San Diego State University

Department of Mathematics and Statistics
Math 639
Nonlinear Waves

Final Project:
Solving the 2D Advection-Diffusion Equation:
Pseudo-Spectral Methods and Finite Difference Schemes

Bin Hoang,
Horacio Lopez,
Matteo Polimeno

Professor:
Dr. Christopher Curtis

May 10th, 2018

Contents

1 Abstract

This project deals with the implementation of numerical methods to solve the 2D advection-diffusion equation. Specifically, this paper focuses on comparing Pseudospectral methods to Finite Difference schemes. In both approaches, the solution to the Poisson equation is discussed as well as the implementation of an appropriate time-stepper to compute the updated vorticity needed to solve the advection-diffusion equation.
A brief mathematical background is given before proceeding to the discussion of the schemes.

2 Background

Following [1, 3], to derive the advection-diffusion equation, the vorticity equation is needed. For a fluid velocity $\mathbf{u}$, we write the momentum Navier-Stokes equation, assuming incompressibility ($\nabla \cdot \mathbf{u} = 0$) and constant density throughout the fluid.
The vorticity is given by $\boldsymbol{\omega} = \nabla \times \mathbf{u}$. From [1], the momentum Navier-Stokes equation can be written as

$$\rho \frac{\mathcal{D}\mathbf{u}}{\mathcal{D}t} = -\nabla p + \rho \mathbf{g} + \mu \Delta \mathbf{u} \tag{1}$$

where $\frac{\mathcal{D}}{\mathcal{D}t}$ is the material derivative, ρ is the density of the fluid, p is the pressure and μ is the viscosity, see [1]. Rescaling leads to

$$\frac{\mathcal{D}\mathbf{u}}{\mathcal{D}t} = -\frac{1}{\rho}\nabla p + \mathbf{g} + \nu \Delta \mathbf{u} \tag{2}$$

where $\nu = \frac{\mu}{\rho}$ is the kinematic viscosity and $\mathbf{g}$ is gravity. It holds $\mathbf{g} = -\nabla \phi$, with ϕ being a scalar function.
For the vorticity $\boldsymbol{\omega}$ it holds $\nabla \cdot \boldsymbol{\omega} = \nabla \cdot (\nabla \times \mathbf{u}) = 0$. An equation for the vorticity can be obtained by taking the curl of both sides of (2)

$$\nabla \times \left(\frac{\mathcal{D}\mathbf{u}}{\mathcal{D}t} = -\frac{1}{\rho}\nabla p + \mathbf{g} + \nu \Delta \mathbf{u} \right) \tag{3}$$

As $\mathbf{g} = -\nabla \phi$ and p is a scalar, the first two terms on the right-hand side of (3) are both 0, because they are gradient of scalar functions. Moreover, the material derivative in (3) can be re-written as

$$\frac{\mathcal{D}\mathbf{u}}{\mathcal{D}t} = \partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} \tag{4}$$

Then, following [1], we derive

$$\nabla \times \left(\frac{\mathcal{D}\mathbf{u}}{\mathcal{D}t} = \partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) = \partial_t \boldsymbol{\omega} + \nabla \times (\boldsymbol{\omega} \times \mathbf{u}) \tag{5}$$

Thus, (3) reduces to

$$\partial_t \boldsymbol{\omega} + \nabla \times (\boldsymbol{\omega} \times \mathbf{u}) = \nu \Delta \boldsymbol{\omega} \tag{6}$$

where the identity $\nabla \times \Delta \mathbf{u} = \Delta(\nabla \times \mathbf{u})$ is used. Furthermore, by re-writing $\nabla \times (\boldsymbol{\omega} \times \mathbf{u}) = (\mathbf{u} \cdot \nabla)\boldsymbol{\omega} - (\boldsymbol{\omega} \cdot \nabla)\mathbf{u}$ and making use of the fact that $\nabla \cdot \mathbf{u} = 0$ and $\nabla \cdot \boldsymbol{\omega} = 0$, then (3) becomes

$$\frac{\mathcal{D}\boldsymbol{\omega}}{\mathcal{D}t} = (\boldsymbol{\omega} \cdot \nabla)\mathbf{u} + \nu \Delta \boldsymbol{\omega} \tag{7}$$

which is the field equation governing the vorticity in a fluid with constant density.
In 2D space (x,y), it holds

$$\mathbf{u} = \begin{bmatrix} u \\ v \\ 0 \end{bmatrix}, \tag{8}$$

which implies

$$(\boldsymbol{\omega} \cdot \nabla)\mathbf{u} = (\omega_x \partial_x + \omega_y \partial_y + \omega_z \partial_z)\, \mathbf{u} = 0, \tag{9}$$

where ω_j, with $j = x, y, z$, indicates the j-component of the vector $\boldsymbol{\omega}$.
Thus, we end up with

$$\frac{\mathcal{D}\boldsymbol{\omega}}{\mathcal{D}t} = \nu \Delta \boldsymbol{\omega} \tag{10}$$

which is the vorticity equation in 2D.
In fluid dynamics, it is typical to work with streamfunctions. Therefore, let $\psi(x, y, t)$ be a streamfunction defined as

$$u = -\partial_y \psi, \;\; v = \partial_x \psi \tag{11}$$

In terms of the vorticity, the streamfunction relates as

$$\omega = \partial_x v - \partial_y u = \Delta \psi \tag{12}$$

which represents the Poisson equation in two dimensions.
Then, making use of

$$\frac{\mathcal{D}\boldsymbol{\omega}}{\mathcal{D}t} = \partial_t \boldsymbol{\omega} + (\boldsymbol{\omega} \cdot \nabla)\boldsymbol{\omega} \tag{13}$$

the advection-diffusion equation for the vorticity can be written as

$$\partial_t \boldsymbol{\omega} + [\psi, \omega] = \nu \Delta \boldsymbol{\omega} \tag{14}$$

where $[\psi, \omega] = \partial_x \psi \partial_y \omega - \partial_y \psi \partial_x \omega$
Finally, we have the following system of equations

$$\partial_t \omega + [\psi, \omega] = \nu \Delta \omega \tag{15}$$

$$\Delta \psi = \omega, \tag{16}$$

which is solved numerically through the schemes implemented in the following sections.

2.1 Overview of the numerical set-up

Both numerical methods follow the outline given by [3]:

- Given the initial vorticity, ω_0, get the streamfunction, ψ_0, by solving the Poisson equation with periodic boundary conditions

$$\Delta \psi_0 = \omega_0$$

- Implement a time stepper to solve the advection-diffusion equation and obtain the updated vorticity ω_1
- Use the new vorticity to obtain the updated streamfunction, and repeat.

3 Body

3.1 Solving the Advection-Diffusion equation using Pseudo-Spectral Methods

The first approach to solving the advection-diffusion equation makes use of pseudo-spectral methods. MATLAB's and PYTHON's NUMPY's built in function FFT (Fast Fourier Transform), is used to calculate spatial derivatives in Fourier space. FFT calculates the discrete Fourier transform of a vector or matrix representing a function.
The wavenumber vector is defined as:

$$\vec{k} = \begin{bmatrix} 0 \\ \vdots \\ (N/2) \\ 1-(N/2) \\ \vdots \\ -1 \end{bmatrix} \tag{17}$$

where our spatial grid is $(N \times N)$. To calculate derivatives in either the x or y direction, we introduce Kronecker tensors, see [6], in order to make use of Kronecker products defined as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix},$$

Figure 1: Figure from Wikipedia

where A and B are square matrices and where $\otimes$ is the Kronecker product operation.
Using $\vec{k}$ and MATLAB's Kronecker product built in function, $kron$, we calculate the spatial derivatives in Fourier space of a 2π periodic grid by using the Kronecker tensors:

$$\widehat{\partial_y} = \frac{i2\pi}{L} \begin{bmatrix} 0 \\ \vdots \\ (N/2) \\ 1-(N/2) \\ \vdots \\ -1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 1 \end{bmatrix}, \quad \widehat{\partial_x} = \frac{i2\pi}{L} \begin{bmatrix} 1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ \vdots \\ (N/2) \\ 1-(N/2) \\ \vdots \\ -1 \end{bmatrix} \tag{18}$$

With our Fourier-space differential operators defined this way, performing element-wise multiplication between the flattened Fourier transformed, $FFT2$, the data matrix and the respective Kronecker tensor is sufficient to calculate derivatives in Fourier space.
In order to reduce operations to $\mathcal{O}(\text{NLogN})$, we ensure that our $(N \times N)$ spatial grid is set up such that $N = 2^n$ for some positive integer n. This is due to properties of the FFT algorithm used [3].

3.1.1 Solving The Poisson Equation Using Fast Fourier Transform (FFT)

In Fourier space, the use of Kronecker tensors (see [6]) as differential operators makes solving for the streamfunction an exercise in Algrebra. For the Poisson equation and its solution [3], it holds:

$$\widehat{\Delta\psi} = \widehat{\omega}. \tag{19}$$

$$\Rightarrow \left(\widehat{\partial}_x^2 + \widehat{\partial}_y^2\right)\widehat{\psi} = \widehat{\omega} \tag{20}$$

$$\Rightarrow \widehat{\psi} = \frac{\widehat{\omega}}{\left(\widehat{\partial_x^2} + \widehat{\partial_y^2}\right)} \tag{21}$$

In order to avoid division by 0 in (16), it is necessary to shift the solutions by introducing a small constant. To compensate for this, we calculate the zeroth order term (the mean of vorticity), defined as $<\omega>= \iint_D \omega_0 dxdy \approx 0.00878$, and used it to rescale the solution in order to maintain the right order of magnitude. To bring the solution of the streamfunction back into real space, Matlab's built-in function $IFFT2$ (Inverse Fourier Transform in 2D) is applied to the reshaped $(N \times N)$ solution matrix.

3.1.2 Solving The Advection-Diffusion Equation Using FFT and 4^{th} Order Runge-Kutta

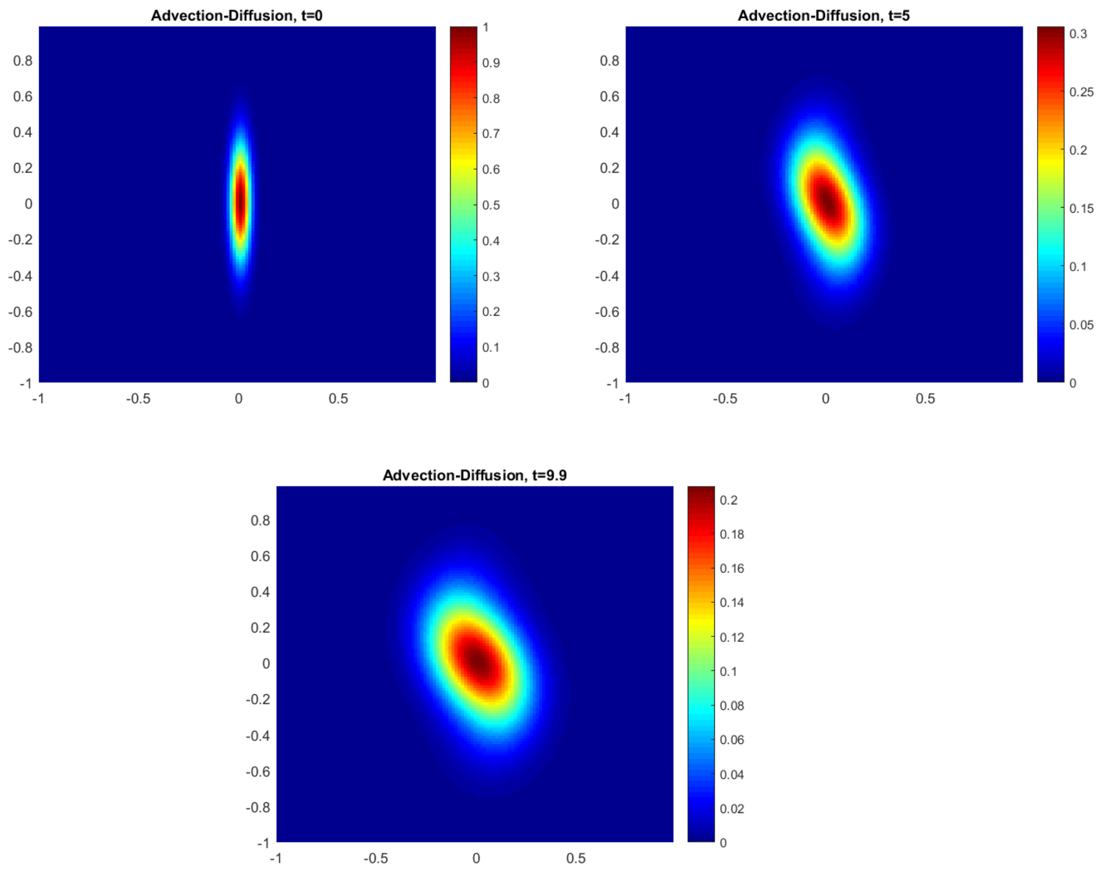
Now that the streamfunction is known from the Poisson equation's solution, the next step is to solve the advection-diffusion equation, (10), with a time-stepper to update the vorticity. First, the advection term, $[\psi, \omega] = \partial_x\psi\partial_y\omega - \partial_y\psi\partial_x\omega$, is calculated by finding the derivatives

$$\frac{\partial\psi}{\partial x}; \ \frac{\partial\omega}{\partial y}; \ \frac{\partial\psi}{\partial y}; \ \frac{\partial\omega}{\partial x}$$

in Fourier space by using Kronecker tensor multiplication. The multiplication and substraction of the derivatives to calculate the advection term is done in real space after reshaping and performing $IFFT2$ and keeping the real terms of the transform. The resulting solution is then transformed back into Fourier space and flattened in order to continue the time-stepping.
The diffusion operator is also calculated by Kronecker tensor multiplication. After this, MATLAB's built in Runge-Kutta 4 numerical integration method, $ODE45$, is used to time-step and solve for the next vorticity term. The process is then repeated by solving the Poisson using the updated vorticity and then time-stepping the advection-diffusion equation using RK-4.

The following plots show the results of solving the 2-D advection-diffusion equation with an initial vorticity $\omega_0 = \exp(-20x^2 - 400y^2)$, $\nu = 0.001$, $\delta t = 0.1, N = 128, t = 10$.

Figure 2: Solution of the advection-diffusion equation with pseudo-spectral methods



3.2 Solving the Advection-Diffusion equation using Finite Difference Methods

In this section, we will take an alternative route to FFT and begin to explore the advection-diffusion equation via finite difference method. Our primary objective is to see how well FFT perform compared to traditional techniques such as finite difference, especially in term of speed and accuracy. The section consists of two parts. The first part focuses on solving the Poisson equations with three five point stencil methods, and the second concentrates on tackling the advection-diffusion equation with time splitting schemes. To be consistent, square grids where $dx = dy$ are used in all of the schemes.

3.2.1 Comparing Finite Differencing and FFT

The first step is to solve (15) using three second-order accurate, five-point stencil schemes, namely the Jacobi, Gauss Seidel and Successive Over-Relaxation (SOR) methods. These methods are closely related to each other, with minor differences. However, their performance can vary greatly.

3.2.2 The Jacobi Method

The Jacobi scheme is given by:

$$v_{i,j}^{k+1} = \frac{1}{4}(v_{i+1,j}^{k} + v_{i-1,j}^{k} + v_{i,j+1}^{k} + v_{i,j-1}^{k} - h^2\omega_{i,j}).$$

Here, k denotes the number of iteration, while i and j denotes the grid points in the x and y direction respectively.
For an arbitrary ω and a boundary condition, Jacobi formula sweeps the grid and approximates the solution for the Poisson equation via successive iterations. From Jacobi formula, if we know all the grid points for iteration k, we can begin to compute the $v_{i,j}^{k+1}$ at every interior grid points, and continue to propagate the scheme forward until the stopping criteria is met. A common stopping criteria is using the difference between two successive iterations $k+1$ and k : when $|v_{i,j}^{k+1} - v_{i,j}^{k}| < tolerance$, the scheme is stopped.

3.2.3 The Gauss-Seidel Method

Highly similar to the Jacobi scheme, the Gauss Seidel method is given by:

$$v_{i,j}^{k+1} = \frac{1}{4}(v_{i+1,j}^{k} + v_{i-1,j}^{k+1} + v_{i,j+1}^{k} + v_{i,j-1}^{k+1} - h^2\omega_{i,j}).$$

In this formula, as we go through the grid for each i and j to compute $v_{i,j}^{k+1}$, old values $v_{i-1,j}^{k} + v_{i,j-1}^{k}$ are immediately replaced with the newly updated values $v_{i-1,j}^{k+1} + v_{i,j-1}^{k+1}$. By doing so, the scheme becomes twice as fast while avoid having to save two copies of the solution, one for the old k^{th} iteration and one the new $(k+1)^{th}$ iteration. Thus comparing to the Jacobi scheme, Gauss Seidel updates twice as fast and uses half the storage.

3.2.4 The SOR Method

Another way to optimize the Gauss Seidel scheme is through introducing a parameter bb. The SOR can be written as:

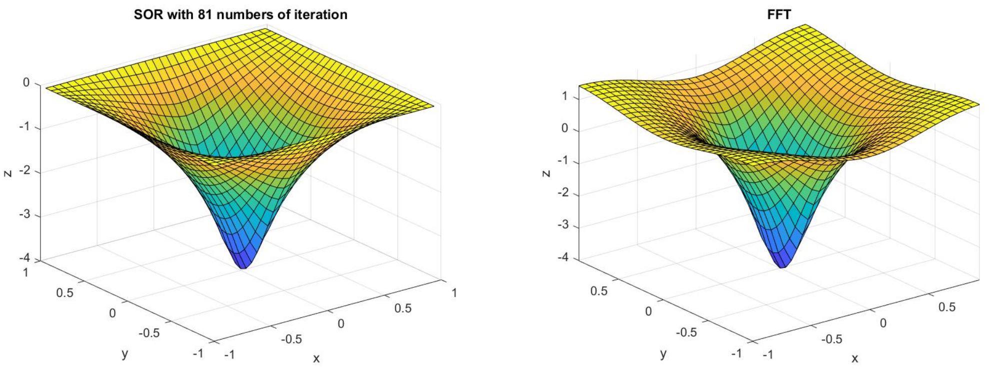
$$v_{i,j}^{k+1} = \frac{bb}{4}(v_{i+1,j}^{k} + v_{i-1,j}^{k+1} + v_{i,j+1}^{k} + v_{i,j-1}^{k+1} - h^2\omega_{i,j}) + (1-bb)v_{i,j}^{k}$$

Note that for $bb = 1$, we recover our familiar Gauss Seidel scheme. It is not quite obvious how this modification makes the computation faster, and the story can be complicated, so we will omit its derivation here, (see [5] for more details). It turns out that the optimal value for bb is around $\frac{2}{1+\sin(\pi/(N-1))}$, and the improvement from this small adjustment can be quite dramatic. With the optimal choice of bb, for the N by N grid, the number of iterations for SOR is of order N, while it is around N^2 for Gauss Seidel and Jacobi [5].
In the appendix, we test (3.2.2)-(3.2.4) for a Poisson equation where the analytical solution is known. If their performance is adequate, we will solve the Poisson equation with the same ω_0 used in the previous section, and draw comparison between finite difference method and FFT. After several test runs, all three of the schemes were able to give the right solution within second order accuracy.

3.2.5 Testing Finite Difference Schemes

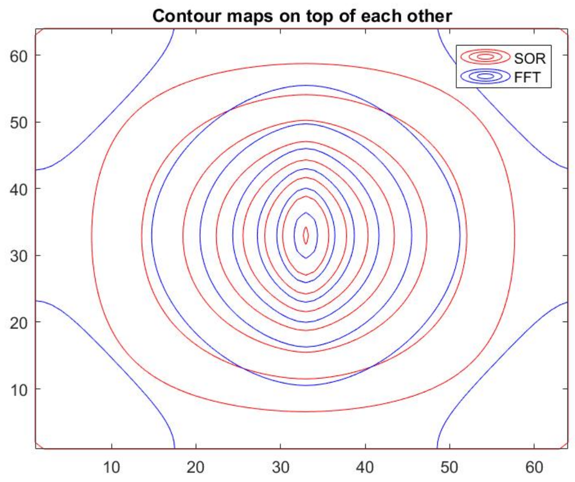
Since the SOR is the fastest scheme out of the three, it is our representative for the finite difference methods. Therefore, we will solve (15) with $\omega_0 = \exp(-20x^2 - 400y^2)$ using SOR scheme and compared its effectiveness to the FFT technique. Once again,the tolerance for the finite schemes is 10^{-4}. Unfortunately, we do not know the analytical solution of the Poisson equation in this case, so we can only rely on how the solutions look compared to each other to judge their accuracy. Since the FFT method does not have a unique solution, we will instead compare their contour plots with the streamfunctions themselves. The essential physics here lies in the derivatives of the ψ anyway, so no important information is lost.
The results of the two schemes are showed side by side in Figure 3.

Figure 3: Solution of Poisson Equation with w_0 between SOR and FFT



While the resulting plots look similar, their differences are clearly visible in the contour plot (Figure 4). The inner contours show good agreement, while the contours on the edge do not. This disagreement persists even after we have taken the zeroth order term mentioned in the previous section into account.

Figure 4: Contour plots for SOR and FFT



The reason for this discrepancy is because we have disagreements in terms of boundary conditions, as the finite difference schemes implemented, in contrast to FFT, make no used of periodic boundary conditions. Therefore to avoid unfair comparison, the vorticity ω is picked so that it will exponential decay faster away from the center. Therefore, the difference at the boundary will be small, and tolerable, but still something to be taken into account in the analysis of the results. We refer to [3], p.185, for a comparison.

3.2.6 Solving the Advection-Diffusion Equation Via Time Splitting

The advection-diffusion equation consists of both wave-like (advection) behavior and diffusion-like behavior. However, most standard finite difference schemes are only stable for either advection or diffusion PDE's but not for both [3]. Thus, to avoid this issue, given a smaller enough time step, we can split the advection-diffusion equation into two parts, and treat the wave-like and diffusion-like components separately. Therefore, we divide the advection-diffusion equation as follows:

- the advection portion,

  $$\frac{\partial \omega}{\partial t} + [\psi, \omega] = 0, \tag{22}$$

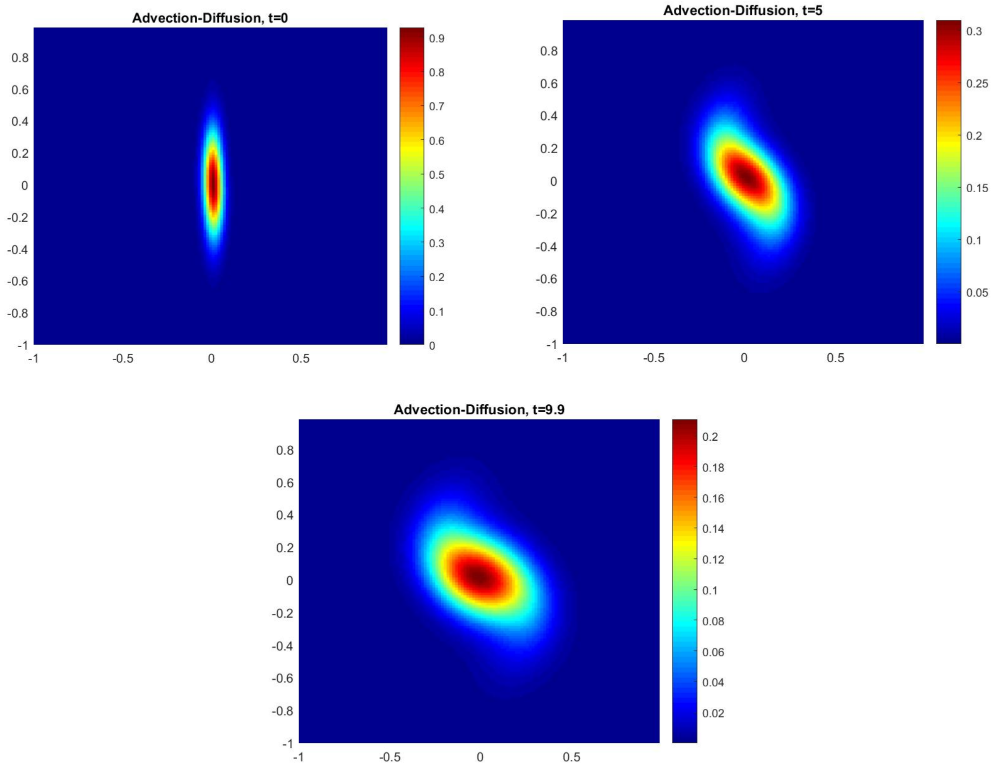
  solved using forward time central space,

- the diffusion portion,

  $$\frac{\partial \omega}{\partial t} = \nu \Delta \omega, \tag{23}$$

solved with Matlab's built-in function *ODE23*. The accuracy of the finite difference scheme is of second order both in space and time.
The solution of the advection-diffusion equation with a time-splitting scheme is shown in Figure 5.

Figure 5: Solution of the advection-diffusion equation with time-splitting method



4 Final Remarks

For relative comparison of the two techniques, the initial vorticity used is $\omega_0 = \exp(-20x^2 - 400y^2)$,while the spatial grid is $(N \times N)$, with N = 64. The time step, δt, is equal to 0.1 and run the simulation until $t = 10$ with $\nu = 0.001$. Due to some issues encountered when solving the Poisson equation and ensuring uniqueness and uniformity in our streamfunction calculations, both the pseudo-spectral method and the finite difference method will use the streamfunction obtained from the FFT solution. The numerical simulation with Pseudo-Spectral methods takes 0.5062 seconds to complete while the simulation with finite differencing methods takes 4.2343 minutes. Results lean heavily in favor of the pseudo-spectral method implemented. The difference in running time is very apparent. Regardless of this running time comparison, a pros and cons for each method are listed to fully appreciate the advantages and limitations of each method.
Pseudo-Spectral Method:

- Pros:
  - Much higher speed, $\mathcal{O}(NLogN)$
  - Allows for better resolution; higher grid point number.
  - Spectral Accuracy.
- Cons:
  - Have to work in periodic square grid.

Finite Differencing Method:

- Pros:
  - Allows for work in rectangular grids
  - Can use periodic, Neumann, and Dirichelet boundary conditions.
- Cons:
  - Very slow compared to Pseudo Spectral method.

In conclusion, selecting either method depends on what results are desired and the setup of the problem. The pseudo-spectral method approach was much faster than the finite differencing method, but it solves a more contrived problem. Even if pseudo-spectral methods were to be implemented with Chebyshev Polynomials [6], in order to get around working with periodic boundary conditions, the constrain of the use of a square grid still stands [6]. It is also worth noting that the finite differencing method implemented here is elementary and more accurate and efficient methods exist [5].

5 Bibliography

References

[1] Cohen, I.M. and Kundu, P.K. *Fluid Mechanics.* Elsevier Science, 2007.

[2] Garcia, A. *Numerical Methods for Physics.* CreateSpace Independent Publishing Platform, 2015.

[3] Kutz, J.N. *Data-Driven Modeling & Scientific Computation Methods for Complex Systems and Big Data.* Oxford University Press, 2013.

[4] Pironneau, O. *Finite Element Methods for Fluids.* Chichester etc., John Wiley & Sons, Paris etc., Masson 1989.

[5] Strikwerda, J.C. *Finite Difference Schemes and Partial Differential Equations.* Society for Industrial and Applied Mathematics, 2004.

[6] Lloyd N. Trefethen. *Spectral Methods in Matlab* SIAM, 2000.

6 Appendix

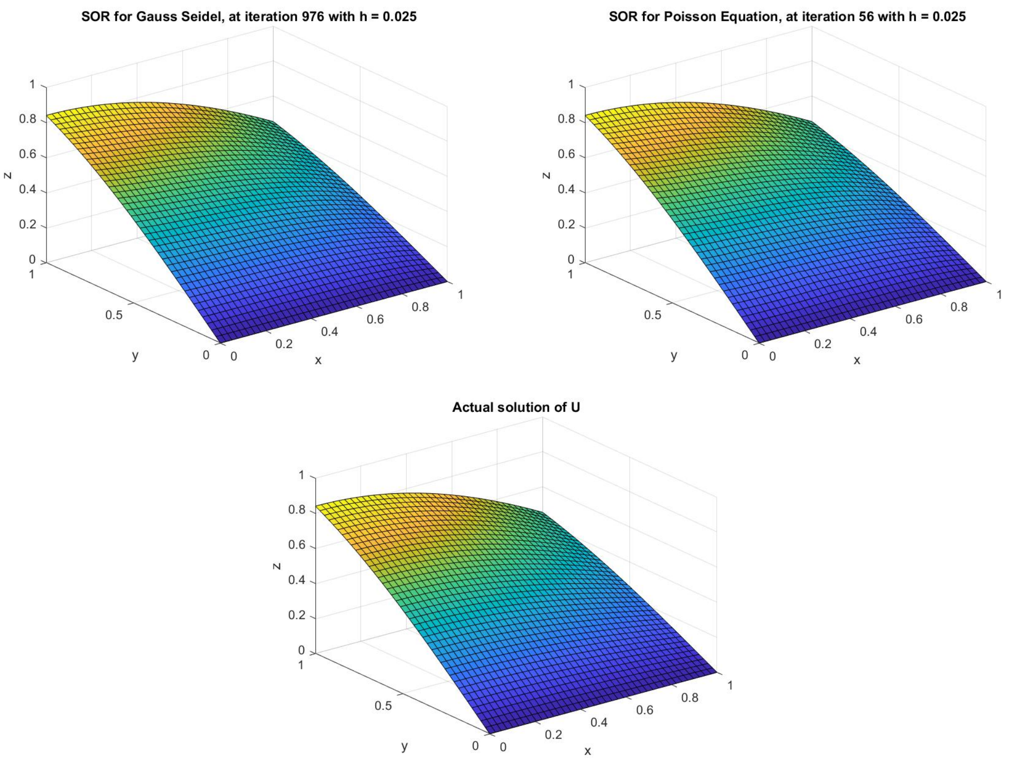
6.1 A: Testing Finite Difference Schemes

Let us solve the following Poisson equation using both the Gauss Seidel and SOR methods :

$$u_{xx} + u_{yy} = -2\cos(x)\sin(y) \tag{24}$$

The analytical solution is given by $u = \cos(x)\sin(y)$. The stopping criteria is set such that the L^2 norm of the difference between the actual solution and the finite difference scheme, $L^2 = ||(T_{actual} - T_{finite})||dx^2$, is of the order of 10^{-4}. The results of the comparison between the Gauss-Seidell scheme and the SOR scheme are shown in below (Figure 6).

Figure 6: Solution of Poisson Equation, Gauss Seidel and SOR, and the actual solution



Both numerical solutions match the analytical solution satisfactorily. However, for the same accuracy, the SOR converges almost twenty time faster than the Gauss-Seidel method. The difference between N vs N^2 convergence rate appears to be significant, thus making SOR the practical choice when working with larger grids.