AMSI VACATION RESEARCH SCHOLARSHIPS 2019–20

EXPLORE THE MATHEMATICAL SCIENCES THIS SUMMER

Computational modelling using exponential integrators

Joshua Wilson

Supervised by Prof. Timothy Moroney and Dr Elliot Carr

Queensland University of Technology

Vacation Research Scholarships are funded jointly by the Department of Education and the Australian Mathematical Sciences Institute.

Abstract

In the recent past, a new method for temporal discretisation of spatially-discretised partial differential equations has arisen, exponential integrator methods. In this report, we investigate and implement an explicit exponential integrator scheme that is able to solve semi-linear problems that arise from PDEs like reaction-diffusion equations. The developed codebase was determined to provide comparable accuracy to in-built solvers available in MATLAB, with reasonable efficiency.

1 Introduction

A major source of interest in computational mathematics is the ability to numerically evaluate solutions to *partial differential equations* in reasonable time. Partial differential equations (PDEs) are a class of equations that relate some set of partial derivatives and values to each other (a simple example of a PDE defined in one dimension of space x and time t can be seen below in Equation 1).

$$f\left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial t}, u, x, t\right) = 0. \tag{1}$$

A common method for numerically evaluating such expressions is to discretise them in both space and time, in order to treat the problem as distinct cells and timesteps. For *reaction-diffusion* problems, those involving the interaction and diffusion of a particular property, this discretisation often results in a reduction to a semi-linear problem, expressible in the form:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \mathbf{A}u + g(u) \tag{2}$$

The constant matrix $\mathbf{A}$ tends to represent the stiffness of the problem, resulting from the conservation equation governing the flow of a given property through the domain, whilst the vector function $\vec{g}$ represents any non-linearities, resulting from any diffusion, sources, and sinks.

There is broadly two classes of solution methods to be considered when determining how best to solve problems such as these: explicit and implicit timestepping methods. Explicit methods are easy to compute, as they wholly use information currently available in the system in order to determine the state of the system at the next timestep. This comes at a cost,

however, as these methods tend to have stability issues, with errors incurred as a result of approximations compounding whilst stepping across the time domain. As such, explicit methods require much smaller timesteps in order to minimise the effect of this compounding error, to maintian the same degree of global accuracy. In contrast, implicit methods have superior forward stability, resulting from the relation of current and future information. They are, however, more computationally expensive, as they involve some form of 'solve' in order determine what value this future information must have in order to reflect the nature of the system.

Exponential integrator schemes arise as an attempt to take the best from both of these methods. The scheme considered in this project is an explicit scheme, and so removes the need for any form of solve. It also has various properties that make it a desirable choice for stiff equations like those being considered, such as exact treatment of the linear portion, and resulting forward stability (Al-Mohy and Higham, 2011).

1.1 Statement of Authorship

Over the course of the project:

- Joshua Wilson developed the solution code in MATLAB, tested and analysed its effectiveness, and wrote this report and a presentation on the method.
- Prof. Timothy Moroney and Dr Elliot Carr provided the theory underpinning the solution, supervised and advised on its creation and testing, and proofread the material produced by Joshua.

2 Solution Method

As a part of this research, we investigated the possibilities surrounding the construction of a solution scheme to semi-linear problems using exponential integrators. This was done in a modular fashion, gradually adjusting and adding to previous stages of the scheme over the course of the project.

2.1 Exponential Integrator Scheme

The foundation of the solution method is the following exponential integrator scheme that advances the solution to our semi-linear problem in discrete timesteps (Al-Mohy and Higham, 2011):

$$u_{n+1} = \exp(\tau \mathbf{A}) u_n + \sum_{k=1}^{p} \tau^k \varphi_k(\tau \mathbf{A}) {g_n}^{(k-1)}. \tag{3}$$

where u_{n+1} is the updated solution profile at each timestep, τ is the current timestep size, and ${g_n}^{(k-1)}$ is the $(k-1)^{\text{th}}$ derivative of g, evaluated at the current timestep.

The scheme involves exactly solving for linear component of the semi-linear problem, and approximating the non-linear component by evaluating a set of derivatives in tandem with the evaluation of a set of matrix φ-functions. In this way, higher order schemes could be obtained by using more derivatives and evaluations of the matrix φ-functions.

When constructing the solution method, the matrix exponential can be calculated using the in-built MATLAB function `expm` [1], whilst the matrix φ-functions can be calculated according the the recurrence relation (Al-Mohy and Higham, 2011):

$$\varphi_{\ell+1}(\mathbf{z}) = \mathbf{z}^{-1}\left(\varphi_\ell(\mathbf{z}) - \frac{\mathbf{I}}{\ell!}\right), \ \varphi_0(\mathbf{z}) = \exp(\mathbf{z}). \tag{4}$$

In order to implement the scheme (3) for higher orders, the derivatives of the function g need to be approximated numerically at each timestep. An algorithm was constructed based on an algorithm developed by (Schneider and Werner, 1986) that uses a set of past data in order to approximate the derivatives of a function at its most recent point. The MATLAB code that implements this algorithm can be found in Appendix A.

2.2 Krylov Subspace Approximations

It is known that for the scaling and squaring operation that MATLAB performs as a part of the `expm` routine, the computation scales as $\mathcal{O}(n^3)$ (Moler and Van Loan, 2003). As such, it is greatly desirable to only need to compute matrix exponential and matrix φ-functions for

[1] https://www.mathworks.com/help/matlab/ref/expm.html

matrices of minimal size. Given the sparse nature of $\mathbf{A}$, arising from spatial discretisation, a popular choice for reducing the amount of computation required is to use a projection method. In particular, Arnoldi's method can be used to project the required matrix functions onto the Krylov subspace $\mathcal{K}_m(\tau\mathbf{A}, b)$ for the required vector b (Saad, 2003). This yields the approximation shown below (Hochbruck, Lubich, and Selhofer, 1998) (Carr, Moroney, and Turner, 2011):

$$\varphi_\ell(\tau\mathbf{A})b \approx \beta_0 V_m \varphi_\ell(\tau H_m)e_1, \tag{5}$$

where $\beta_0 = ||b||_2$, and V_m and H_m are matrices produced as a result of the Arnoldi method (Saad, 2003). As can be seen by this approximation, for $m|||N$, the computational impact that these matrix functions have on the runtime of the program can be greatly reduced (as a trade-off for accuracy).

It is important to keep in mind, however, that using a fixed subspace size M would be a poor choice overall. Depending on the complexity of behaviour displayed by the system at different timesteps, it is likely that we would frequently over or underdefine the subspace approximating it. Instead, a particular exit criterion was chosen, based on a criterion previously used in a matrix φ-function only scheme (Carr, Moroney, and Turner, 2011):

$$\left|\left|\beta_0\beta_m \left[{e_m}^T \varphi_\ell(\tau H_m)e_1\right] v_{m+1}\right|\right|_{\text{WRMS}} < 1, \tag{6}$$

where $||\cdot||_{\text{WRMS}}$ is defined as (Carr, Moroney, and Turner, 2011):

$$||y||_{\text{WRMS}} = \left(\frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i}{w_i}\right)^2\right)^{1/2}, \tag{7}$$

with $w_i = \text{tol}_r \cdot u_{n,i} + \text{tol}_a$, where $u_{n,i}$ is the i^{th} solution component at time $t = t_n$, and tol_r and tol_a are the relative and absolute tolerances, respectively.

2.3 Compact Evaluation

It can be seen quite readily that as the order of the scheme (3) is increased, an increasing number of Krylov subspaces must be created to match. This is obviously undesirable, and so obtaining a method that would require fewer matrix operations is ideal. A key result from

Al-Mohy and Higham, 2011 is the construction of an augmented matrix that, when the matrix exponential is applied to it, reproduces the scheme 3 whilst only requiring a single operation of the matrix exponential.

$$\tilde{\mathbf{A}} = \left[\begin{array}{ccc|c|ccc} & \mathbf{A} & & g_n^{(p-1)} & g_n^{(p-2)} & \cdots & g_n^{(0)} \\ \hline 0 & \cdots & 0 & 0 & & & \\ \vdots & & \vdots & \vdots & & \mathbf{I}_{p-1} & \\ \vdots & & \vdots & \vdots & & & \\ 0 & \cdots & 0 & 0 & \cdots & \cdots & 0 \end{array}\right] \quad \tilde{u}_n = \left[\begin{array}{c} u_n \\ \hline 0 \\ \vdots \\ 0 \\ \hline 1 \end{array}\right] \tag{8}$$

Using $\tilde{\mathbf{A}}$ and $\tilde{u}_n$ as shown above in Equation 8, it is known that (Al-Mohy and Higham, 2011):

$$\left[\exp(\tau\tilde{\mathbf{A}})\tilde{u}_n\right]_{1:N} = \exp(\tau\mathbf{A})u_n + \sum_{k=1}^{p} \tau^k \varphi_k(\tau\mathbf{A}) g_n^{(k-1)} \tag{9}$$

where $[\cdot]_{1:N}$ represents taking the first N components of a vector. This can easily be seen to be recovered form of the desired scheme (3).

2.4 Adaptive Timestepping

It is well known that for solution schemes involving discrete timesteps, the accuracy of each step is closely linked to the size of the timestep taken. As such, it is desirable to be able to variably adjust the size of each timestep such that it appropriately matches the level of accuracy required. In order to perform this adjustment, it is necessary to come up with some method of predicting the level of error incurred by each timestep. Whilst the stability concerns of explicit methods are often glaring enough to discourage their use as a main timestepping method (as discussed in Section 1), they form a good basis for a prediction method, given their relative cheapness to compute.

One would expect that when comparing the exponential integrator scheme and an explicit scheme of the same order, the difference between them should be indicative of the error incurred by advancing the solution using that timestep size. In this case, the Adams-Bashforth

family of methods were chosen as they form a logical extension of order to the forward Euler method (Butcher, 2016). These methods are generally given as fixed timestep methods, but by examining their initial derivation (Bashforth and Adams, 1883), they can be logically extended to account for a varying stepsize. Whilst the methods condense down very nicely for a fixed timestep, the number and complexity of terms explodes quite quickly as the order is increased, and so only the first and second order forms are given, respectively, as:

$$u_{n+1} = u_n + \tau_{n+1} f(u_n, t_n) \tag{10}$$

$$u_{n+1} = u_n + \left(\frac{{\tau_{n+1}}^2}{2\tau_n} + \tau_{n+1} \right) f(u_n, t_n) - \left(\frac{{\tau_{n+1}}^2}{2\tau_n} \right) f(u_{n-1}, t_{n-1}) \tag{11}$$

where $f(u_n, t_n) = \frac{\mathrm{d}u}{\mathrm{d}t}\big|_{t=t_n}$, $f(u_{n-1}, t_{n-1}) = \frac{\mathrm{d}u}{\mathrm{d}t}\big|_{t=t_{n-1}}$, and τ_n refers to the timestep size used to advance from timestep $n-1$ to timestep n.

Using this method, we can construct our predictor as the WRMS norm (see Equation 7) of the difference between the Adams-Bashforth and exponential integrator schemes, using the same tolerances as for the exit criterion of our Krylov subspace (6). Using this predictor value, we can construct a simple timestepping scheme: if the WRMS predictor exceeds a reasonable tolerance level then the timestep will be halved and re-run. This forces the timestep quickly down until it is within reasonable bounds, so as not to waste excess computation. If, instead, the predictor drops below a particular efficiency margin for several consecutive timesteps, then the timestep is multiplied by a small factor. This allows the timestep to gradually be increased again if the predictor indicates that computation is wasted on evaluating the solution to higher accuracy than is required. In this way, the system can be evaluated over a time domain, with timesteps variably chosen such that the required level of tolerance is maintained throughout the lifetime of the solution.

3 Application to Test Problem

In order to evaluate the effectiveness of the produced codebase, it was necessary to apply it to a test problem. Throughout the course of the project, the exponential integrator scheme was compared to an in-built MATLAB solver for stiff equations, `ode15s` [2]. Various options

[2] https://www.mathworks.com/help/matlab/ref/ode15s.html

can be selected within MATLAB in order to allow the behaviour of this solver to mirror the approach we take with the exponential integrator solver (such as order, timestepping method, error control, etc). This allowed a direct way to evaluate the performance and ability of the exponential integrator solver against an industry-level implicit solver.

3.1 Fisher's Equation

Fisher's equation[3] is a particular reaction-diffusion equation that was chosen for use as a test problem due to its simplicity and clear solution type. When the correct coefficients are chosen, the equation presents a travelling wavefront as a solution. It is given in one-dimension as:

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + u(1-u) \tag{12}$$

For the purposes of these tests, the spatial domain was fixed as $x \in [0, 1]$, with $D = 0.001$ and boundary conditions $u(0, 0) = 1$ and $\frac{\partial u}{\partial x}\big|_{x=1} = 0$. The initial condition was given as $u(x, 0) = 0$ for $x \neq 0$. These particular choices were made into order to elucidate a clear travelling wave in the solution, such that its representation could be visually compared between methods.

This continuous PDE needs to first be spatially discretised in order to reduce it to a system of ODEs for use with the aforementioned solution schemes. A popular technique for this discretisation is the finite volume method (FVM), an approach which involves describing the behaviour of the system by a set of discrete connected nodes that represent a control volume surrounding it (Versteeg and Malalasekera, 2007). By integrating across these control volumes, and applying various approximations, these nodes can accurately represent the dynamics of the system via discrete interactions. It is this technique that was applied in order to transform Fisher's equation into a semi-linear system of ODEs for implementation.

3.2 Results

The image shown in Figure 1 demonstrates the solution profile given the parameters chosen for the test problem, calculated using the exponential integrator scheme and `ode15s` (represented by the points and lines, respectively). With the different coloured curves from left to right representing the solution after 0, 10, 20, 30, 40, and 50 seconds of simulated time, it can clearly

[3] http://mathworld.wolfram.com/FishersEquation.html

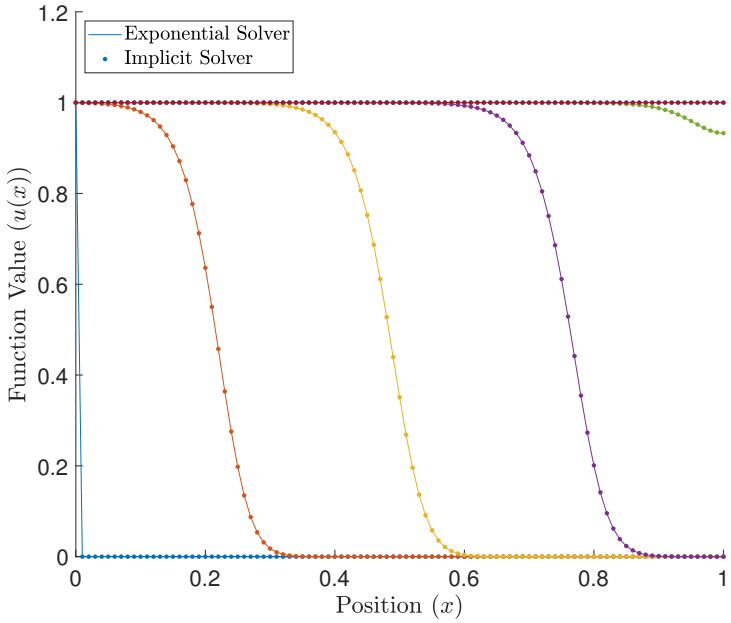
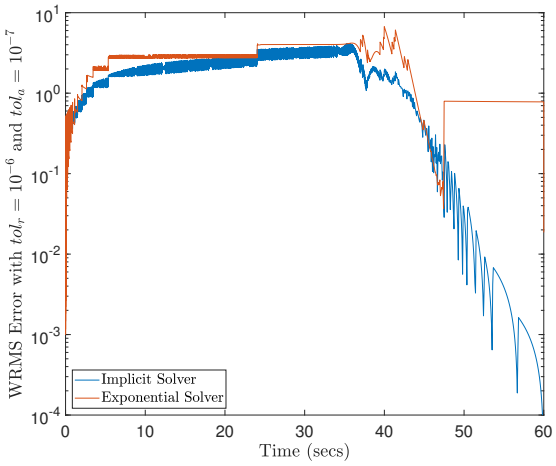


Figure 1: Solution profiles for the test problem with curves representing the profiles at 0, 10, 20, 30, 40, and 50 seconds (left to right)

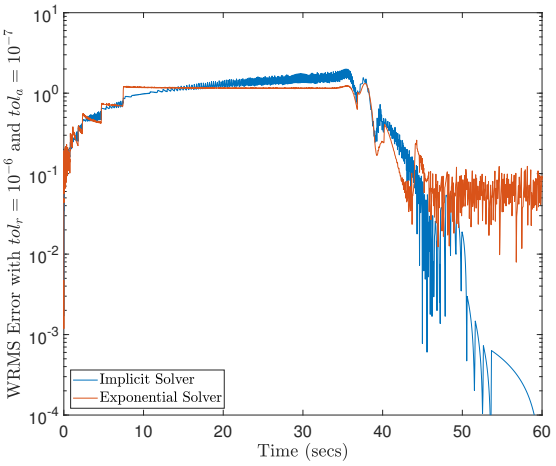
be seen that the developed exponential integrator scheme closely agrees with the solution produced by `ode15s`. Figure 1 was generated using the highest order scheme representable by `ode15s` (for accuracy), which was order 5.

In order to evaluate the degree of accuracy to which the solution was evaluated, it was necessary to devise some way to approximate a 'true' solution to compare both the exponential integrator and `ode15s` solutions against. In order to do so, each of the two solution methods were first run individually with the same tolerance and order, each generating their own set of timesteps. With this done, each of these timesteps was then individually taken and run to machine precision accuracy using `ode15s`, with the eventual solution after each timestep being compared to that which was produced by its method. This allowed the accuracy of each timestep to be represented using a WRMS norm of the error of each timestep against the reference solution. A plot of some of the key orders for comparison can be seen in Figure 2.

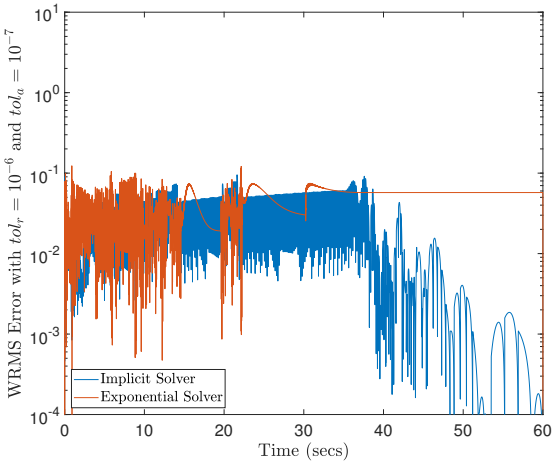
As can be seen in Figures 2a – 2c, the level of error incurred by the exponential integrator



(a) 1st Order Comparison



(b) 2nd Order Comparison



(c) 5th Order Comparison

Figure 2: Comparison of timestepping error for exponential and `ode15s` schemes

scheme at each timestep is wholly comparable to that of `ode15s`. As the order increases, however (as shown in Figure 2c), the level of error incurred drops below requested tolerance, implying that the solution has been solved to a higher degree of accuracy than required at higher orders, wasting computation. This is likely due to the nature of the predictor method, which has not been finetuned with any form of heuristics to close the gap between it and the generated timesteps.

4 Future Investigation

In the future, more investigation could be done concerning:

- **Intelligent stepsize heuristics** - Rather than simply changing the timestep up and down based on the accuracy predictor, further investigation could be made into heuristics which can adaptively select a stepsize that will meet the required level of accuracy with little to no overcomputation. This would allow the scheme to run more efficiently, by taking as large timesteps as possible whilst still maintaining the required accuracy.

- **Variable order methods** - Whilst only adaptively varying the timestep of the scheme was considered, it is also be possible to vary the order of solution in a similar fashion. By combining these two techniques effectively, the overall efficiency could be increased with extra computation not being wasted on unneccesary levels of accuracy.

- **Wider testing** - Once the solution scheme has been completely finalised, many more problems should be used to test it. In order to produce a truly meaningful codebase, it would need to be tested and validated against many different kinds of behaviours exhibited by semi-linear problems derived from PDEs, which a single test problem cannot provide.

- **Runtime analysis** - Finally, the efficiency of the solution scheme should be analysed, to see how it compares to state of the art solvers for semi-linear PDEs. The end goal of further investigation into this topic would be to produce a solution method that has comparable efficiency to other mainstream solvers currently in use. In order to determine whether that goal has been achieved, rigorous testing of multiple problems over various problem sizes would need to be conducted.

5 Conclusion

Throughout the course of this project, we have investigated the use of a particular exponential integrator scheme in evaluating numerical solutions to semi-linear problems arising from spatial discretisation of reaction-diffusion equations. We have investigated various relevant approximations and optimisations for this scheme, with the goal of improving the efficiency of its

evaluation without losing out on accuracy in any meaningful way. This has involved producing a codebase that is capable of solving these semi-linear problems with comparable accuracy to an in-built solver for stiff equations in MATLAB, `ode15s`. In the future, there are further avenues to pursue in order to increase this efficiency even further, before ultimately providing rigorous testing of the codebase in order to evaluate its performance in comparison to industry-standard solvers for these types of problems.

References

Bashforth, F. and Adams, J.C., 1883. *An attempt to test the theories of capillary action: by comparing the theoretical and measured forms of drops of fluid*. University Press.

Butcher, J.C., 2016. *Numerical methods for ordinary differential equations*. John Wiley & Sons, pp.111–116.

Carr, E.J., Moroney, T.J., and Turner, I.W., 2011. Efficient simulation of unsaturated flow using exponential time integration. *Applied mathematics and computation*, 217(14), pp.6587–6596.

Hochbruck, M., Lubich, C., and Selhofer, H., 1998. Exponential integrators for large systems of differential equations. *Siam journal on scientific computing*, 19(5), pp.1552–1574.

Al-Mohy, A.H. and Higham, N.J., 2011. Computing the action of the matrix exponential, with an application to exponential integrators. *Siam journal on scientific computing*, 33(2), pp.488–511.

Moler, C. and Van Loan, C., 2003. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *Siam review*, 45(1), pp.3–49.

Saad, Y., 2003. *Iterative methods for sparse linear systems*. Vol. 82. siam, pp.157–162.

Schneider, C. and Werner, W., 1986. Some new aspects of rational interpolation. *Mathematics of computation*, 47(175), pp.285–299.

Versteeg, H.K. and Malalasekera, W., 2007. *An introduction to computational fluid dynamics: the finite volume method*. Pearson education.

Appendix A numDiff.m

```
function DF = numDiff(x, F, n)
if n >= length(x)
        error('Trying to compute derivatives higher than the data
            allow.')
elseif n == 0
        DF = F(:, 1);
else
        DF = zeros(size(F,1), n);
        dx = x(:) - x(:)' + eye(length(x));
        w = 1 ./ prod(dx);
        xdiff = dx(1, 2:end);
        kfact   = 1;
        divdiff = F(:, 2:end);
        deriv   = F(:, 1);
        for k = 1:n
                divdiff = (deriv / kfact - divdiff) ./ xdiff;
                kfact   = kfact * k;
                deriv   = -kfact * sum(w(2:end) .* divdiff, 2) /
                   w(1);
                DF(:,k) = deriv;
        end
        DF = [F(:, 1), DF];
end
end
```

Figure 3: Evaluating numerical derivatives (Schneider and Werner, 1986)