1 Lecture 9: Matrix exponentials and the (fractional) heat equation

1.1 Matrix exponentials

Let $A \in \mathbb{C}^{d\times d}$, $\mathbf{u}_0 \in \mathbb{C}^d$ and consider the constant coefficient linear ODE

$$\mathbf{u}'(t) = A\mathbf{u}(t) \qquad \text{and} \qquad \mathbf{u}(0) = \mathbf{u}_0(0)$$

The solution to this is given by the *matrix exponential*

$$\mathbf{u}(t) = \exp(At)\mathbf{u}_0$$

where the matrix exponential is defined by its Taylor series:

$$\exp(A) = \sum_{k=0}^{\infty} \frac{A^k}{k!}$$

This has stability problems, so a more convenient form is as follows:

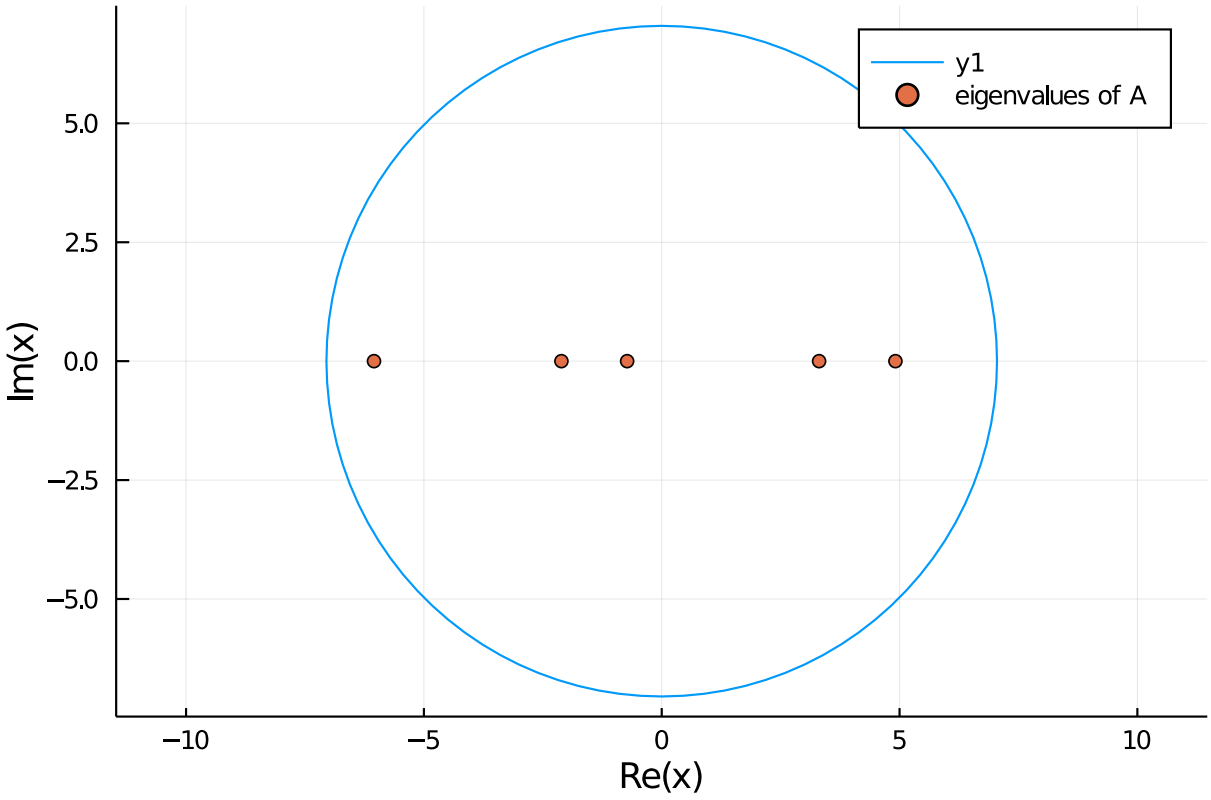
Demonstration we use this formula alongside the complex trapezium rule to calculate matrix exponentials. Begin by creating a random symmetric matrix (which only has real eigenvalues):

```
using LinearAlgebra, Plots, ComplexPhasePortrait, ApproxFun

A = randn(5,5)
A = A + A'
λ = eigvals(A)
```

```
5-element Array{Float64,1}:
 -6.049213738987934
 -2.1092490997038333
 -0.7274172568961783
  3.3086209210144317
  4.913382322134822
```

We can now by hand create a circle that surrounds all the eigenvalues:

```
periodic_rule(N) = 2π/N*(0:(N-1)), 2π/N*ones(N)
function circle_rule(n, r)
    θ = periodic_rule(n)[1]
    r*exp.(im*θ), 2π*im*r/n*exp.(im*θ)
end
z,w = circle_rule(100,maximum(abs.(λ))+1)

plot(vcat(z,z[1]))
scatter!(λ,zeros(5); label = "eigenvalues of A", ratio=1.0)
```



Here we wrap this up into a function `circle_exp` that calculates the matrix exponential:

```
function circle_exp(A, n, z_0, r)
    z,w = circle_rule(n,r)
    z .+= z_0

    ret = zero(A)
    for j=1:n
        ret += w[j]*exp(z[j])*inv(z[j]*I - A)
    end

    ret/(2π*im)
end

circle_exp(A, 100, 0, 8.0) -exp(A) |>norm
```

```
7.791475725307949e-13
```

In this case, it is beneficial to use an ellipse:

```
function ellipse_rule(n, a, b)
    θ = periodic_rule(n)[1]
    a*cos.(θ) + b*im*sin.(θ), 2π/n*(-a*sin.(θ) + im*b*cos.(θ))
end
function ellipse_exp(A, n, z_0, a, b)
    z,w = ellipse_rule(n,a,b)
    z .+= z_0

    ret = zero(A)
    for j=1:n
        ret += w[j]*exp(z[j])*inv(z[j]*I - A)
    end
    ret/(2π*im)
end
```

```
ellipse_exp(A, 50, 0, 8.0, 5.0) -exp(A) |>norm
```

```
5.004706602329218e-13
```

For matrices with large negative eigenvalues (For example, discretisations of the Laplacian), complex quadrature can lead to much better accuracy than Taylor series:

```
function taylor_exp(A,n)
    ret = Matrix(I, size(A))
    for k=1:n
        ret += A^k/factorial(1.0k)
    end
    ret
end
```

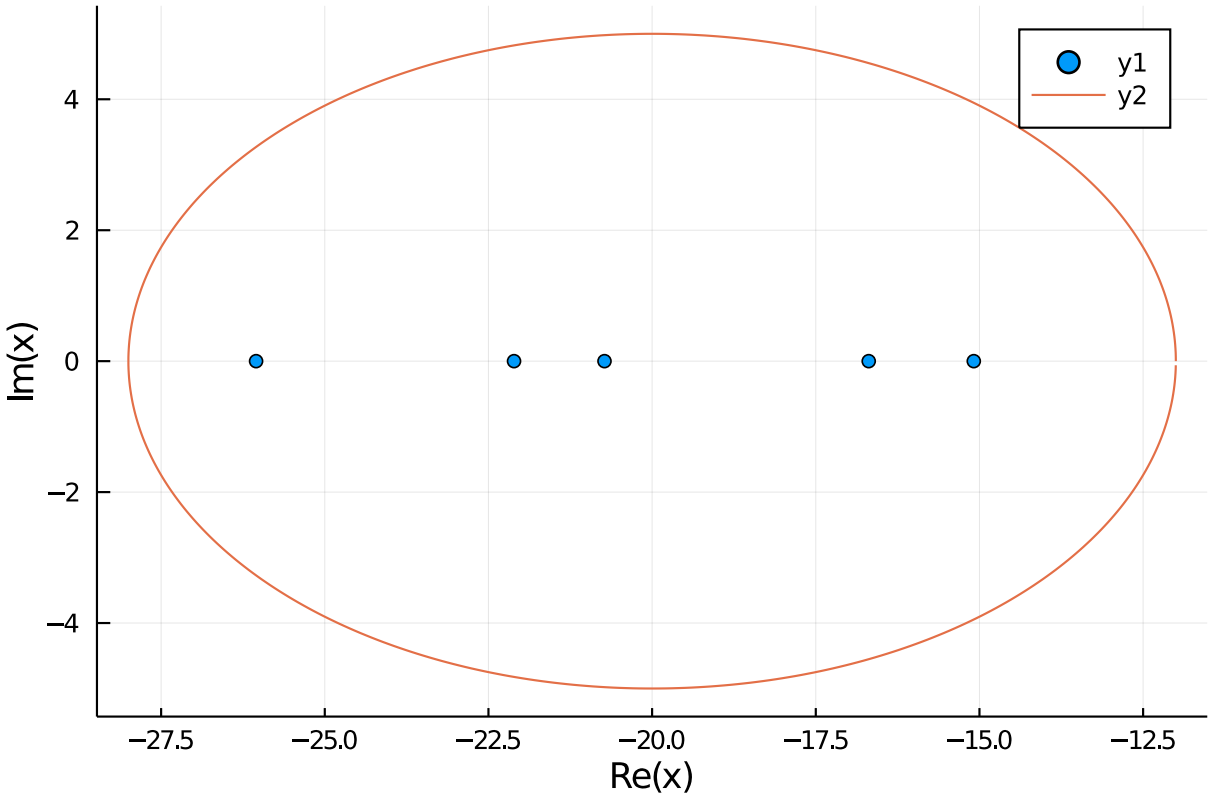
```
B = A - 20I
```

```
taylor_exp(B, 200) -exp(B) |>norm
```

```
4.11529740224729e-6
```

We can use an ellipse to surround the spectrum:

```
scatter(complex.(eigvals(B)))
plot!(ellipse_rule(500,8,5)[1] .- 20;ratio=1.0)
```




```
norm(ellipse_exp(B, 50, -20.0, 8.0, 5.0) - exp(B))
```

```
1.564157554840039e-21
```

1.2 Finite differences and the (fractional) heat equation

As a first application we consider approximation of the solution of the heat equation $u(t,x)$, for $\Delta u := u_{xx}$ we have:

$$u_t = \Delta u$$

on $x \in [0,1]$ with Dirichlet conditions $u(t,0) = u(t,1) = 0$ and initial condition $u(0,x) = u_0(x)$. We approximate the solution by its values at a grid, that is, we have a time dependent vector $\mathbf{u}(t)$ which we hope satisfies the property that at any time t

$$\mathbf{u}(t) \approx \begin{pmatrix} u(t,x_1) \\ \vdots \\ u(t,x_N) \end{pmatrix}$$

where $x_j = j/(N+1)$ is an evenly spaced grid. Note we use that $u(t,x_0) = u(t,0) = 0$ and $u(t,x_{N+1}) = u(t,1) = 0$.

Recall from Taylor series that for a smooth enough function $f(x)$ we have

$$\begin{aligned} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} &\approx \frac{f(x) + f'(x)h + f''(x)h^2/2 - 2f(x) + f(x) - f'(x)h + f''(x)h^2/2}{h^2} \\ &= f''(x) \end{aligned}$$

For $h = 1/(N+1)$ this therefore gives

$$\begin{aligned} u_{xx}(t,x_1) &\approx \frac{u(t,x_2) - 2u(t,x_1) + u(t,x_0)}{h^2} = \frac{u(t,x_2) - 2u(t,x_1)}{h^2} \\ u_{xx}(t,x_N) &\approx \frac{u(t,x_{N+1}) - 2u(t,x_N) + u(t,x_{N-1})}{h^2} = \frac{-2u(t,x_N) + u(t,x_{N-1})}{h^2} \\ u_{xx}(t,x_j) &\approx \frac{u(t,x_{j+1}) - 2u(t,x_j) + u(t,x_{j-1})}{h^2} \end{aligned}$$

Or in vector form we have

$$\begin{pmatrix} u_{xx}(t,x_1) \\ \vdots \\ u_{xx}(t,x_N) \end{pmatrix} \approx \Delta \begin{pmatrix} u(t,x_1) \\ \vdots \\ u(t,x_N) \end{pmatrix}$$

where

$$\Delta_N = \frac{1}{h^2} \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix}$$

In short: a natural approximation to the heat equation is

$$\mathbf{u}'(t) = \Delta_N \mathbf{u}(t), \qquad \mathbf{u}(0) = \mathbf{u}_0$$

where $\mathbf{u}_0 = (u_0(x_1), \ldots, u_0(x_N))^\top$. Thus to evaluate the solution at time t we have

$$\mathbf{u}(t) = \mathrm{e}^{\Delta_N t}\mathbf{u}_0 = \frac{1}{2\pi \mathrm{i}} \oint_\gamma \mathrm{e}^{\zeta t}(\zeta I - \Delta_N)^{-1}\mathbf{u}_0 \mathrm{d}\zeta.$$

We need to surround the spectrum of Δ_N. In this case it suffices to use Gershgorin's theorem to determine:

$$\sigma(\Delta_N) \subset [-4/h^2, 0] = [-4(N+1)^2, 0]$$

Now for something more complicated, the fractional heat equation:

$$u_t = -(-\Delta)^\alpha u$$

where α is say $1/2$. These come up in applications involving "memory", for example, in medical imaging where tissue changes over time. They also come up in Levy flights, that is, stochastic differential equations with heavy-tailed distributions. The precise definition is beyond the scope of this course, but in terms of numerical approximation it is natural to discretise as

$$\mathbf{u}'(t) = -(-\Delta_N)^\alpha \mathbf{u}, \qquad \mathbf{u}(0) = \mathbf{u}_0$$

Or in other words,

$$\mathbf{u}(t) = \mathrm{e}^{-(-\Delta_N)^\alpha t}\mathbf{u}_0 = \frac{1}{2\pi \mathrm{i}} \oint_\gamma \mathrm{e}^{-\zeta^\alpha t}(\zeta I + \Delta_N)^{-1}\mathbf{u}_0 \mathrm{d}\zeta.$$

Gershgorin's theorem does not help us: it states

$$\sigma(-\Delta_N) \subset [0, 4/h^2]$$

but $\mathrm{e}^{-\zeta^\alpha t}$ has a branch point right at 0. Fortunately in this case we have more understanding of the spectrum of Δ_N. In particular, for

$$J = \begin{pmatrix} 0 & 1 & & & \\ 1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 0 & 1 \\ & & & 1 & 0 \end{pmatrix}$$

we have from trigonometric relationships

$$2\cos\theta \sin k\theta = \sin(k+1)\theta + \sin(k-1)\theta$$

that

$$J\begin{pmatrix}\sin\theta\\ \sin 2\theta\\ \vdots\\ \sin N\theta\end{pmatrix} = \begin{pmatrix}\sin 2\theta\\ \sin 3\theta + \sin\theta\\ \vdots\\ \sin(N-2)\theta + \sin N\theta\\ \sin(N-1)\theta\end{pmatrix}$$

$$= 2\cos\theta\begin{pmatrix}\sin\theta\\ \sin 2\theta\\ \vdots\\ \sin(N-1)\theta\\ \sin N\theta\end{pmatrix} - \begin{pmatrix}0\\ 0\\ \vdots\\ 0\\ \sin(N+1)\theta\end{pmatrix}$$

The second term is zero when $\theta = \theta_j = j\pi/(N+1)$, thus we know the eigenvalues of Δ are $\lambda_j = 2\cos\theta_j$ for $j = 1, \ldots, N$. ($j = 0$ is not valid as then the "eigenvector" would be zero.) We need to bound the spectrum away from 0 and 2. First note from telescoping properties of the Taylor series for cosine we have

$$\cos x = 1 - x^2/2 + x^4/24 - \cdots \le 1 - x^2/2 + x^4/24$$

thus our bound above is

$$2\cos\theta_1 = 2\cos\frac{\pi}{N+1} \le 2 - \left(\frac{\pi}{N+1}\right)^2 + \left(\frac{\pi}{N+1}\right)^4/12$$

The same holds for the bound below:

$$2\cos\theta_N \ge -2 + \left(\frac{\pi}{N+1}\right)^2 - \frac{\pi^4}{12(N+1)^4}$$

Thus we have

$$\sigma(\Delta_N) = \sigma(J - 2I)/h^2 \subset \left[-4(N+1)^2 + \pi^2 - \frac{\pi^4}{12(N+1)^2}, -\pi^2 + \frac{\pi^4}{12(N+1)^2}\right]$$

Thus it suffices to take an elliptical contour passing through say $-\pi^2/2$ and $-4(N+1)^2$.

Demonstration We first demonstrate using the built-in matrix exponential. We can construct Δ as follows:

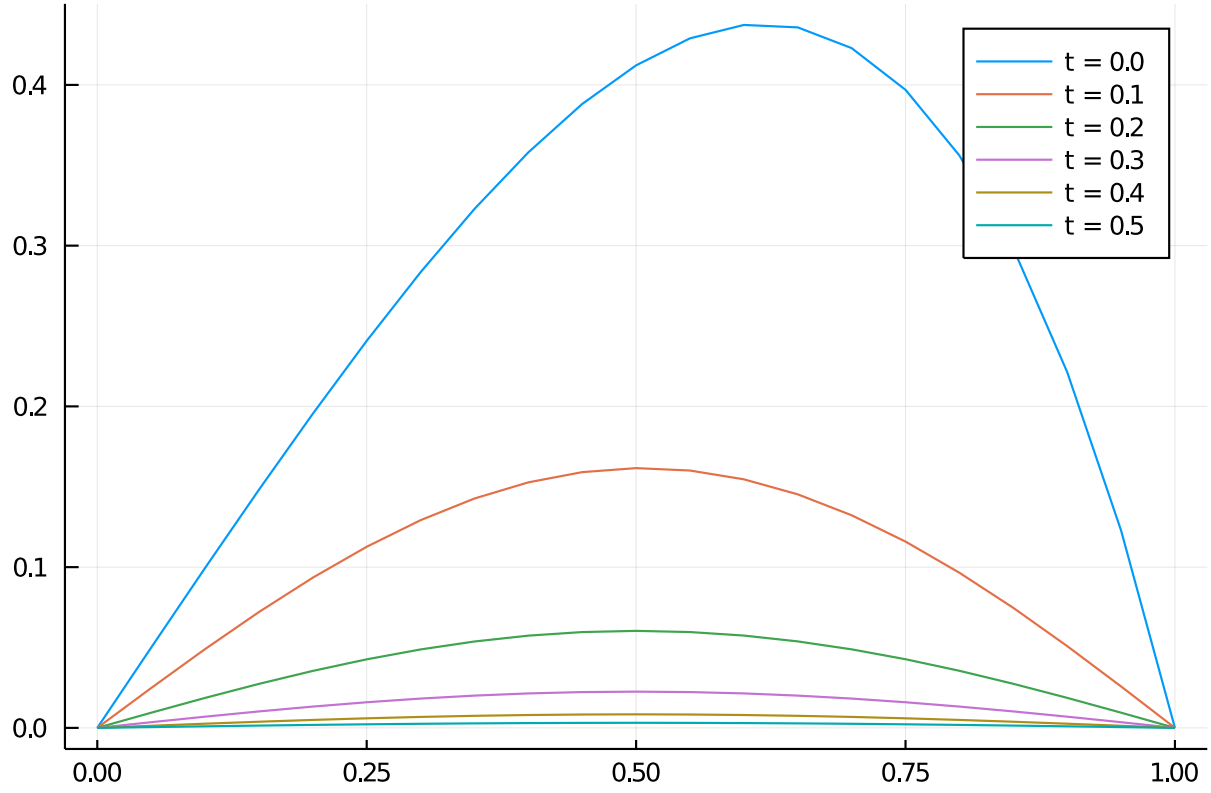
```
N = 19
h = 1/(N+1)
Δ = SymTridiagonal(fill(-2,N), fill(1,N-1))/h^2
```

```
19×19 SymTridiagonal{Float64,Array{Float64,1}}:
 -800.0   400.0      ·        ·        ·   ...      ·        ·        ·        ·
  400.0  -800.0   400.0       ·        ·            ·        ·        ·        ·
    ·     400.0  -800.0   400.0       ·            ·        ·        ·        ·
    ·       ·    400.0  -800.0   400.0             ·        ·        ·        ·
    ·       ·      ·    400.0  -800.0              ·        ·        ·        ·
    ·       ·      ·      ·    400.0  ...          ·        ·        ·        ·
    ·       ·      ·      ·      ·                ·        ·        ·        ·
    ·       ·      ·      ·      ·                ·        ·        ·        ·
    ·       ·      ·      ·      ·                ·        ·        ·        ·
```

```
  .       .       .       .       .          .       .       .       .
  .       .       .       .       .   ...      .       .       .       .
  .       .       .       .       .          .       .       .       .
  .       .       .       .       .          .       .       .       .
  .       .       .       .       .          .       .       .       .
  .       .       .       .       .          .       .       .       .
  .       .       .       .       .   ...   400.0      .       .       .
  .       .       .       .       .      -800.0   400.0      .       .
  .       .       .       .       .       400.0  -800.0   400.0      .
  .       .       .       .       .          .    400.0  -800.0   400.0
  .       .       .       .       .          .       .    400.0  -800.0
```

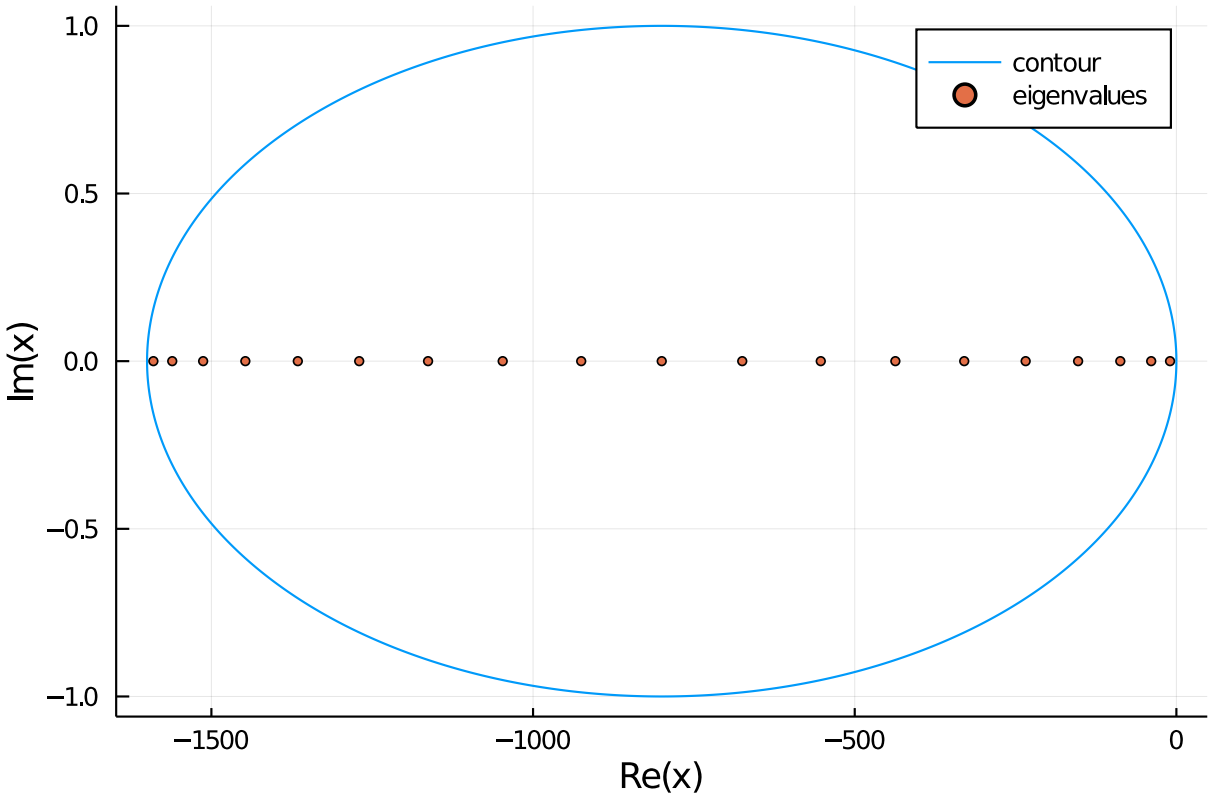
The solution with an initial condition $u_0(x) = x(1-x)e^x$ is then:

```
u0 = x -> x * (1-x) * exp(x)

xx = range(0,1; length=N+2)
A = Matrix(Δ) # Need to convert to dense matrix to diagonalise
p = plot()
for t in 0:0.1:0.5
    plot!(xx, [0; exp(A*t)*u0.(xx[2:end-1]); 0]; label="t = $t")
end
p
```



The above calculation is expensive: it does not take into account the sparsity of Δ. To do better we use Cauchy's integral formula and the trapezium rule. First design an ellipse that surrounds the spectrum:

```
m = 20_000
z,w = ellipse_rule(m,2/h^2,1)
z .-= 2/h^2 # shift to negative
plot(z; label="contour")
scatter!(eigvals(Δ), zeros(N); markersize=3, label="eigenvalues")
```



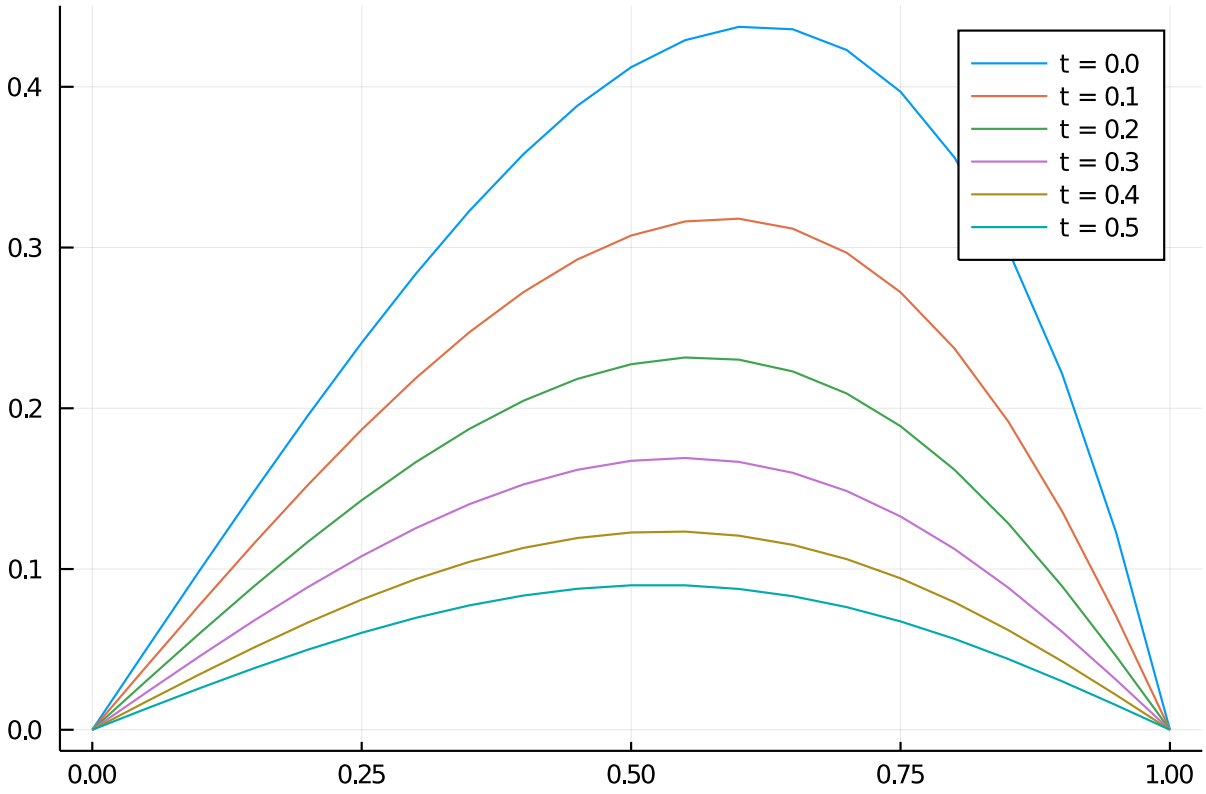
We thus can evaluate the matrix exponential at time t using the quadrature rule:

```
u0_v = u0.(xx[2:end-1])
ret = zero(complex(u0_v))
t = 0.5
for j=1:length(z)
    global ret
    ret += w[j]*exp(t*z[j]) * ((z[j]*I - Δ) \ u0_v)
end
norm(ret/(2π*im) - exp(A*t)*u0_v) # matches to high accuracy
```

```
2.750483902720472e-13
```

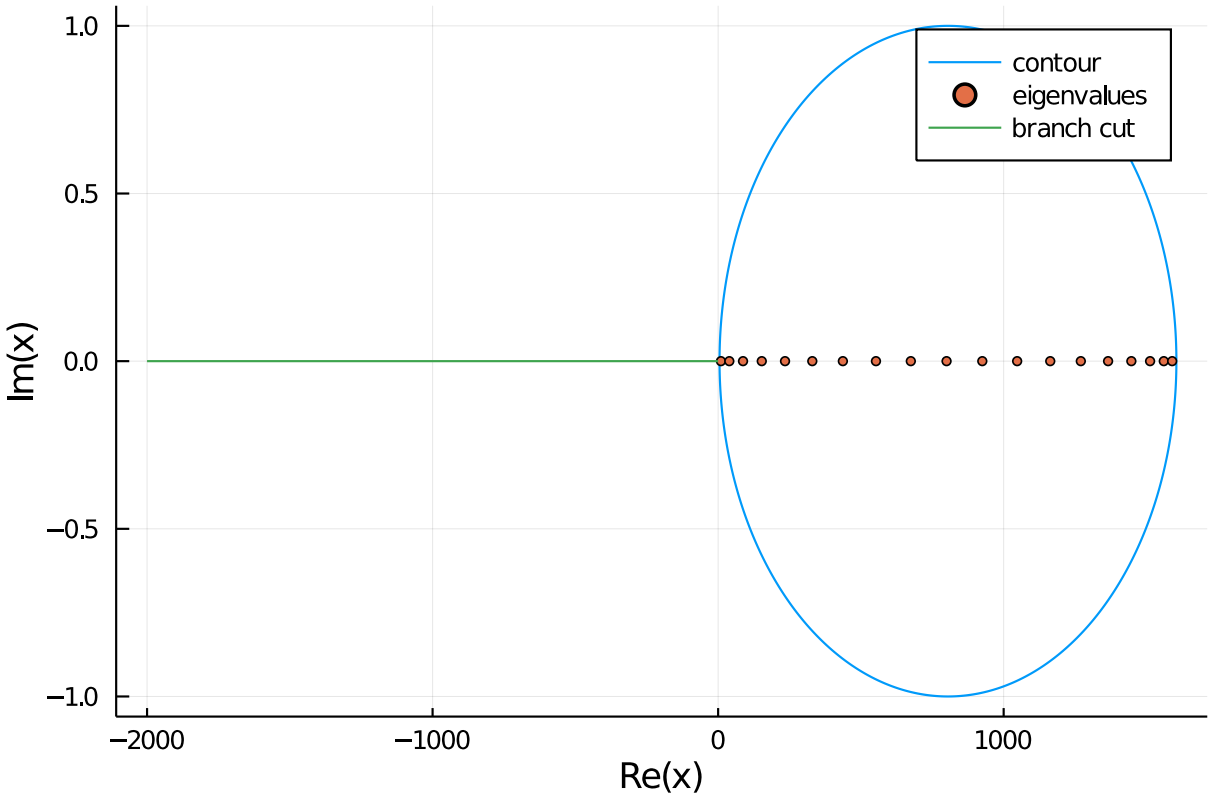
For fractional heat equation, we have slower diffusion:

```
A = -sqrt(-Matrix(Δ))
p = plot()
for t in 0:0.1:0.5
    plot!(xx, [0; exp(A*t)*u0.(xx[2:end-1]); 0]; label="t = $t")
end
p
```



We need to be more careful and navigate the branch cut of $e^{-\sqrt{z}}$. That is, we need the following contour (note we evaluate on $-\Delta$):

```
m = 20_000
z,w = ellipse_rule(m,2/h^2,1)
z .+= 2/h^2 + π^2/2 # shift to positive
plot(z; label="contour")
scatter!(eigvals(-Δ), zeros(N); markersize=3, label="eigenvalues")
plot!([-2000,0],[0,0]; label="branch cut")
```




```
u0_v = u0.(xx[2:end-1])
ret = zero(complex(u0_v))
t = 0.5
for j=1:length(z)
    global ret
    ret += w[j]*exp(-t*sqrt(z[j])) * ((z[j]*I + Δ) \ u0_v)
end
norm(ret/(2π*im) - exp(A*t)*u0_v) # matches to high accuracy
```

```
7.893133333614467e-12
```

In both cases this is the start, not the end, of a very interesting computational problem. As `N` becomes large there it is delicate to get the shape of the contour and the number of quadrature points right to get a desired accuracy. Higham, Functions of Matrices: Theory and Computation, SIAM, 2008 provides a starting point with a number of more recent developments.