathbb{R}^{(n-1)\times(n-1)}.$$

Earlier in this chapter, we showed that if A has an eigendecomposition, then this system can be diagonalised (i.e., the coupled system of ODEs can be expressed as $n-1$ decoupled scalar ODEs) and we derived the following step size restriction for Euler's method to be stable:

$$\tau \leq \frac{2}{\rho(A)}.$$

Based on our numerical results above, we estimate that

$$\tau \leq \frac{2}{\rho(A)} \approx \frac{2}{4(0.05n^4)} = \frac{10}{n^4}.$$

Let's test this:

```
n = 24
f = x -> sin(π*x/2) + 0.5*sin(2π*x)
ϕ1 = t -> exp(-π^2*t/4)
ϕ0 = t -> 0
T = 0.25
D,s = cheb(n)
x = (s .+ 1)/2
D2 = D^2
F = (t,v) -> 4D2[2:n,2:n]*v + 4*ϕ1(t)*D2[2:n,1] + 4*ϕ0(t)*D2[2:n,n+1]
τest =  10/n^4
τ =  2/(4*norm(eigvals(D2[2:n,2:n]),Inf))
@show [τ τest]
```

```
# Euler's method

t = 0:τ:T
nt = length(t)-1
@show nt
u = zeros(n-1,nt+1); u[:,1] = f.(x[2:n])
@time begin
for i = 1:nt
    u[:,i+1] = u[:,i] + τ*F(t[i],u[:,i])
end
end

@gif for i = 1:100:nt+1
    plot(x[2:n], u[:,1], size = (700, 150), label = "u0")
    plot!(x[2:n], u[:,i], lw=2, label = "Chebyshev collocation,t = " * (@sprintf("%.4f",
t[i])), legend = :outertopright)
    plot!(xx, ue.(xx,t[i]), label = "exact solution")
end

[τ τest] = [3.150641581125344e-5 3.0140817901234566e-5]
nt = 7934
  0.435054 seconds (482.43 k allocations: 108.321 MiB, 10.27% gc time, 60.6
9% compilation time)
Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_yo39tj1Z9D.gi
f")

τ += 1E-7
t = 0:τ:T; nt = length(t)-1
@show nt
u = zeros(n-1,nt+1); u[:,1] = f.(x[2:n])
@time begin
for i = 1:nt
    u[:,i+1] = u[:,i] + τ*F(t[i],u[:,i])
end
end

@gif for i = 1:100:nt+1
    plot(x[2:n], u[:,1], size = (700, 150), label = "u0")
    plot!(x[2:n], u[:,i], lw=2, label = "Chebyshev collocation,t = " * (@sprintf("%.4f",
t[i])), legend = :outertopright)
    plot!(xx, ue.(xx,t[i]), label = "exact solution")
end

nt = 7909
  0.126685 seconds (234.21 k allocations: 87.448 MiB, 42.45% gc time)
Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_WxNm77Qsfg.gi
f")
```

Notice that the (spurious) exponentially growing solution produced by Euler's method appears to resemble the high-frequency eigenfunctions we plotted before.

Of course, there are much better time stepping methods than Euler's method:

```
F = (v,p,t) -> 4D2[2:n,2:n]*v + 4*ϕ1(t)*D2[2:n,1] + 4*ϕ0(t)*D2[2:n,n+1]
prob = ODEProblem(F, f.(x[2:n]), (0.0, T))
soln =  solve(prob,Rodas4(),abstol=1e-4);
length(soln.t)
```

```
13
```

1.3.4 Chebyshev / ultraspherical spectral method

Here we use the orthogonal polynomials we learnt about in Chapter 4 to approximate the solution to the diffusion equation in coefficient space. Specifically, we assume the solution has an expansion in Chebyshev polynomials with time-dependent expansion coefficients, i.e., we assume

$$u(x,t) = \sum_{k=0}^{\infty} u_k(t) T_k(2x-1), \qquad x \in [0,1], \qquad t \in [0,T].$$

Setting

$$T(2x-1) = [T_0(2x-1)|T_1(2x-1)|\cdots]$$

and

$$\mathbf{u}(t) = \begin{pmatrix} u_0(t) \\ u_1(t) \\ \vdots \end{pmatrix}$$

we can write

$$u(x,t) = T(2x-1)\mathbf{u}(t).$$

Recall from Chapter 4 that

$$\frac{\mathrm{d}}{\mathrm{d}s} T(s) = C^{(1)}(s)\mathcal{D}_0, \qquad s \in [-1,1],$$

where

$$C^{(1)}(s) = \left[C_0^{(1)}(s)|C_1^{(1)}(s)|\cdots\right]$$

and $\mathcal{D}_0$ is a banded matrix with bandwidths $(-1,1)$. Also,

$$\frac{\mathrm{d}^2}{\mathrm{d}s^2} T(s) = \frac{\mathrm{d}}{\mathrm{d}s} C^{(1)}(s)\mathcal{D}_0 = C^{(2)}(s)\mathcal{D}_1\mathcal{D}_0$$

where $\mathcal{D}_1$ is an infinite matrix with bandwidths $(-1,1)$ and therefore, setting $s = 2x-1$, we have

$$\frac{\partial^2}{\partial x^2} u(x,t) = \frac{\mathrm{d}^2}{\mathrm{d}x^2} T(2x-1)\mathbf{u}(t) = 4C^{(2)}(2x-1)\mathcal{D}_1\mathcal{D}_0\mathbf{u}(t).$$

We therefore need to express the time derivative in the $C^{(2)}$ basis:

$$\frac{\partial}{\partial t} u(x,t) = T(2x-1)\mathbf{u}'(t) = C^{(1)}(2x-1)\mathcal{S}_0\mathbf{u}'(t) = C^{(2)}(2x-1)\mathcal{S}_1\mathcal{S}_0\mathbf{u}'(t),$$

where $\mathcal{S}_0$ and $\mathcal{S}_1$ are infinite banded matrices with bandwidths $(0,2)$ (see chapter 4). Setting $u_t = u_{xx}$, we conclude that

$$\mathcal{S}_1\mathcal{S}_0\mathbf{u}'(t) = 4\mathcal{D}_1\mathcal{D}_0\mathbf{u}(t)$$

From the initial data, we require

$$u(x,0) = T(2x-1)\mathbf{u}(0) = \sum_{k=0}^{\infty} T_k(2x-1)u_k(0) = f(x).$$

As we know from Chapter 3, we can approximate the Chebyshev coefficients $u_k(0)$ of $f(x)$ using the FFT or, in Julia we can use ApproxFun.jl, in Matlab we can use Chebfun and in Python, ChebyPy.

```
f = x -> sin(π*x/2) + 0.5*sin(2π*x)
fs = s -> f((s+1)/2)
g = θ -> fs(cos(θ))
n = 30
θ = range(0,2π;length= 2n+2)[1:end-1] # the points θ_j
ct = fft(g.(θ))/(2n+1) # compute approximate Fourier coefficients of g
# approximate Chebyshev coefficients
ut = [ct[1];2ct[2:n+1]]
fc = Fun(f,Chebyshev(0..1))
n = @show length(fc.coefficients)
norm(fc.coefficients - ut[1:n],Inf)

length(fc.coefficients) = 22
1.5484256077849816e-16
```

Let's compute the solution to the system of ODEs

$$\mathcal{S}_1\mathcal{S}_0\mathbf{u}'(t) = 4\mathcal{D}_1\mathcal{D}_0\mathbf{u}(t),$$

where we truncate the infinite system and $\mathbf{u}(0)$ is replaced with a finite vector of approximate Chebyshev coefficients of $f(x)$.

```
S10 = Conversion(Chebyshev(),Ultraspherical(2))
D10 = Derivative(Chebyshev(),2)
L = S10[1:n,1:n]
R = 4*D10[1:n,1:n]
T = 0.1
F = (u,p,t) -> L\(R*u)
prob = ODEProblem(F, fc.coefficients, (0.0, T))
soln =  solve(prob,RK4(),reltol=1E-6);

xx = range(0,1;length=301)
t = soln.t
u = soln.u
S = Chebyshev(0..1)
nt = length(soln.t)
@gif for i = 1:nt
    plot(xx, Fun(S,u[1]).(xx), size = (700, 150), label = "u0")
    plot!(xx, Fun(S,u[i]).(xx), lw=2, label = "Chebyshev spectral
    method,t = " * (@sprintf("%.4f", t[i])), legend = :outertopright)
    plot!(xx, ue.(xx,t[i]), label = "exact solution")
end

Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_LlbARxTc6k.gi
f")
```

The problem is that we have not imposed the boundary conditions. From the boundary data, we require

$$u(0,t) = \varphi_0(t) = T(-1)\mathbf{u}(t) = (1 \;\; -1 \;\; 1 \;\; -1 \;\; \cdots)\,\mathbf{u}(t)$$

and

$$u(1,t) = \varphi_1(t) = T(1)\mathbf{u}(t) = (1 \;\; 1 \;\; 1 \;\; 1 \;\; \cdots)\,\mathbf{u}(t).$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{S}_1\mathcal{S}_0\mathcal{B}\mathbf{u}(t) = \mathcal{D}_1\mathcal{D}_0\mathbf{u}(t)$$

Suppose we approximate the solution to the ODE system with Euler's method, then

$$\mathcal{S}_1\mathcal{S}_0\left(\frac{\mathbf{u}^{i+1}-\mathbf{u}^i}{\tau}\right) = 4\mathcal{D}_1\mathcal{D}_0\mathbf{u}^i$$

where

$$\mathbf{u}(t_i) \approx \mathbf{u}^i, \qquad t_i = i\tau, \qquad i = 0, 1, \ldots, n_t.$$

Then, imposing the time-dependent boundary conditions on the approximate solution, the method becomes

$$\begin{pmatrix} 1 & -1 & 1 & -1 & \cdots \\ 1 & 1 & 1 & 1 & \cdots \\ & & \mathcal{S}_1\mathcal{S}_0 & & \end{pmatrix}\mathbf{u}^{i+1} = \begin{pmatrix} \varphi_0(t_{i+1}) \\ \varphi_1(t_{i+1}) \\ (\mathcal{S}_1\mathcal{S}_0 + 4\tau\mathcal{D}_1\mathcal{D}_0)\,\mathbf{u}^i \end{pmatrix}.$$

What step size τ should be used? For simplicity, let's consider again the ODE system without the imposing the boundary data, i.e.,

$$\mathcal{S}_1\mathcal{S}_0\mathbf{u}'(t) = 4\mathcal{D}_1\mathcal{D}_0\mathbf{u}(t).$$

Here is a finite section of the matrix $\mathcal{S}_0^{-1}\mathcal{S}_1^{-1}\mathcal{D}_1\mathcal{D}_0$, which maps the Chebyshev coefficients of a function to the Chebyshev coefficients of the second derivative of a function.

```
n = 8
S10[1:n,1:n]\D10[1:n,1:n]
```

```
8×8 Matrix{Float64}:
 0.0  0.0  4.0   0.0  32.0    0.0  108.0    0.0
 0.0  0.0  0.0  24.0   0.0  120.0    0.0  336.0
 0.0  0.0  0.0   0.0  48.0    0.0  192.0    0.0
 0.0  0.0  0.0   0.0   0.0   80.0    0.0  280.0
 0.0  0.0  0.0   0.0   0.0    0.0  120.0    0.0
 0.0  0.0  0.0   0.0   0.0    0.0    0.0  168.0
 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
```

```
nv = 6:82
matnorms = [opnorm((S10[1:n,1:n]\D10[1:n,1:n])[1:n-2,3:n],2) for n = nv]
```

```
77-element Vector{Float64}:
   145.60951439410422
   255.9559967512866
   487.81915083771935
   743.0512089932888
  1223.055719159874
  1712.7272396989702
```

```
 2572.9125430209397
 3407.855952249272
 4808.18389773677
 6120.519836054798
    ⋮
    3.8412010264818533e6
    4.023889395572194e6
    4.273797786140858e6
    4.471621178620519e6
    4.741939169402972e6
    4.955711123536096e6
    5.247520600745311e6
    5.478075949281649e6
    5.7924867364494335e6
```

```
[(-1).^(0:n-1)';ones(1,n);[I(n-2) zeros(n-2,2)]]\S10[1:n,1:n]\D10[1:n,1:n]
```

```
Error: MethodError: \(::SparseArrays.SparseMatrixCSC{Float64, Int64}, ::Ban
dedMatrices.BandedMatrix{Float64, Matrix{Float64}, Base.OneTo{Int64}}) is a
mbiguous. Candidates:
  \(x::AbstractMatrix, A::ArrayLayouts.LayoutMatrix) in ArrayLayouts at C:\
Users\mfaso\.julia\packages\ArrayLayouts\wLwYi\src\ldiv.jl:143
  \(A::SparseArrays.AbstractSparseMatrixCSC, B::AbstractVecOrMat) in Sparse
Arrays at C:\Users\mfaso\AppData\Local\Programs\Julia-1.7.2\share\julia\std
lib\v1.7\SparseArrays\src\linalg.jl:1538
Possible fix, define
  \(::SparseArrays.AbstractSparseMatrixCSC, ::ArrayLayouts.LayoutMatrix)
```

```
nmax = 80
nv = 4:nmax
ρ2 =  [ ([(-1).^(0:n-1)';ones(1,n);[I(n-2) zeros(n-2,2)]]\S10[1:n,1:n]\D10[1:n,1:n])[n,n]
for n = nv]
plot(nv,ρ2;yscale=:log10)
plot!(nv,0.05nv.^4)
```

```
Error: MethodError: \(::SparseArrays.SparseMatrixCSC{Float64, Int64}, ::Ban
dedMatrices.BandedMatrix{Float64, Matrix{Float64}, Base.OneTo{Int64}}) is a
mbiguous. Candidates:
  \(x::AbstractMatrix, A::ArrayLayouts.LayoutMatrix) in ArrayLayouts at C:\
Users\mfaso\.julia\packages\ArrayLayouts\wLwYi\src\ldiv.jl:143
  \(A::SparseArrays.AbstractSparseMatrixCSC, B::AbstractVecOrMat) in Sparse
Arrays at C:\Users\mfaso\AppData\Local\Programs\Julia-1.7.2\share\julia\std
lib\v1.7\SparseArrays\src\linalg.jl:1538
Possible fix, define
  \(::SparseArrays.AbstractSparseMatrixCSC, ::ArrayLayouts.LayoutMatrix)
```

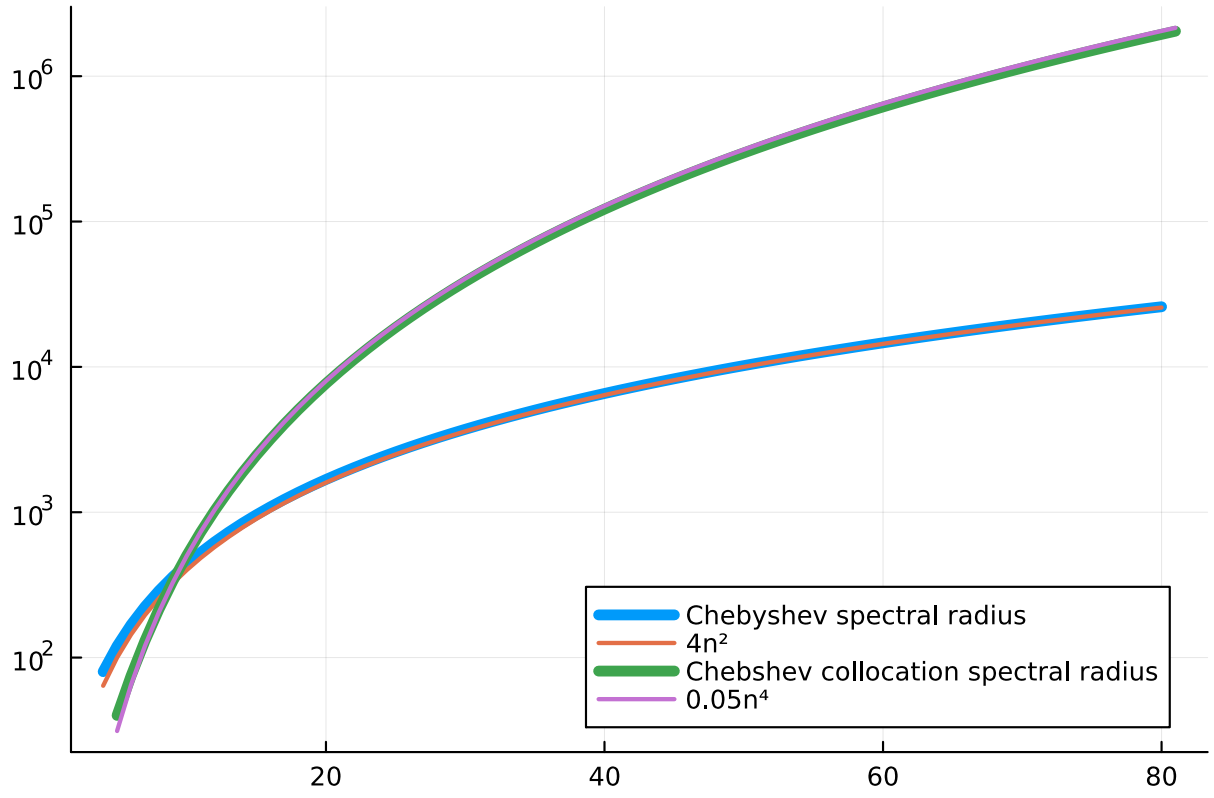
```
n=10
Le = [(-1).^(0:n-1)';ones(1,n);S10[1:n-2,1:n]]
eigvals(inv(Le)[3:n,3:n]*S10[1:n-2,1:n-2])
eigvals(inv(Le)[3:n,3:n]*D10[1:n-2,1:n-2])
eigvals(inv(Le)[3:n,3:n]*(S10[1:n-2,1:n-2] + D10[1:n-2,1:n-2]))
```

```
8-element Vector{Float64}:
 -105.77838341222447
  -61.48478505849292
   -4.12809462416497
   -1.5107460839592013
   -0.19855390883471524
   -0.053248540821022844
```

```
-0.005914948713148342
-0.003909786425958468
```

This suggests that we consider the matrix $A := 4\left(\mathcal{S}_0^{-1}\mathcal{S}_1^{-1}\mathcal{D}_1\mathcal{D}_0\right)[1:n-2, 3:n]$. We'll assume A has an eigendecomposition and use $2/\rho(A)$ as an estimate of the upper bound on τ for Euler's method. Here is how $\rho(A)$ grows with n, which we'll also compare with the growth of the spectral radius for the collocation method we considered in a previous section.

```
nv = 6:82
ρ = [( (S10[1:n,1:n]\D10[1:n,1:n])[n-2,n] ) for n = nv]
plot(nv .- 2,ρ;yscale=:log10,lw=5,label="Chebyshev spectral radius",legend=:bottomright)
plot!(nv .-2,4(nv .-2).^2,lw=2,label="4n2")
plot!(nvc,ρc,lw=5,label="Chebshev collocation spectral radius")
plot!(nvc,0.05*nvc.^4,lw=2,label="0.05n4")
```



This suggests that we should use the following estimate for the upper bound on τ for Euler's method

$$\frac{2}{\rho(A)} \approx \frac{1}{8n^2},$$

which is larger, for large n, compared to the upper bound we estimated for the collocation method, namely

$$\frac{10}{n^4}.$$

Let's test the upper bound:

```
T = 2
n = length(fc.coefficients)
```

```
@show τcollocation = 10/n^4
τ = @show 0.125/n^2
τ = @show 0.000123
t = 0:τ:T
Le = [(-1).^(0:n-1)';ones(1,n);S10[1:n-2,1:n]]
Re = (S10 + 4τ*D10)[1:n-2,1:n]
nt = @show length(t)-1
u = zeros(n,nt+1)
u[:,1] = fc.coefficients

@time begin
for i = 1:nt
    u[:,i+1] = Le\[ϕ0(t[i+1]);ϕ1(t[i+1]);Re*u[:,i]]
end
end
```

```
τcollocation = 10 / n ^ 4 = 4.268834096031692e-5
0.125 / n ^ 2 = 0.00025826446280991736
0.000123 = 0.000123
length(t) - 1 = 16260
  1.979481 seconds (467.97 k allocations: 88.024 MiB)
```

```
xx = range(0,1;length=301)
@gif for i = 1:100:nt+1
    plot(xx, Fun(S,u[:,1]).(xx), size = (700, 150), label = "u0")
    plot!(xx, Fun(S,u[:,i]).(xx), lw=2, label = "Chebyshev spectral method,t = " *
(@sprintf("%.4f", t[i])), legend = :outertopright)
    plot!(xx, ue.(xx,t[i]), label = "exact solution")
end

Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_Fzv74CCquE.gi
f")
```

```
τ = 0.000122
Re = (S10 + 4τ*D10)[1:n-2,1:n]
t = 0:τ:T
nt = @show length(t)-1
u = zeros(n,nt+1)
u[:,1] = fc.coefficients

for i = 1:nt
    u[:,i+1] = Le\[ϕ0(t[i+1]);ϕ1(t[i+1]);Re*u[:,i]]
end

length(t) - 1 = 16393

xx = range(0,1;length=301)
@gif for i = 1:100:nt+1
    plot(xx, Fun(S,u[:,1]).(xx), size = (700, 150), label = "u0")
    plot!(xx, Fun(S,u[:,i]).(xx), lw=2, label = "Chebyshev spectral method,t = " *
(@sprintf("%.4f", t[i])), legend = :outertopright)
    plot!(xx, ue.(xx,t[i]), label = "exact solution")
end

Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_7HqOKNgr7C.gi
f")
```

1.4 Spectral methods from Chapters 2 & 3 revisited

1.4.1 Advection equation

In Chapter 2, we considered the variable coefficient advection equation,

$$u_t + c(x)u_x = 0, \qquad x \in [0, 2\pi), \qquad t \in [0, T],$$

with

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

and the initial data $u(x, 0) = f(x) = \mathrm{e}^{-100(x-1)^2}$. Since the initial data decay rapidly to zero away from $x = 1$, we approximated the solution in the x direction with a periodic trigonometric interpolant and we replaced the time derivative with a central difference approximation to obtain the following Fourier pseudospectral / Fourier collocation method:

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

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the application of the FFT and inverse FFT, the entries of $\mathbf{x} \in \mathbb{R}^{n_x}$ are x_j and the entries of $\mathbf{u}^i \in \mathbb{R}^{n_x}$ are $u_j^i \approx u(x_j, t_i)$, with $x_j = jh$, $h = 2\pi/n_x$, $j = 0, \ldots, n_x - 1$, $n_x = 2m + 1$ and $t_i = i\tau$, $i = 0, \ldots, n_t$ with $n_t\tau = T$.

To analyse the stability of this method, we recall from earlier in this chapter that the FFT can be diagonalised via unitary matrix, i.e.,

$$\mathcal{F}^{-1}\left\{\mathrm{i}(-m\!:\!m) \cdot \mathcal{F}\{\mathbf{u}^i\}\right\} = Q_{n_x}^* P^* \Lambda P Q_{n_x} \mathbf{u}^i, \qquad \Lambda = \mathrm{i}\,\mathrm{diag}(-m, -(m-1), \ldots, 0, 1, \ldots, m).$$

If we replace $c(\mathbf{x}) \in \mathbb{R}^{n_x}$ with $c_{\max} = \max\{c(x) : x \in [0, 2\pi)\} = 6/5$ and set $\mathbf{w}^i = PQ_{n_x}\mathbf{u}^i$, then the method decouples into n_x scalar equations,

$$w_j^{i+1} = w_j^{i-1} - 2\mathrm{i}\tau c_{\max}\lambda_j w_j^i, \qquad \lambda_j = j - m, \qquad j = 0, \ldots, n_x - 1.$$

Using the Von Neumann method of stability analysis, we use the ansatz $w_j^i = \rho^i \mathrm{e}^{\mathrm{i}kx_j}$ and conclude that

$$\rho - \rho^{-1} = -2i\tau c_{max}\lambda_j$$

Call the solutions to this equation ρ_+ and ρ_-. Note that if ρ_+ satisfies this equation, then so does $-\rho_+^{-1}$, which implies that $\rho_- = -\rho_+^{-1}$. For the method to be stable, it is necessary and sufficient that $|\rho_\pm| \leq 1$ and if $|\rho_\pm| = 1$, we additionally require that $\rho_+ \neq \rho_-$ (this is known as the root condition, see *A First Course in the Numerical Analysis of Differential Equations* by A. Iserles, p. 25). If $|\rho_+| < 1$, then $|\rho_-| > 1$, therefore we require $|\rho_+| = 1$ and $\rho_+ \neq \rho_- = -\rho_+^{-1}$, i.e., $\rho_+ \neq \pm i$.

Using the quadratic formula, it follows that

$$\rho_\pm = -\mathrm{i}\tau c_{\max}\lambda_j \pm \left(-\tau^2 c_{\max}^2 \lambda_j^2 + 1\right)^{1/2}$$

and hence the method will be stable if and only if $\tau^2 c_{\max}^2 \lambda_j^2 < 1$ or

$$\tau < \min_{j=0,\ldots,n_x-1} \left\{ \frac{1}{c_{\max}|\lambda_j|} \right\} = \frac{1}{c_{\max} m} = \frac{2}{c_{\max}(n-1)},$$

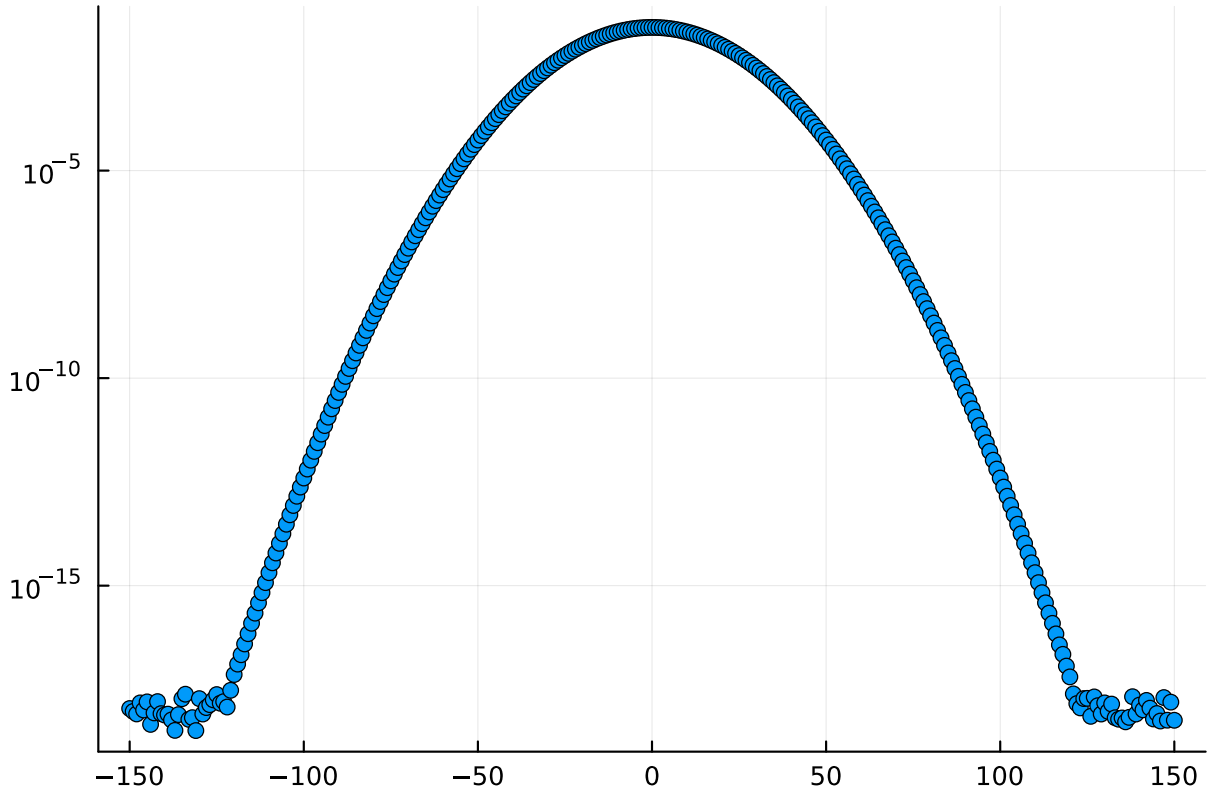
because then

$$|\rho_\pm|^2 = 1, \qquad \rho_+ \neq \rho_-.$$

If there's a j such that $\tau^2 c_{\max}^2 \lambda_j^2 \geq 1$, then these conditions are not satisfied.

Let's test the upper bound on the step size with $c(x) = 1$. First, we determine how many approximate Fourier coefficients we need to resolve the initial data to machine precision accuracy:

```
n_x = 301
m = (n_x-1)÷2
f = x -> exp(-100*(x-1)^2)
x = range(0,2π;length=n_x+1)[1:end-1]
c = fftshift(fft(f.(x)))/n_x
scatter(-m:m,abs.(c);yscale=:log10,legend=false)
```




```
T = 2
cmax = 1
τ = @show 1/(cmax*m)+1e-4
t = 0:τ:T
n_t = @show length(t)-1
u = zeros(n_t + 2,n_x)
u[2,:] = f.(x)  # initial data
u[1,:] = f.(x .+ τ)
for n = 2:n_t
    u[n+1,:] = real.(u[n-1,:] - 2τ*ifft(ifftshift(im*(-m:m)).*fft(u[n,:])))
```

```
end
```

```
1 / (cmax * m) + 0.0001 = 0.006766666666666667
length(t) - 1 = 295
```

```
@gif for i = 2:2:n_t+1
    plot(x, u[2,:], size = (700, 150), label = "u0")
    plot!(x, f.(x .- mod2pi((i-1)*τ*cmax)), lw=2, size = (700, 150),label="exact
solution")
    plot!(x, u[i,:], label = "Fourier pseudospectral method,t = " * (@sprintf("%.4f",
t[i])), legend = :outertopright)
end
```

```
Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_97OAnvmI5A.gi
f")
```

```
τ -= 2E-4
T = 2π
t = 0:τ:T
n_t = @show length(t)-1
@show τ
u = zeros(n_t + 2,n_x)
u[2,:] = f.(x)  # initial data
u[1,:] = f.(x .+ τ)
for n = 2:n_t
    u[n+1,:] = real.(u[n-1,:] - 2τ*ifft(ifftshift(im*(-m:m)).*fft(u[n,:])))
end
```

```
length(t) - 1 = 956
τ = 0.006566666666666668
```

```
@gif for i = 2:10:n_t+1
    plot(x, u[2,:], size = (700, 150), label = "u0")
    plot!(x, f.(mod2pi.(x .- (i-1)*τ*cmax)), lw=2, size = (700, 150),label="exact
solution")
    plot!(x, u[i,:], label = "Fourier pseudospectral method,t = " * (@sprintf("%.4f",
t[i])), legend = :outertopright)
end
```

```
Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_HT6Yt3lwQ2.gi
f")
```

Now let's consider the case when $c(x) = 1/5 + \sin^2(x-1)$

```
c = x -> 0.2 + sin(x - 1)^2
n_x = 401
cmax = 6/5
m = (n_x-1)÷2
x = range(0,2π;length=n_x+1)[1:end-1]
τ = @show 1/(m*cmax)
τ = 0.0044 # no blow up
#τ = 0.00441 #blow up
T = 12.825
t = 0:τ:T
n_t = @show length(t)-1
u = zeros(n_t + 2,n_x)
u[2,:] = f.(x)
u[1,:] = f.(x .+ 0.2*τ)

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
```

```
1 / (m * cmax) = 0.004166666666666667
length(t) - 1 = 2914

@gif for i = 2:20:n_t+1
    plot(x, u[2,:], size = (700, 150), label = "u0")
    plot!(x, u[i,:], lw=2, label = "Fourier collocation,t = " * (@sprintf("%.4f",
t[i])), legend = :outertopright)
end

Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_1wl17GMvlh.gi
f")

τ = 0.00441 #blow up
t = 0:τ:T
n_t = @show length(t)-1
u = zeros(n_t + 2,n_x)
u[2,:] = f.(x)
u[1,:] = f.(x .+ 0.2*τ)
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

length(t) - 1 = 2908

@gif for i = 2:20:n_t+1
    plot(x, u[2,:], size = (700, 150), label = "u0")
    plot!(x, u[i,:], lw=2, label = "Fourier collocation,t = " * (@sprintf("%.4f",
t[i])), legend = :outertopright)
end

Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_z6Zsj5RAdF.gi
f")
```

1.4.2 Wave equation

In Chapter 3, we considered the wave equation

$$u_{tt} = u_{xx}, \qquad t > 0, \qquad -1 < x < 1, \qquad ,$$

subject to zero boundary conditions $u(\pm 1, t) = 0$ and the initial data

$$u(x, 0) = f(x) = \mathrm{e}^{-200x^2}.$$

Rather than specifying $u_t(x, 0)$, we imposed the condition

$$u(x, -\tau) = f(x - \tau) = \mathrm{e}^{-200(x-\tau)^2}.$$

We approximated the solution with a Chebyshev interpolant in x, i.e.,

$$u(x,t) \approx u_{n_x}(x,t) = \sum_{k=0}^{n_x} \ell_k(x) v_k(t), \quad \ell_k(x) = \prod_{\substack{j=0 \\ j \neq k}}^{n_x} \frac{x - x_j}{x_k - x_j}, \quad x_k = \cos\left(\frac{\pi k}{n_x}\right),$$

where $u(x_j, t) \approx v_j(t)$. Due to the zero boundary data, we set $v_0(t) = v_{n_x}(t)$. We specify that $u_{n_x}(x, t)$ satisfy the wave equation at the collocation points, i.e.,

$$\frac{\partial^2}{\partial t^2} u_{n_x}(x_j, t) = \frac{\partial^2}{\partial x^2} u_{n_x}(x_j, t),$$

which is equivalent to the second-order system of ODEs:

$$\begin{pmatrix} v_1''(t) \\ v_2''(t) \\ \vdots \\ v_{n_x-1}''(t) \end{pmatrix} = \begin{pmatrix} \ell_0''(x_1) & \ell_1''(x_1) & \cdots & \ell_{n_x-1}''(x_1) & \ell_{n_x}''(x_1) \\ \ell_0''(x_2) & \ell_1''(x_2) & \cdots & \ell_{n_x-1}''(x_2) & \ell_{n_x}''(x_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \ell_0''(x_{n_x-1}) & \ell_1''(x_{n_x-1}) & \cdots & \ell_{n_x-1}''(x_{n_x-1}) & \ell_{n_x}''(x_{n_x-1}) \end{pmatrix} \begin{pmatrix} v_0(t) \\ v_1(t) \\ \vdots \\ v_{n_x-1}(t) \\ v_{n_x}(t) \end{pmatrix}$$

$$= \begin{pmatrix} \ell_1''(x_1) & \cdots & \ell_{n_x-1}''(x_1) \\ \ell_1''(x_2) & \cdots & \ell_{n_x-1}''(x_2) \\ \vdots & \vdots & \vdots \\ \ell_1''(x_{n_x-1}) & \cdots & \ell_{n_x-1}''(x_{n_x-1}) \end{pmatrix} \begin{pmatrix} v_1(t) \\ \vdots \\ v_{n_x-1}(t) \end{pmatrix} := A\mathbf{v}(t).$$

Note that the matrix A is the same as the matrix we encountered when we solved the diffusion equation with a Chebyshev collocation method earlier in this chapter; it is the second to n-th columns of the second-order Chebyshev differentiation matrix.

We approximated the second time derivative with a central difference, hence we set

$$v_j(t_i) \approx u_j^i, \qquad t_i = i\tau, \qquad i = 0, \ldots, n_t,$$

and

$$v_j''(t_i) \approx \frac{u_j^{i+1} - 2u_j^i + u_j^{i-1}}{\tau^2}.$$

Substituting this into the second-order ODE system, we obtain the leap frog method

$$\mathbf{u}^{i+1} = 2\mathbf{u}^i - \mathbf{u}^{i-1} + \tau^2 A\mathbf{u}^i, \qquad i = 0, \ldots, n_t - 1.$$

and from the initial data, we set

$$\mathbf{u}^0 = f(\mathbf{x}), \qquad \mathbf{u}^{-1} = f(\mathbf{x} - \tau).$$

To analyse the stability of the method, we recall from earlier in this chapter that A has the eigendecomposition $A = W\Lambda W^{-1}$. Setting $\mathbf{w}^i = W^{-1}\mathbf{u}^i$, the leap frog method becomes

$$\mathbf{w}^{i+1} = 2\mathbf{w}^i - \mathbf{w}^{i-1} + \tau^2 \Lambda \mathbf{w}^i,$$

i.e.,

$$w_j^{i+1} = 2w_j^i - w_j^{i-1} + \tau^2 \lambda_j w_j^i, \qquad j = 1, \ldots, n_x - 1,$$

where the λ_j are the eigenvalues of A. We learnt earlier in this chapter that the λ_j are negative real numbers and that

$$\rho(A) = \max_{j=1,\ldots,n_x-1} \{|\lambda_j|\} \approx \frac{(n_x - 1)^4}{20}.$$

Using the Von Neumann method of stability analysis, we set $w_j^i = \rho^i e^{ikx_j}$ and find that

$$\rho + \rho^{-1} = 2\left(1 - \frac{\tau^2}{2}|\lambda_j|\right)$$

Call the solutions to this equation ρ_+ and ρ_-. Note that if ρ_+ satisfies this equation, then so does ρ_+^{-1}, which implies that $\rho_- = \rho_+^{-1}$. For the method to be stable, it is necessary and sufficient that $|\rho_\pm| \leq 1$ and if $|\rho_\pm| = 1$, we additionally require that $\rho_+ \neq \rho_-$. If $|\rho_+| < 1$, then $|\rho_-| > 1$, therefore we require $|\rho_+| = 1$ and $\rho_+ \neq \rho_- = \rho_+^{-1}$, i.e., $\rho_+ \neq \pm 1$.

Using the quadratic formula and assuming $\tau^4|\lambda_j|^2/4 - \tau^2|\lambda_j| < 0$ or $\tau < 2/\sqrt{|\lambda_j|}$ for $j = 1, \ldots, n_x - 1$, it follows that

$$\rho_\pm \quad = \quad 1 - \frac{\tau^2}{2}|\lambda_j| \pm i\left(\tau^2|\lambda_j| - \frac{\tau^4}{4}|\lambda_j|^2\right)^{1/2}$$

and hence the method is stable because $|\rho_\pm|^2 = 1$ and $\rho_+ \neq \rho_-$. It also follows that if there's a j such that $\tau \geq 2/\sqrt{|\lambda_j|}$, then these conditions are not satisfied and the method is unstable.

We estimate the step size restriction to be

$$\tau < \min_{j=1,\ldots,n_x-1}\left\{\frac{2}{\sqrt{|\lambda_j|}}\right\} \approx \frac{4\sqrt{5}}{(n_x-1)^2}$$

Let's test this: First, we shall use the following function to compute the matrix-vector product $A\mathbf{u}^i$ via the FFT.

```
function chebfft(f)
    n = length(f)-1
    x = cos.((0:n)*π/n) # Chebyshev points
    ii = 0:n-1
    q = [f; f[n:-1:2]]      # transform x -> θ
    # differentiate the interpolant q_n in coefficient space and map back to function
values
    cq = real.(fft(q))
    dq = real.(ifft(im*[ii; 0; 1-n:-1] .*cq))
    df = zeros(n+1,1)
    # Compute approximations to f' at the interior points
    df[2:n] = -dq[2:n] ./sqrt.(1 .- x[2:n].^2)    # transform θ -> x
    # At the boundary points
    df[1] = sum(ii.^2 .*cq[1:n])/n + .5*n*cq[n+1]
    df[n+1] = sum((-1) .^(ii .+1) .* ii.^2 .*cq[1:n])/n +
              .5*(-1)^(n+1)*n*cq[n+1]
    df
end
```

```
chebfft (generic function with 1 method)
```

```
N = 80; x = cos.(π*(0:N)/N);
@show τest = 4*sqrt(5)/(N-1)^2
@show dt = 0.0015385 #blow up
dt = 0.0015384  # no blow up
dt = τest/2
v = exp.(-200*x.^2);vold = exp.(-200*(x .- dt).^2)
```

```
tmax = 4; tplot = 0.02;
plotgap = Int64(round(tplot/dt)); dt = tplot/plotgap;
nplots = @show Int64(round(tmax/tplot))
plotdata = [transpose(v); zeros(nplots,N+1)]; tdata = [0.0];
@time begin
for i = 1:nplots
    for n = 1:plotgap
        global v
        global vold
        w = chebfft(chebfft(v)); w[1]=0; w[N+1] = 0;
        vnew = 2*v - vold + dt^2*w; vold = v; v = vnew;
    end
        global tdata
    plotdata[i+1,:] = v; tdata = [tdata;dt*i*plotgap]
end
end
```

```
τest = (4 * sqrt(5)) / (N - 1) ^ 2 = 0.0014331472376220413
dt = 0.0015385 = 0.0015385
Int64(round(tmax / tplot)) = 200
  2.408964 seconds (3.00 M allocations: 446.374 MiB, 3.46% gc time, 29.47%
compilation time)
```

```
2/sqrt(0.05*(N-1)^4)
D,~ = cheb(N)
D2 = D^2
A = D2[2:N,2:N]
2/sqrt(norm(eigvals(A),Inf))
```

```
0.001435271552465032
```

```
@gif for i = 1:length(tdata)
    plot(x, plotdata[1,:], size = (700, 150), label = "u0")
    plot!(x, plotdata[i,:], lw=2, label = "Chebyshev collocation,t = " *
(@sprintf("%.4f", tdata[i])), legend = :outertopright)
end
```

```
Plots.AnimatedGif("C:\\Users\\mfaso\\AppData\\Local\\Temp\\jl_Vn6K6PQy1g.gi
f")
```

```
fc = Fun(x -> exp(-200*x^2))
fc.coefficients[80:81]
```

```
2-element Vector{Float64}:
 0.0
 2.8582872672610978e-5
```

1.5 Concluding remarks

Big topics we didn't get to:

-PDEs with more than one spatial variable

-Nonlinear PDEs and automatic differentiation for computation of jacobians

-Structure preservation

-Refer to the two trefethen books and Ortner's notes

-Finite element method, see Manolis's notes

-Software: dedalus