Solving Stiff PDEs using ETDRK4

18.303 Final Project

Arvid Lunnemark

May 20, 2019

1 Background

1.1 Introduction

The Fourth-Order Exponential Time Differencing Runge Kutta (ETDRK4) scheme was originally developed by Cox and Matthews [1]. It is a method for solving PDEs that have both linear and nonlinear parts, and is especially useful if the linear part is stiff. The original scheme had problems with numerical stability, and Kassam and Trefethen [2] proposed a modified ETDRK4 scheme to solve that problem.

In this paper, I will examine Kassam and Trefethen's ETDRK4 scheme for solving stiff PDEs. The first section will define the set of PDEs we want to solve, define the method and examine the numerical stability. The second section will present six plots using a Julia implementation of the method, in order to show that the method is correct and accurate. Finally, the third section will discuss these plots.

1.2 The Setup

We will solve PDEs on the form

$$u_t = \mathcal{L}u + \mathcal{N}(u,t),$$

where $\mathcal{L}$ is a linear operator and $\mathcal{N}(u,t)$ a nonlinear function. Many PDEs can be written in this form, including (and this is by no means an exhausting list): the heat equation with a forcing term, the KdV equation, the Burgers equation, the KS equation, and the Allen-Cahn equation.

To solve this PDE, we will discretize in space. Normally, if the boundary conditions allow for it, we will use a spectral method such as Fourier or Chebyshev series; in other cases, we might use finite differencing. In any case, we will end up with a system of ODEs on the form

$$u_t = \mathbf{L}u + \mathbf{N}(u,t).$$

If the discretization is in Fourier space, $\mathbf{L}$ will be diagonal and will sometimes have small eigenvalues, contributing to the stiffness.

1.3 The ETDRK4 Scheme

The idea behind the ETD scheme is similar to that of the integrating factor scheme. We rewrite our equation as follows:

$$\begin{aligned} u_t = \mathbf{L}u + \mathbf{N}(u,t) &\implies u_t - \mathbf{L}u = \mathbf{N}(u,t) \\ &\implies e^{-\mathbf{L}t}(u_t - \mathbf{L}u) = e^{-\mathbf{L}t}\mathbf{N}(u,t) \\ &\implies \frac{\partial e^{-\mathbf{L}t}u}{\partial t} = e^{-\mathbf{L}t}\mathbf{N}(u,t) \\ &\implies e^{-\mathbf{L}t}u = u_0 + \int_0^t e^{-\mathbf{L}\tau}\mathbf{N}(u(\tau),\tau)d\tau \\ &\implies u(t) = e^{\mathbf{L}t}u_0 + e^{\mathbf{L}t}\int_0^t e^{-\mathbf{L}\tau}\mathbf{N}(u(\tau),\tau)d\tau. \end{aligned}$$

This equation is exact. To evaluate the integral, however, we already need to know u, and to overcome that problem we compute the solution incrementally. In other words, ETD computes values u_n which are made to approximate $u_n \approx u(nh)$ (where h is the timestep). It then uses u_n as the initial condition in the above integral equation, to get an approximation for u_{n+1}:

$$u_{n+1} = e^{\mathbf{L}h}u_n + e^{\mathbf{L}h}\int_0^h e^{-\mathbf{L}\tau}\mathbf{N}(u(t_n+\tau),t_n+\tau)d\tau,$$

where $t_n = nh$.

Note again that if u_n is the true solution, then the above equation gives an exact true solution for u_{n+1}. The ETDRK4 method approximates the above integral using a fourth-order Runge Kutta method. Accodring to Cox and Matthews, these formulas have been verified by a symbolic manipulation system. The update equation above turns into the following set of equations:

$$\begin{aligned} a_n &= e^{\mathbf{L}h/2}u_n + \mathbf{L}^{-1}\left(e^{\mathbf{L}h/2} - \mathbf{I}\right)\mathbf{N}\left(u_n, t_n\right), \\ b_n &= e^{\mathbf{L}h/2}u_n + \mathbf{L}^{-1}\left(e^{\mathbf{L}h/2} - \mathbf{I}\right)\mathbf{N}\left(a_n, t_n + h/2\right), \\ c_n &= e^{\mathbf{L}h/2}a_n + \mathbf{L}^{-1}\left(e^{\mathbf{L}h/2} - \mathbf{I}\right)\left(2\mathbf{N}\left(b_n, t_n + h/2\right) - \mathbf{N}\left(u_n, t_n\right)\right), \\ \alpha &= h^{-2}\mathbf{L}^{-3}\left[-4 - \mathbf{L}h + e^{\mathbf{L}h}\left(4 - 3\mathbf{L}h + (\mathbf{L}h)^2\right)\right], \\ \beta &= h^{-2}\mathbf{L}^{-3}\left[2 + \mathbf{L}h + e^{\mathbf{L}h}(-2 + \mathbf{L}h)\right], \\ \gamma &= h^{-2}\mathbf{L}^{-3}\left[-4 - 3\mathbf{L}h - (\mathbf{L}h)^2 + e^{\mathbf{L}h}(4 - \mathbf{L}h)\right], \\ u_{n+1} &= e^{\mathbf{L}h}u_n + \alpha\mathbf{N}(u_n, t_n) + 2\beta\left[\mathbf{N}(a_n, t_n + h/2) + \mathbf{N}(b_n, t_n + h/2)\right] + \gamma\mathbf{N}(c_n, t_n + h) \end{aligned}$$

Note that α, β and γ are independent of time and need only be evaluated once, at the beginning of the scheme.

1.4 Numerical Stability

While the ETDRK4 scheme is fourth-order, it suffers from numerical instability if evaluated naïvely. The coefficients α, β and γ are namely analogous to

$$g(z) = \frac{e^z - 1}{z},$$

which, when z is very small, suffers from cancellation errors. This can be solved by using the Taylor approximation instead of the explicit formula for small values, but Kassam and Trefethen

claim that this is not an optimal idea for two reasons: it is hard to know the exact cutoff point for when to use the Taylor expansion and when to use the explicit formula, and there might also be a region where neither Taylor expansions nor the explicit formula yield satisfactory results; also, to work with non-diagonal $\mathbf{L}$ with both small and big eigenvalues, the same method must be able to handle both. They propose another way of evaluating α, β, γ, by using contour integration:

$$f(\mathbf{L}) = \frac{1}{2\pi i} \int_\Gamma f(t)(t\mathbf{I} - \mathbf{L})^{-1} dt,$$

where f is a function with a removable singularity at 0, and the contour Γ encloses all eigenvalues of $\mathbf{L}$ and is well separated from 0. In practice, when we use this way of evaluating α, β, γ, we will let Γ be a circle, and evaluate the integral on that. The evaluation becomes particularly simple if $\mathbf{L}$ is diagonal, in which case we can evaluate the above separately for each eigenvalue and consequently choose Γ differently based on each eigenvalue. Kassam and Trefethen show that using the contour integral to evaluate this set of functions is highly numerically stable both for small and large eigenvalues, which makes it a particularly useful tool for when we are using a spectral method.

2 Results

The ETDRK4 scheme was implemented in Julia, and tested on (1) the Kuramoto-Sivashinsky (KS) equation, and (2) the Korteweg–de Vries (KdV) equation. This section includes a number of plots based on these two implementations. All code producing these plots can be found on GitHub [3].

The KS equation in Figure 1 is on the form

$$u_t = -uu_x - u_{xx} - u_{xxxx}$$

where $x \in [0, 32\pi]$ and t goes from 0 to 150, with periodic boundary conditions. The initial condition used is

$$u(x, 0) = \cos(x/16)(1 + \sin(x/16))$$

and the spatial discretization is based on Fourier series with $N = 128$. The used timestep is $h = 0.25$. The solution ran in less than a second.

The KdV equation in Figure 2 is on the form

$$u_t = -uu_x - \delta^2 u_{xxx}$$

where $\delta = 0.022$, $x \in [0, 2]$ and t goes from 0 to 2, with periodic boundary conditions. The initial condition used is

$$u(x, 0) = \cos(\pi x)$$

and the spatial discretization is based on Fourier series with $N = 1024$. The used timestep is $h = 0.001$. The solution ran in less than a second.

The same KdV equation with the same initial and boundary conditions was also solved using the DifferentialEquations.jl package [4]. To make the latter method run in reasonable time (less than a second), N was decreased to 128. Figure 3 shows both solutions at time 1. The timestep for ETDRK4 was then decreased to $h = 0.01$ in Figure 4, showing that it diverges from the true solution at that point.

The ETDRK4 scheme is a fourth-order scheme, and to validate that that is actually the case, the method was run for different timesteps h on the KS equation. While I know of no exact analytic solution to the equation, the exact solution was approximated using a timestep half as

big as the smallest of the analyzed timesteps (also using ETDRK4). Then, for each different timestep, the relative ∞-norm error was computed:

$$\epsilon = \frac{\max_{x_i}(|u_{\text{exact}}(x_i) - u_{\text{etdrk4}}(x_i)|)}{\max_{x_i}(|u_{\text{exact}}(x_i)|)},$$

which is what is shown in Figure 6 for different values of h.

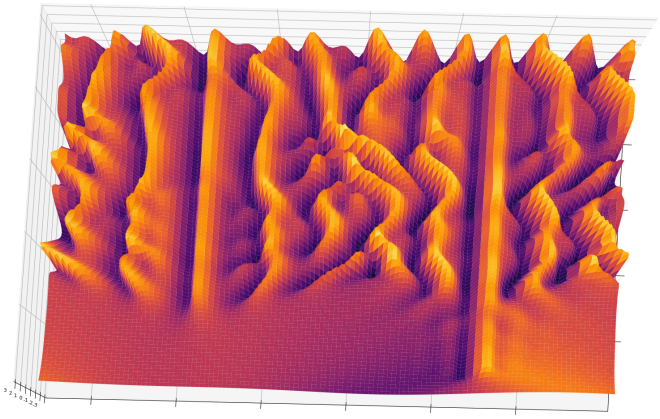

Figure 1: *Solution to the KS equation, with time evolving from 0 in the bottom of the figure to 150 at the top. The spatial dimension is from 0 to 32π horizontally. The plot is produced by* `figure1.jl` *in [3].*

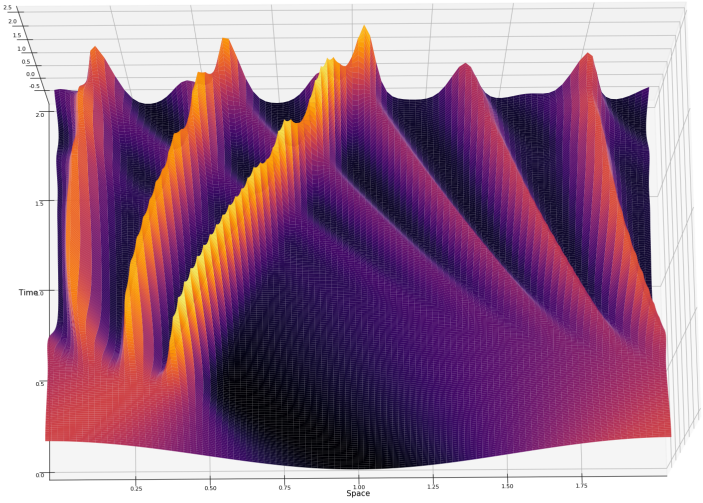

Figure 2: *Solution to the KdV equation, with time evolving from 0 in the bottom of the figure to 2 at the top. The spatial dimension is from 0 to 2 horizontally. The plot is produced by* `figure2.jl` *in [3].*

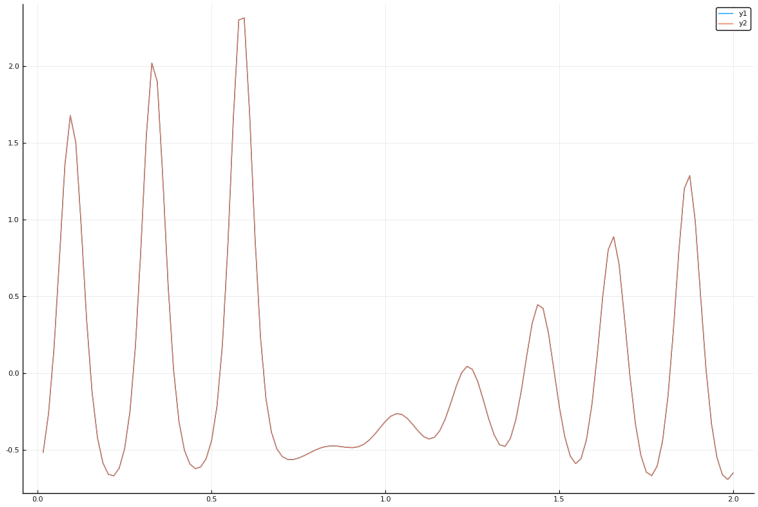

Figure 3: *The solution to the KdV equation at time 1, using both my implementation of ETDRK4 (blue) with* $h = 0.001$ *and a simple implementation using DifferentialEquations.jl (red). Up to the plot resolution, the solutions are identical, which is why only the red plot can be seen. The plot is produced by* `figure3.jl` *in [3].*

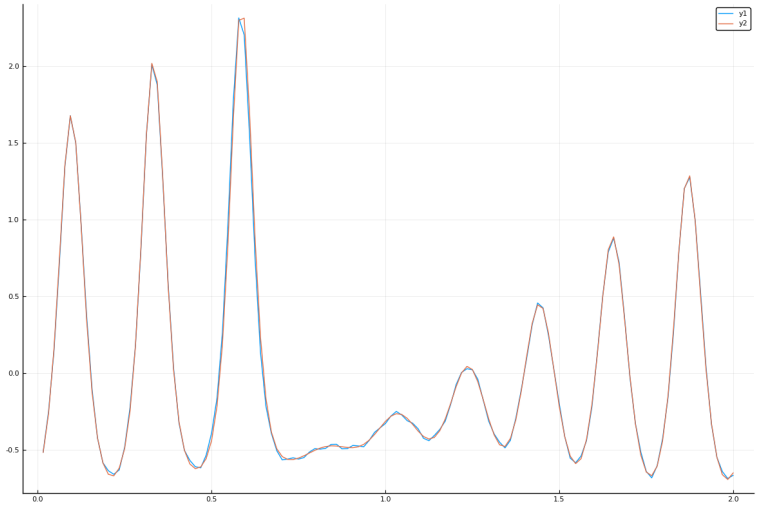

Figure 4: *The solution to the KdV equation at time 1, using both my implementation of ETDRK4 (blue) with* $h = 0.01$ *and a simple implementation using DifferentialEquations.jl (red). Here we can see that ETDRK4 diverges from the true solution. The plot is produced by* `figure4.jl` *in [3].*

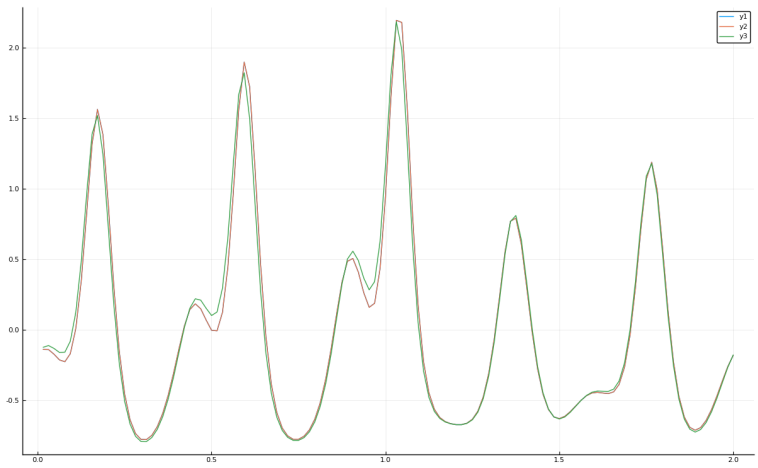

Figure 5: *The solution to the KdV equation at time 2, using both my implementation of ETDRK4 (blue) with* $h = 0.002$ *and a simple implementation using DifferentialEquations.jl (red). In green is the ETDRK4 with a small radius on the contour integration, diverging from the two other solutions. The plot is produced by* `figure5.jl` *in [3].*

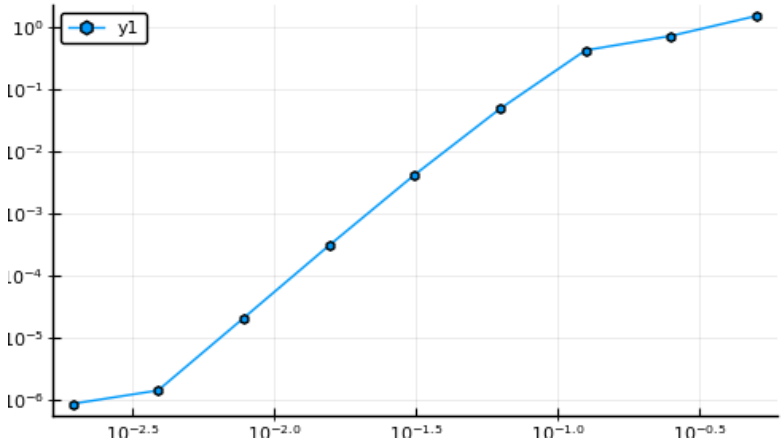

Figure 6: *The ETDRK4 method used on the KS equation with different timesteps* h*. The* x *axis corresponds to the timesteps and the* y *axis corresponds to the* ∞*-norm relative error against a more accurate solution. The plot is produced by* `figure6.jl` *in [3].*

3 Discussion

3.1 KS Equation

The KS equation was used in order to validate my Julia implementation, since Kassam and Trefethen implemented ETDRK4 on the KS equation with the same boundary and initial conditions. As can be seen in Figure 1, the solution is identical to the one in Kassam and Trefethen, which is reassuring for the application of the method on the KdV equation.

In Figure 6, we can see that the slope of the graph is roughly 4 (disregarding the outliers at the edges). This confirms that the ETDRK4 method is indeed a fourth order method, since

$$\frac{\log(\epsilon)}{\log(h)} \approx 4 \implies \epsilon \approx h^4.$$

3.2 KdV Equation

In the plots for the KdV equation, we see that the solution evolve as we would expect it to. This particular KdV was studied by Zabusky and Kruskal [5] who got similar results. We note how the solitons move more or less independently of each other, even as they pass through other solitons.

When using the DifferentialEquations.jl package to solve the equation, we note that we got the same result as when using ETDRK4 with timestep $h = 0.001$. Note that the implementation of ETDRK4 was significantly faster than the package implementation when more Fourier modes were used (1024 instead of 128), which is promising. Figure 4 shows how the solution using ETDRK4 might not be stable if too small timesteps are used, however, which is an important thing to note.

3.3 Numerical Stability

For all plots except Figure 5, Kassam and Trefethen's way of solving the numerical instability by using contour integration was used. In Figure 5, the issue with numerical instability is clear: the radius of the contour was decreased from 1 to 0.000001, which effectively means that the explicit formula was used in the latter case, and as we can see in the figure that change significantly impacted the accuracy.

4 Conclusion

The ETDRK4 method explored in this paper, outlined by Kassam and Trefethen, proved successful for solving both the KS equation and the KdV equation. I verified that my Julia implementation was working properly, and that it indeed has a convergence of order four, as expected. The contour integration method for computing the coefficients α, β, γ was also proved to be effective (as well as easier to code, since eigenvalues of zero are no longer an edge case).

References

[1] Steven M Cox and Paul C Matthews. "Exponential time differencing for stiff systems". In: *Journal of Computational Physics* 176.2 (2002), pp. 430–455.

[2] Aly-Khan Kassam and Lloyd N Trefethen. "Fourth-order time-stepping for stiff PDEs". In: *SIAM Journal on Scientific Computing* 26.4 (2005), pp. 1214–1233.

[3] Arvid Lunnemark. *Julia Implementation of ETDRK4*. 2019. URL: `https://github.com/arvid220u/18303-final-project`.

[4] C. Rackauckas and Q. Nie. "DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia". In: *Journal of Open Research Software* 5 (2017), p. 15.

[5] Norman J Zabusky and Martin D Kruskal. "Interaction of solitons in a collisionless plasma and the recurrence of initial states". In: *Physical review letters* 15.6 (1965), p. 240.