M3M6: Methods of Mathematical Physics

Dr. Sheehan Olver

s.olver@imperial.ac.uk

1 Lecture 9: Computing matrix functions via the trapezium rule

Here we look at computing matrix functions via discretising Cauchy's integral formula with the Trapezium rule. That is, we saw that we can define matrix functions via

$$f(A) = \frac{1}{2\pi \mathrm{i}} \oint_\gamma f(\zeta)(\zeta I - A)^{-1} \mathrm{d}\zeta$$

where γ is a contour surrounding the spectrum such that f is analytic in the interior, often a circle or an ellipse. If we parameterise the curve from the circle $\gamma : [0, 2\pi) \to \mathcal{C}$ then we can apply Trapezium rule:

$$f(A) = \frac{1}{2\pi \mathrm{i}} \int_0^{2\pi} f(\gamma(\theta))(\gamma(\theta) I - A)^{-1} \gamma'(\theta) \mathrm{d}\theta \approx \frac{1}{\mathrm{i}N} \sum_{j=0}^{N-1} f(\gamma(\theta_j))\gamma'(\theta_j)(\gamma(\theta_j) I - A)^{-1}.$$

Thus matrix functions are reduced to a sum of inverses. This is useful if applying an inverse is fast, for example, we have

$$f(A)\mathbf{v} \approx \frac{1}{\mathrm{i}N} \sum_{j=0}^{N-1} f(\gamma(\theta_j))\gamma'(\theta_j)(\gamma(\theta_j) I - A)^{-1}\mathbf{v}$$

and if A is sparse then each inverse is fast.

1.1 Example: 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 it's Taylor series:

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

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

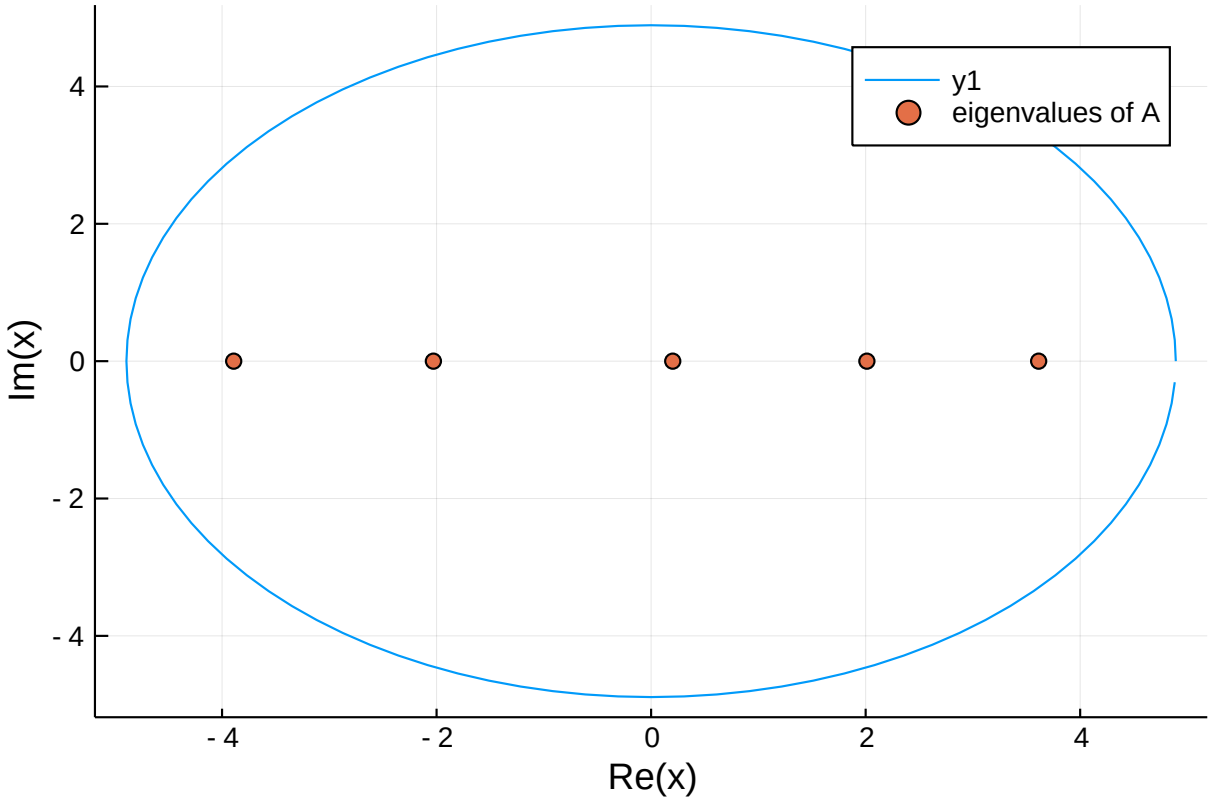
Demonstration we use this formula alongside the complex trapezium rule to calculate matrix exponentials. Begin by creating a random symmetric matrix (which only has real eigenvalues):

```
using LinearAlgebra, Plots, ComplexPhasePortrait, ApproxFun

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

```
5-element Array{Float64,1}:
 -3.8914989454160533
 -2.029755565698448
  0.2016229890832312
  2.0116215644349933
  3.6130034729306777
```

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(z)
scatter!(λ,zeros(5); label = "eigenvalues of A")
```



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
```

```
4.411579696631165e-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
```

```
1.3655250400391074e-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
```

```
1.1961108560189016e-6
```

We can use an ellpise to surround the spectrum:

```
scatter(complex.(eigvals(B)))
plot!(ellipse_rule(50,8,5)[1] .- 20)

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

```
8.393158516734329e-22
```

1.2 Finite differences and heat equation

As a first application we consider approximation of the solution of the heat equation $u(t,x)$:

$$u_t = u_{xx}$$

on $[0,1]$ 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 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

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

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

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

Or in vector form we have

$$\begin{pmatrix} u_x x(t \\ x_1) \\ \vdots \\ u_x x(t \\ x_N) \end{pmatrix} \approx \Delta \begin{pmatrix} u(t \\ x_1) \\ \vdots \\ u(t \\ x_N) \end{pmatrix}$$

where

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

Here we consider two cases

```
N = 1000
h = 1/N
Δ = SymTridiagonal(fill(-2,N), fill(1,N-1))/h^2
eigvals(Δ)

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

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