Undergraduate Research Project Report

Khang Ee Pang[1], Lennon Ó Náraigh[1], and Andrew Gloster[1]

[1]School of Mathematics and Statistics, University College Dublin, Belfield, Dublin 4

July 24, 2018

1 UTSD equation

When a weak shock ($M \to 1+$) reflects off a thin wedge ($\theta \to 0+$), it was observed that the incident and reflected shock produces a Mach stem near the contact with the slope where the incident shock, reflected shock and Mach stem form a triple point configuration. von Neumann (1943) shows that the triple point configuration is not possible for sufficient weak shock, hence the von Neumann triple point paradox.

In 2000, Hunter and Brio obtained a numerical solution to what now known as the Unsteady Transonic Small-Disturbance (UTSD) equation. The simulation suggests that there is a supersonic patch that sustains the triple point configuration and therefore resolves the paradox. Their simulation coincides with the claim of Guderly (1962) that had been lack of support up until this result is published. Their simulation estimates a size for the supersonic patch, which is extremely small, to advocate for experimental observation. The supersonic patch was soon confirmed experimentally by Skews and Ashworth (2005).

The UTSD equation is a simpler case of a larger family of the Riemann problem which is very useful for understanding shocks. However, the theory for Riemann problems is poorly understood due to its complex non-linearity [2]. Therefore we resolve to numerical analysis. The study of the UTSD equation could hopefully give us more insight to understanding the Riemann problem.

In Hunter and Brio's original paper, the approach for solving the UTSD equation is to discretize the x, y, and t derivatives directly. This method, however, does not support parallelization as each point on the grid depends heavily on its neighbours and the points from previous time stepping. In the first part of the project, we look at an alternative method for solving the UTSD equation to support parallel computing in order to achieve greater computation speed.

Problem setup

We look at the following Unsteady Transonic Small-Disturbance (UTSD) equation

$$\frac{\partial^2 u}{\partial x \partial t} + \frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \qquad t > 0, \qquad (x, y) \in \Omega, \tag{1}$$

where F is the flux

$$F(u) = \frac{1}{2}u^2,$$

over a domain $\Omega = (-L, L) \times (0, L_y)$ with boundary condition

$$u_y(x, y = 0, t) = 0, \qquad u(x, y = L_y, t) = b(x), \qquad u(x = L_x, y, t) = 0, \tag{2}$$

and initial condition

$$u(x, y, t = 0) = \begin{cases} 0 & x > ay, \\ 1 & x \leq ay. \end{cases}$$

To solve this neumerically, we first discretize the UTSD equation in time with $t = n\Delta t$, and $n \in \{0, 1, 2, ...\}$. As such, Equation (1) becomes

$$\frac{\partial}{\partial x}\left(\frac{u^{n+1} - u^n}{\Delta t}\right) + \frac{\partial^2 F^n}{\partial x^2} + \frac{\partial^2 u^{n+1}}{\partial y^2} = 0.$$

We could not discretize the equation further as this will distroy its parallelizability. Therefore we seek for an analytic solution for the expression. Re-arranging the equation yields

$$\frac{\partial u^{n+1}}{\partial x} + \Delta t \frac{\partial^2 u^{n+1}}{\partial y^2} = \frac{\partial \Phi^n}{\partial x},$$

where

$$\Phi^n(x, y) = u^n - \Delta t \frac{\partial F^n}{\partial x}.$$

Thus, at each timestep $t = n\Delta t$, we need to solve a backwards heat equation of the form

$$\frac{\partial \psi}{\partial x} + \kappa \frac{\partial^2 \psi}{\partial y^2} = \frac{\partial \Phi}{\partial x}. \tag{3}$$

The separate the solution into two parts

$$\psi(x, y) = \widetilde{\psi}(x, y) + r(x, y),$$

where $\widetilde{\psi}$ satisfies the equation (3) with homogeneous boundary condition and r satisfies the inhomogeneous boundary condition (2).

Spectral method for solving the backwards heat equation

The the backward heat equation we would like to solve is

$$\frac{\partial \widetilde{\psi}}{\partial x}(x, y) = -\kappa \frac{\partial^2 \widetilde{\psi}}{\partial y^2}(x, y) + \frac{\partial \Phi}{\partial x}(x, y) + Q_{corr}(x, y), \qquad (x, y) \in \Omega, \tag{4}$$

over a domain $\Omega = (-L, L)\times(0, L_y)$ with homogeneous boundary condition and final condition

$$\frac{\partial\widetilde{\psi}}{\partial x}(L, y) = -b(L)f(y),$$

where Φ is given by

$$\Phi(x, y) = u - \kappa\frac{\partial F}{\partial x}(u). \tag{5}$$

We solve this by first applying cosine transform the equation to get

$$\frac{d\widehat{\psi}_n}{dx}(x) = \kappa k_n^2\widehat{\psi}_n(x) + \frac{d\widehat{\Phi}_n}{dx}(x) + \widehat{Q}_{corr,n}(x),$$

and final condition

$$\widehat{\psi}_n(L) = -b(L)\int_0^{L_y}\Psi_n(y)f(y)dy.$$

Solving the ODE via integrating factor method yields

$$\widehat{\psi}_n(x) = \widehat{\psi}_n(L)e^{-\kappa k_n^2(L-x)} - \int_x^L e^{-\kappa k_n^2(x'-x)}\left[\frac{\partial\widehat{\Phi}_n}{\partial x'}(x') + \widehat{Q}_{corr,n}(x')\right]dx', \tag{6}$$

As $F(x, y)$ is not continuous in Ω, naively differentiating $F(x, y)$ will produce numerical oscillation. There are two instances of differentiating F, the first such instance is in equation (5) and the second is in equation (6).

To avoid this, the x derivative in equation (5) is dealt with by applying the Engquist-Osher (EO) scheme,

$$\frac{\partial F_i}{\partial x}(u) = \frac{f^{EO}_{i+\frac{1}{2}} - f^{EO}_{i-\frac{1}{2}}}{\Delta x}, \tag{7}$$

where

$$\begin{aligned} f^{EO}_{i-\frac{1}{2}} &= \frac{1}{2}\max\{u_{i-1} - c, 0\}^2 + \frac{1}{2}\min\{u_i - c, 0\}^2, \\ f^{EO}_{i+\frac{1}{2}} &= \frac{1}{2}\max\{u_i - c, 0\}^2 + \frac{1}{2}\min\{u_{i+1} - c, 0\}^2. \end{aligned}$$

The scheme up to this point along with discontinuity smoothing of $\frac{\partial\Phi}{\partial x}$ was validated using a MATLAB code to solve the UTSD equation (see Figure 1). The fuzziness of the incident shock is cause by the discontinuity smoothing.

The second x derivative could be avoided by applying integration by parts to evaluate the integrand in equation (6). Using this method, the solution becomes

$$\begin{aligned} \widehat{\psi}_n(x) =& e^{-\kappa k_n^2(L-x)}\big(\widehat{\psi}_n(L) - \widehat{\Phi}_n(L)\big) + \widehat{\Phi}_n(x) \\ &- \kappa k_n^2\int_x^L e^{-\kappa k_n^2(x'-x)}\widehat{\Phi}_n(x')dx' - \int_x^L e^{-\kappa k_n^2(x'-x)}\widehat{Q}_{corr,n}(x')dx'. \end{aligned}$$

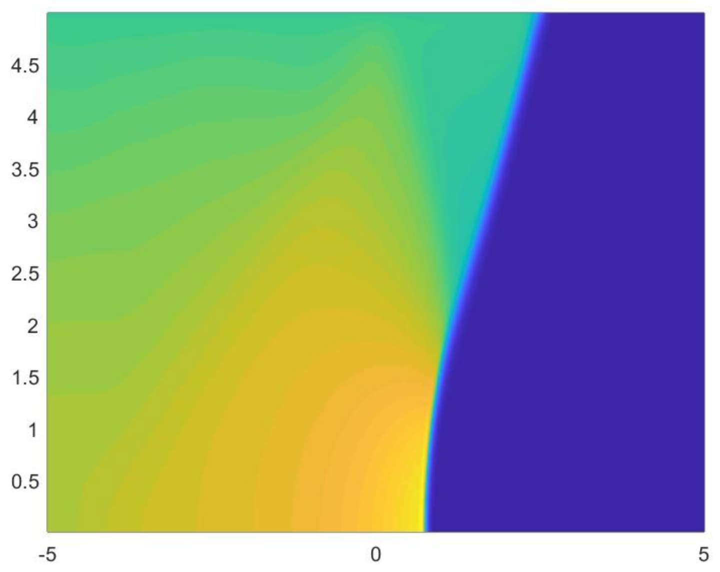

Figure 1: Numerical simulation of the UTSD equation for $a = 0.5$, $\Delta t = 0.0098$ for $N_t = 400$ steps.

The integration by parts method however, causes a boundary mismatch at $u(x, y = L_y, t)$ in the numerical simulation and we are still trying to identify the root cause (see Figure 2).

Regardless, the fourier space solution is then transform back to the real space by

$$\widetilde{\psi}(x, y) = \sum_{n=0}^{\infty} \widehat{\psi}_n(x)\Psi_n(y).$$

This process is applied repeatedly for each timestep to solve the UTSD equation.

2 Fast Cosine Transform for DCT-IV

For each timestep n, the algorithm need to perform a discrete cosine transform (DCT) on Φ_n with $N = N_x \times N_y$ number of grid points. Using the traditional approach of discrete cosine transform with matrix multiplication, it requires a storage size of N_y^2 and the operation count is of order $\mathrm{O}(N^2)$. By implementing fast cosine transform, the storage size is reduced to $2N_y$ whereas the operation count is reduced to order $\mathrm{O}(N \log_2 N)$ while having the same accuracy.

The DCT-IV is given by

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)\left(k + \frac{1}{2}\right)\right], \qquad k = 0, 1, 2, ..., N-1 \tag{8}$$

or in polar form

$$X_k = \mathrm{Re}\left\{\sum_{n=0}^{N-1} x_n e^{i\frac{\pi}{N}\left(n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)}\right\}.$$

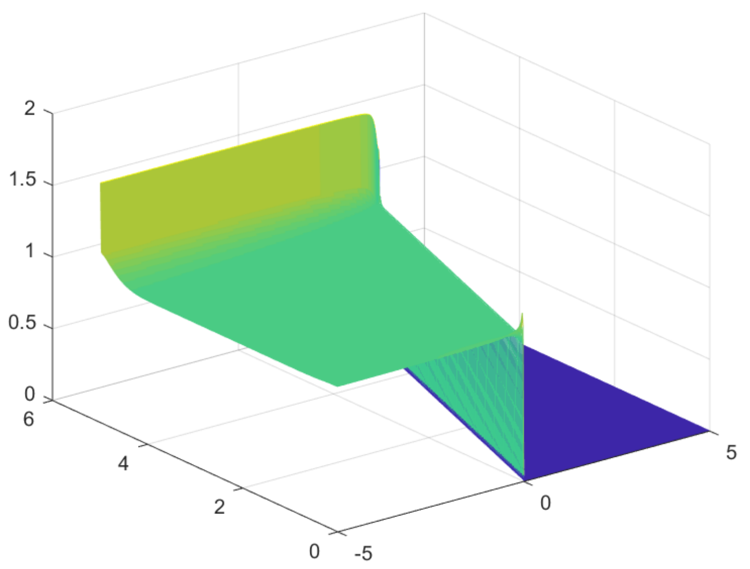

Figure 2: UTSD equaiton with integration by parts for $a = 0.5$, $\Delta t = 0.0098$ for $N_t = 1$ step.

We then seperate the sum into even and odd cases

$$X_k = \mathrm{Re}\left\{ \sum_{n=0}^{N/2-1} \left[x_{2n} e^{i\frac{\pi}{N}\left(2n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)} + x_{2n+1} e^{i\frac{\pi}{N}\left(2n+1+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)} \right] \right\},$$

reverse the index of the odd summad

$$X_k = \mathrm{Re}\left\{ \sum_{n=0}^{N/2-1} \left[x_{2n} e^{i\frac{\pi}{N}\left(2n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)} + x_{N-1-2n} e^{i\frac{\pi}{N}\left(N-1-2n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)} \right] \right\},$$

and factor out $e^{i\pi\left(k+\frac{1}{2}\right)}$, which is equal to $i(-1)^k$

$$X_k = \mathrm{Re}\left\{ \sum_{n=0}^{N/2-1} \left[x_{2n} e^{i\frac{\pi}{N}\left(2n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)} + i(-1)^k x_{N-1-2n} e^{i\frac{\pi}{N}\left(-2n-\frac{1}{2}\right)\left(k+\frac{1}{2}\right)} \right] \right\}.$$

Therefore,

$$X_{2k} = \mathrm{Re}\left\{ \sum_{n=0}^{N/2-1} (x_{2n} + i x_{N-1-2n}) e^{-i\frac{\pi}{N}\left(2n+\frac{1}{2}\right)\left(2k+\frac{1}{2}\right)} \right\},$$
$$X_{N-1-2k} = \mathrm{Re}\left\{ \sum_{n=0}^{N/2-1} (i x_{2n} - x_{N-1-2n}) e^{-i\frac{\pi}{N}\left(2n+\frac{1}{2}\right)\left(2k+\frac{1}{2}\right)} \right\},$$

for $k = 0, 1, 2, ..., N/2 - 1$.

Or in other words

$$X_k = \begin{cases} \mathrm{Re}\{Z_{k/2}\} & \text{for even k,} \\ -\mathrm{Im}\{Z_{(N-1-k)/2}\} & \text{for odd k,} \end{cases} \tag{9}$$

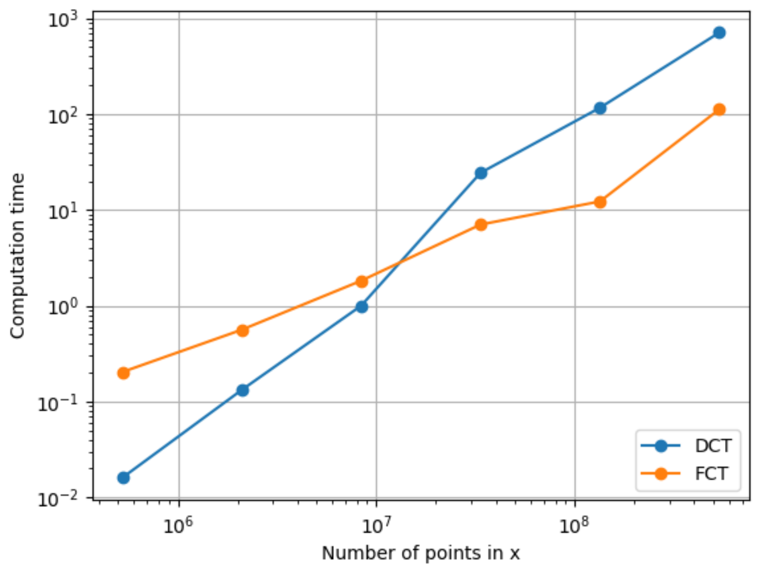

Figure 3: Computation time comparison of DCT and FCT.

for $k = 0, 1, 2, ..., N-1$, where

$$Z_k = e^{-i\frac{\pi}{2N}(2k+\frac{1}{2})} \sum_{n=0}^{N/2-1} (x_{2n} + ix_{N-1-2n})e^{-i\frac{\pi}{N}n}e^{-i\frac{\pi}{N/2}2nk},$$

which itself is a $N/2$ point fourier transform of $z_n := (x_{2n}+ix_{N-1-2n})e^{-i\frac{\pi}{N}n}$ to some multiple. This can be solve effectively using the existing fast fourier transform algorithm.

The inverse is then simply undo the multiplication and apply the inverse fast fourier transform:

$$x_n = \begin{cases} \mathrm{Re}\{\widehat{Z}_{n/2}\} & \text{for even n,} \\ \mathrm{Im}\{\widehat{Z}_{(N-1-n)/2}\} & \text{for odd n,} \end{cases} \tag{10}$$

for $n = 0, 1, 2, ..., N-1$, where

$$\widehat{Z}_n = e^{i\frac{pi}{N}n}\frac{1}{N}\sum_{k=0}^{N/2-1} Z_k e^{i\frac{\pi}{2N}\left(2k+\frac{1}{2}\right)}e^{i\frac{\pi}{N/2}2nk}.$$

This scheme has been validated using a MATLAB code and implemented to the UTSD solver. It should be noted that the Fast Fourier Transform (FFT) and Inverse FFT is done by MATLAB's build in function. A comparison is done with transforming one matrix of size $N = N_x \times N_y$. From Figure 3, we can see that the FCT outperform DCT for sufficiently large N and the efficiency will scale directly with the number of timesteps N_t while solving the UTSD equation.

3 Hyperdiffusion equation

Motivated to solve the Cahn-Hilliard equation in large quantities, we look at producing an efficient, parallelizable algorithm to be used in NVIDIA GPU [6]. In particular, we focus on solving a simpler case of the Cahn-Hilliard equation, namely the Hyperdiffusion equation [7]. Solving the hyperdiffusion equation require us to solve a system of equations $Ax = b$ for each grid point for every timestep, where A is a cyclic pentadiagonal matrix. We utilize the fact that A stays the same for every computation to improve the efficiency of the algorithm.

The 1D hyperdiffusion equaiton is given by

$$\frac{\partial C}{\partial t} = -\gamma D \frac{\partial^4 C}{\partial x^4}, \qquad t > 0, \qquad x \in (0, L), \tag{11}$$

with periodic boundary condition $C(x + L) = C(x)$ and initial condition $C(x, t = 0) = f(x)$.

By discretizing the hyperdiffusion equation using the Crank-Nicholson scheme, we get

$$\begin{aligned}\frac{C_i^{n+1} - C_i}{\Delta t} = -\frac{1}{2}\Delta x^4[C_{i+2}^{n+1} - 4C_{i+1}^{n+1} + 6C_i^{n+1} - 4C_{i-1}^{n+1} + C_{i-2}^{n+1}]\\ - \frac{1}{2}\Delta x^4[C_{i+2}^{n} - 4C_{i+1}^{n} + 6C_i^{n} - 4C_{i-1}^{n} + C_{i-2}^{n}],\end{aligned}$$

re-aranging yields

$$\begin{aligned}\frac{1}{2}rC_{i+2}^{n+1} - 2rC_{i+1}^{n+1} + (1+3r)C_i^{n+1} - 2rC_{i-1}^{n+1} + \frac{1}{2}rC_{i-2}^{n+1}\\ = -\frac{1}{2}rC_{i+2}^{n} + 2rC_{i+1}^{n} + (1-3r)C_i^{n} + 2rC_{i-1}^{n} - \frac{1}{2}rC_{i-2}^{n}.\end{aligned} \tag{12}$$

This can be expressed as a system of equation

$$\begin{pmatrix} c & d & e & 0 & & & 0 & b & a \\ b & c & d & e & 0 & & & 0 & b \\ a & b & c & d & e & 0 & & & 0 \\ 0 & a & b & c & d & e & 0 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 0 & a & b & c & d & e & 0 \\ 0 & & & 0 & a & b & c & d & e \\ e & 0 & & & 0 & a & b & c & d \\ d & e & 0 & & & 0 & a & b & c \end{pmatrix} \begin{pmatrix} C_1^{n+1} \\ C_2^{n+1} \\ C_3^{n+1} \\ C_4^{n+1} \\ \vdots \\ C_{N-3}^{n+1} \\ C_{N-2}^{n+1} \\ C_{N-1}^{n+1} \\ C_N^{n+1} \end{pmatrix} = \begin{pmatrix} d_1^n \\ d_2^n \\ d_3^n \\ d_4^n \\ \vdots \\ d_{N-3}^n \\ d_{N-2}^n \\ d_{N-1}^n \\ d_N^n \end{pmatrix},$$

where d_i^n is given by

$$d_i^n = -\frac{1}{2}rC_{i+2}^n + 2rC_{i+1}^n + (1-3r)C_i^n + 2rC_{i-1}^n - \frac{1}{2}rC_{i-2}^n.$$

We solve this by first dividing the matrix into sections:

$$\left(\begin{array}{ccccccc|cc} c & d & e & 0 & & & 0 & b & a \\ b & c & d & e & 0 & & & 0 & b \\ a & b & c & d & e & 0 & & & 0 \\ 0 & a & b & c & d & e & 0 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 0 & a & b & c & d & e & 0 \\ 0 & & & 0 & a & b & c & d & e \\ \hline e & 0 & & & 0 & a & b & c & d \\ d & e & 0 & & & 0 & a & b & c \end{array}\right) \left(\begin{array}{c} C_1^{n+1} \\ C_2^{n+1} \\ C_3^{n+1} \\ C_4^{n+1} \\ \vdots \\ C_{N-3}^{n+1} \\ C_{N-2}^{n+1} \\ \hline C_{N-1}^{n+1} \\ C_N^{n+1} \end{array}\right) = \left(\begin{array}{c} d_1^n \\ d_2^n \\ d_3^n \\ d_4^n \\ \vdots \\ d_{N-3}^n \\ d_{N-2}^n \\ \hline d_{N-1}^n \\ d_N^n \end{array}\right),$$

and lable each section as such

$$\begin{pmatrix} E & f \\ g^T & W \end{pmatrix} \begin{pmatrix} \widehat{X} \\ \widetilde{X} \end{pmatrix} = \begin{pmatrix} \widehat{d} \\ \widetilde{d} \end{pmatrix}.$$

Thus the system becomes two coupled simultaneous equations

$$E\widehat{X} + f\widetilde{X} = \widehat{d}$$
$$g^T\widehat{X} + W\widetilde{X} = \widetilde{d}$$

solving this simultaneous equation yields

$$\widetilde{X} = \big(W - g^T E^{-1} f\big)^{-1}\big(\widetilde{d} - g^T E^{-1}\widehat{d}\big) \tag{13}$$
$$\widehat{X} = E^{-1}\big(\widehat{d} - f\widetilde{X}\big). \tag{14}$$

The LU factorization of E, $\big(W - g^T E^{-1} f\big)^{-1}$, and $g^T E^{-1}$ can be computed and stored beforehand. A C code was produced for the implementation described above for validation and it was used as a benchmark for the more sophisticated cuPentBatch which interleaves the input into a more accessible format. cuPentBatch is able to outperform the current state-of-the-art algorithm - gpsvInterleavedBatch from NVIDIA's CUDA library. A C code to solve the two-dimensional hyperdiffusion equation was also produced to set up the pathway to tackle the Cahn-Hilliard equation.

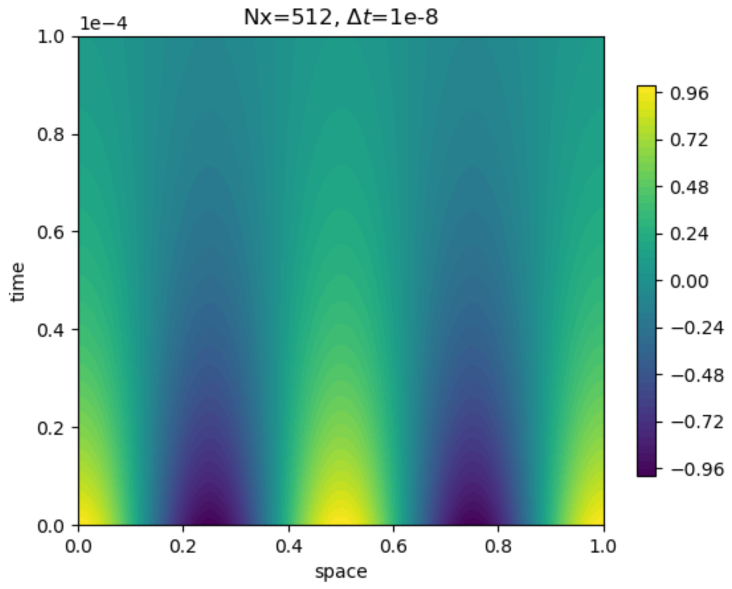

Figure 4: A plot of the one-dimensional diffusion equation with initial condition $f(x) = cos(2n\pi x)$ where $n = 2$.

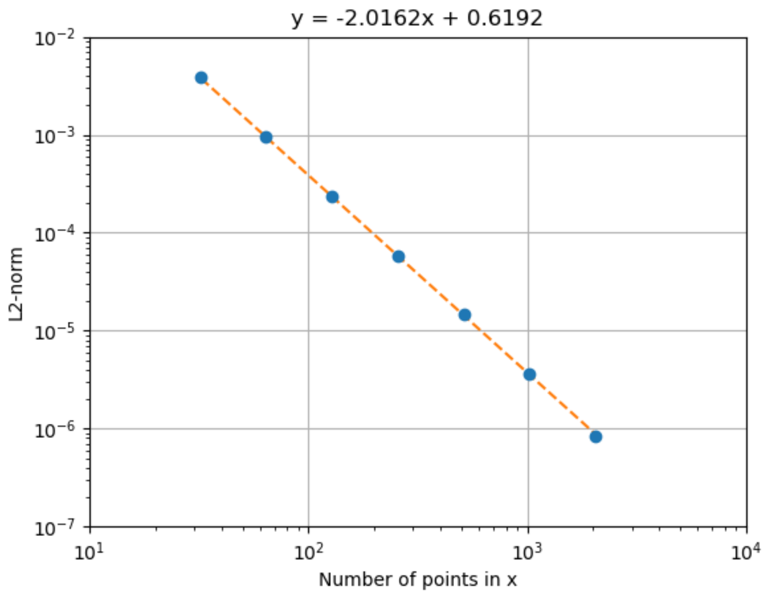

Figure 5: L2 norm plot of the algorithm as a function of resolution. As we can see from the plot, the slop of -2 as the Crank-Nicholson scheme is $O(N^2)$ accurate.

4 Conclusion

In the first part of the project, we improve the algorithm to solve the UTSD equation to support parallel computing by finding the analytic solution for the backward heat equation. This method makes use of Fourier transform which we then implement fast cosine transform to optimize the computation speed. Further effort will be put on integrating a better method to resolve the discontinuity while solving the backward heat equation.

In the second part of the project, we utilize a property of the hyperdiffusion equation to produce an efficient algorithm for the hyperdiffusion equation. We are able to achieve a significant speedup in computational time compared to other libraries. Moreover, this idea is applicable to solve higher order PDE that retain the LHS matrix during numerical computation. Our work is featured in the journal - Computer Physics Communications titled 'cuPentBatch - A Batch Pentadiagonal Solver for NVIDIA GPUs'. Future work would be applying this algorithm to solve the Cahn-Hilliard equation in one-dimension and ultimately, in higher dimensions.

Overall, my summer student internship project has been fruitful and it certainly gives me a taste of what the front line of mathematical research is like with the successes and failures. On top of that, I have gained a lot from this internship including picking up two programming language (MATLAB and C), as well as developing good programming and scientific routine for debugging and validating results. I am positive that my time and experience during this internship will prove valuable and aid me along my path to pursue a career in academia.

Acknowledgement

Khang Ee Pang would like to thank the UCD School of Mathematics and Statistics for the opportunity and funding for the Summer Research Placement 2018 program. In addition, he would like to thank Dr. Lennon Ó Náraigh and Andrew Gloster for the guidance and supervision.

References

[1] J. von Neumann. *Collected works*, Vol 6. Pergamon Press, New York (1963).

[2] J. K. Hunter and M. Brio. Weak shock reflection. *J. Fluid Mech.*, 410:235-261, 2000.

[3] K. G. Guderly. *The Theory of Transonic Flow*, 144-149. Pergamon Press, New York (1962).

[4] B. W. Skews and J. T. Ashworth. The physical nature of weak shock wave reflection. *J. Fluid Mech.*, 542:105-144, 2005.

[5] L. Ó Náraigh. New idea for the UTSD equation. 2018.

[6] L. Ó Náraigh and Andrew Gloster. Potential applications for a pentadiagonal solver. 2018.

[7] L. Ó Náraigh and Andrew Gloster. cuPentBatch - A batch pentadiagonal solver for NVIDIA GPUs. 2018.