Solving the Schrödinger Equation Using Mimetic Differences

Mani Amani and Miguel A. Dumett

April 17, 2024

Publication Number: CSRCR2024-05

Computational Science & Engineering Faculty and Students Research Articles

Database Powered by the Computational Science Research Center Computing Group & Visualization Lab

COMPUTATIONAL SCIENCE & ENGINEERING

SAN DIEGO STATE UNIVERSITY

Computational Science Research Center
College of Sciences
5500 Campanile Drive
San Diego, CA 92182-1245
(619) 594-3430

© 2024

Solving the Schrödinger Equation using Mimetic Differences

Mani Amani* Miguel A. Dumett †‡

April 17, 2024

Abstract

In this document we solve the Schrödinger equation with unbounded potential utilizing mimetic differences.

1 Introduction

The Schrödinger equation, formulated by Erwin Schrödinger in 1925, is a cornerstone of quantum mechanics, providing a quantitative description of the dynamics of quantum systems. It is a linear partial differential equation that governs the wave function of a quantum system, which is a complex-valued function representing the probability amplitude of a system's quantum state. The wave function itself encapsulates all the information about a system's state, enabling the calculation of physical properties like energy, momentum, and position.

There have been a plethora of previous studies that have successfully solved the Schrödinger equation using different explicit and implicit methods, such as Crank-Nicholson or finite difference methods. In this manuscript, we propose a mimetic difference method to solve the system.

The Schrödinger equation can be reformulated and non-dimensionalized into the following equation and can be solved with Dirichlet boundary conditions.

$$-i\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \omega(x,y)u \tag{1}$$

We propose to solve the partial differential equation using a Mimetic differences scheme. This would consist of discretizing the spatial derivative using mimetic methods and discretizing the time using traditional time integrators. We follow the work of [3] and analyze solving the Schrodinger equation using this method given by conditions and solutions presented by the authors.

*Computational Science Master's Program at San Diego State University (mamani5250@sdsu.edu).

†Editor: Jose E. Castillo

‡Computational Science Research Center at San Diego State University (mdumett@sdsu.edu).

Mimetic Discretization

The mimetic discretization is given by methods developed by[1] and using the MOLE package [2] to solve. In this specific equation, we only have one mimetic operator which is the Laplacian. This term can be rewritten into the Laplacian U and the function multiplication given by ω can be transformed into a diagonal matrix to be accounted for in the numerical analysis. The mimetic discretization of equation (1) can be formulated into:

$$\frac{du}{dt} = i\left(LU + \text{diag}(\omega(x, y) \cdot U)\right) \tag{2}$$

Where L is the Laplacian operator. After the space discretization is complete we proceed with the time discretization. This spatial discretization turns the Partial Differential Equation (PDE) into an Ordinary Differential Equation (ODE), which can be solved using integration with respect to time.

Time discretization

We opted for a Runge-Kutta 4 method for increased stability in calculating the time derivative. Forward and Backward Euler proved unstable to solve the equation. The Pseudocode of the RK4 method is given by the following:

Algorithm 1 Runge-Kutta 4th Order Method (RK4)

function RK4(y_0, t_0, t_{end}, h)
    $y \leftarrow y_0$
    $t \leftarrow t_0$
    while $t < t_{\text{end}}$ **do**
        $k_1 \leftarrow h \cdot f(t, y)$
        $k_2 \leftarrow h \cdot f(t + 0.5 \cdot h, y + 0.5 \cdot k_1)$
        $k_3 \leftarrow h \cdot f(t + 0.5 \cdot h, y + 0.5 \cdot k_2)$
        $k_4 \leftarrow h \cdot f(t + h, y + k_3)$
        $y \leftarrow y + \frac{1}{6} \cdot (k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4)$
        $t \leftarrow t + h$
    end while
    return y
end function

Experimentation and Results

We discretize the x and y space by 0.1 increments and we discretize the time by dt = 0.0005 as prescribed by the reference paper. To encapsulate longer-term dynamics and demonstrate the stability of the system, we simulate the solution

from t=0 to t=4 seconds. The analytical solution of the equation is equal to

$$u(x, y, t) = x^2 y^2 exp(it)$$

The domain is defined as $0 \leq x \leq 1$ and $0 \leq y \leq 1$. Given the domain, by using the analytical solution, the boundary conditions of the function are given by:

$$\begin{aligned} &u(0, y, t) = 0, \\ &u(1, y, t) = y^2 exp(it), \\ &u(x, 0, t) = 0, \\ &u(x, 1, t) = x^2 exp(it) \end{aligned}$$

And the initial condition is given by:

$$u(x, y, 0) = x^2 y^2$$

The potential function is given by the following:

$$\omega(x, y) = 1 - \frac{2}{x^2} - \frac{2}{y^2}$$

To avoid an undefined term at $x = 0$ and $y = 0$, we opted to add a slight perturbation of $\epsilon = 10^-8$ to those terms to calculate computable numbers. This system is highly prone to instability. Utilizing higher-order accuracy and implicit integration schemes is crucial to ensure accuracy.

Time	Error
1	5.3474×10^{-3}
2	6.1664×10^{-3}
3	4.1387×10^{-3}
4	1.6512×10^{-3}

Table 1: Error at times t =1,2,3,4

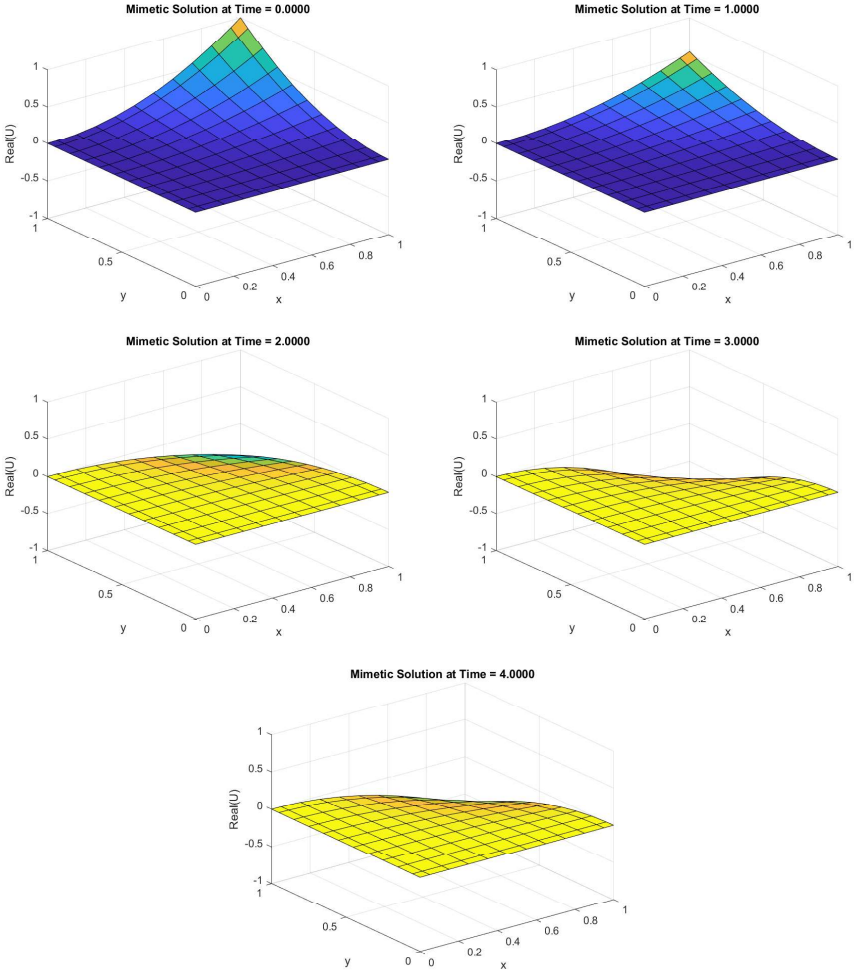

Figure 1: Result for Mimetic Method Simulation at Various t Values

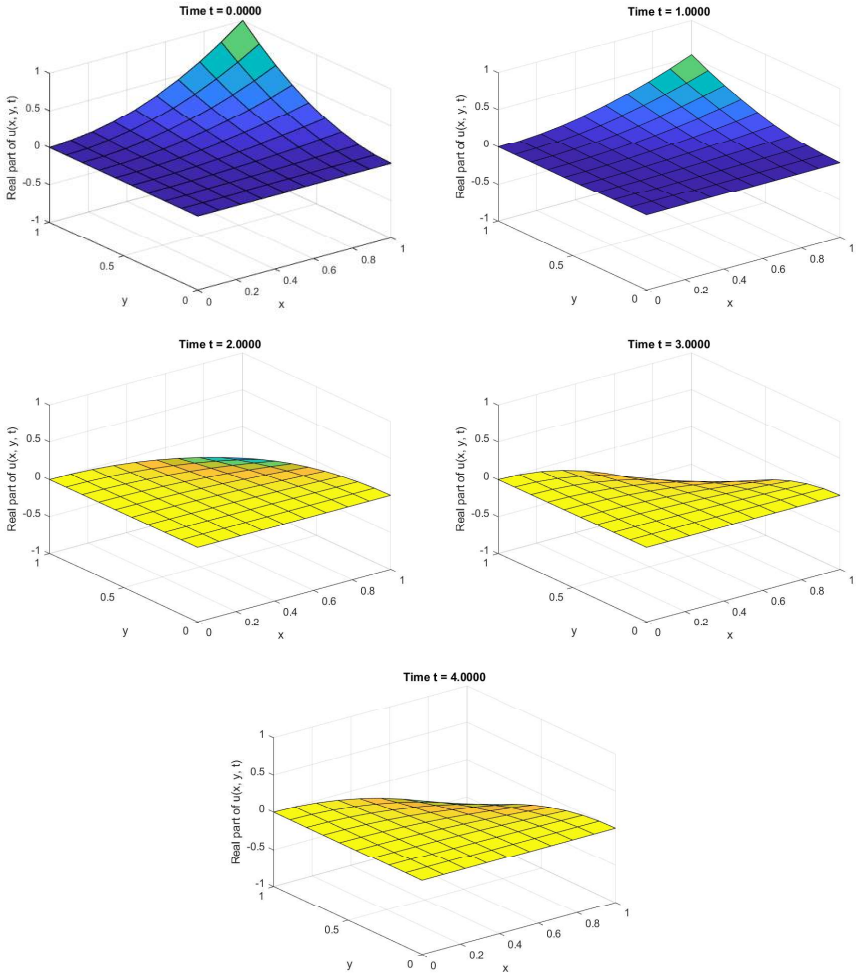

Figure 2: Result for Analytical Method Simulation at Various t Values

The errors are given by the following:

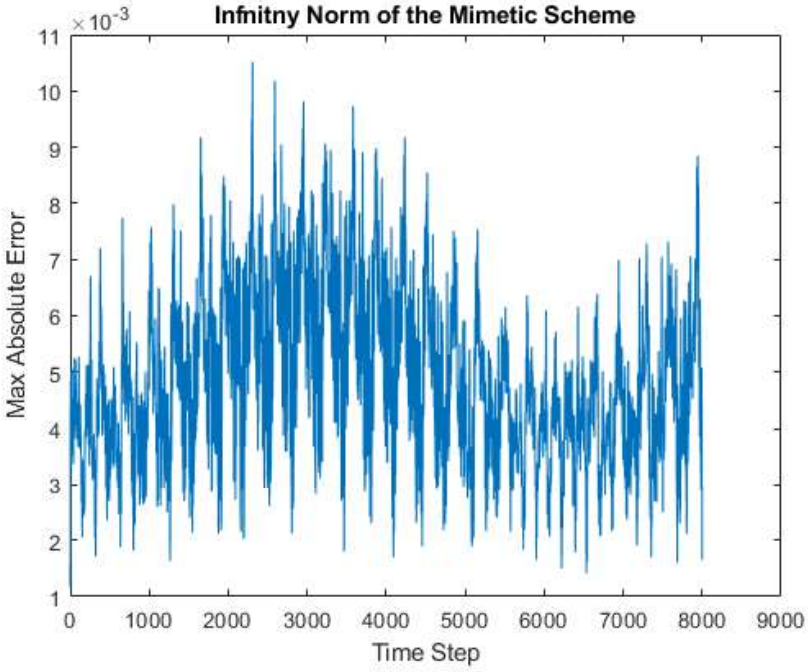

Figure 3: Error Over Time Plot

Conclusion

To conclude, we demonstrate impressive accuracy using the MOLE library and mimetic differences in solving the Schrödinger equation.

References

[1] José E Castillo and Guillermo F Miranda. *Mimetic discretization methods*. CRC Press, 2013.

[2] Johnny Corbino, Miguel A Dumett, and Jose E Castillo. Mole: Mimetic operators library enhanced. *CSRC Report Sep*, 25, 2017.

[3] Mehdi Dehghan and Ali Shokri. A numerical method for two-dimensional schrödinger equation using collocation and radial basis functions. *Computers & Mathematics with Applications*, 54(1):136–146, 2007.

Code

```
clc;
close all;
addpath('../mole_MATLAB');
%%%%%SCHORDINGERS EQUATION AUTHOR: MANI AMANI%%%%%%%%%%%%%%%%


% Parameters
Lxy = 1;
k = 4; %Order of accuracy
m = 10; %Spatial discretization
n = 10;%Spatial discretization
dx = Lxy/m;
dy = Lxy/n;
dt = 0.0005;
tSteps = 2000; % Defining number of time steps for the system to
    simulate

% Staggered grid setup
xgrid = [0 dx/2:dx:Lxy-dx/2 Lxy];
ygrid = [0 dy/2:dy:Lxy-dy/2 Lxy];
[X, Y] = meshgrid(xgrid, ygrid);

% Mimetic Laplacian operator
L = lap2D(k, m, dx, n, dy);
U_first = X.^2 .* Y.^2;
W = 1 - 2 ./ (X.^2+1e-8) - 2 ./ (Y.^2+1e-8); %To Prevent negative
    infinity in the potential function
H = @(U) (L * U + (diag(W(:)) * U)) * 1i; %Spatial discretization
    using mimetics
U_old = U_first(:);

% Video setup
videoFile = 'Mimetic_simulation.mp4';
v = VideoWriter(videoFile, 'MPEG-4');
v.FrameRate = 30;
v.Quality = 100;
%open(v);
errors = zeros(tSteps, 1); %Saving all the errors
%%%%%%%%%%%%%%%%%%%% Time Loop Intiation %%%%%%%%%%%%%%%%%%%%%%
for ii = 0:tSteps
    % Time-stepping with RK4
    k1 = dt * H(U_old);
    k2 = dt * H(U_old + 0.5 * k1);
    k3 = dt * H(U_old + 0.5 * k2);
    k4 = dt * H(U_old + k3);
    U_new = U_old + (k1 + 2*k2 + 2*k3 + k4) / 6;

    % Boundary conditions and reshaping
    U_matrix = reshape(U_new, m+2, n+2);
    t = ii * dt;
    U_matrix(:, end) = xgrid.^2 .* exp(1i * t);
    U_matrix(end, :) = ygrid.^2 .* exp(1i * t);
    U_matrix(1, :) = 0;
    U_matrix(:, 1) = 0;
    U_new = reshape(U_matrix, (n+2)*(m+2),1);
```

```
53
54      % Visualization
55      figure(1);
56      surf(X, Y, real(U_matrix)); %Both real part and imaginary parts
         can be depicted, use imag() for imaginary
57      title(['Mimetic Solution at Time = ', num2str(t, '%.4f')]);
58      xlabel('x'); ylabel('y'); zlabel('Real(U)');
59      zlim([-1, 1]);
60      drawnow;
61
62      % Capture and save frame
63      %frame = getframe(gcf);
64      %writeVideo(v, frame);
65
66      % Error calculation
67      U_analytical = X.^2 .* Y.^2 .* exp(1i * t);
68      errors(ii+1) = norm(real(U_matrix) - real(U_analytical),'inf');
69
70      % Save figure at time t = 1s, Just to get the Last frame for
        reference
71      if abs(t - 1.0) < 0.0005 %Some tolerence, can be changed
        depending on the desired time
72          savefig(gcf, 'Figure_at_Time_1.fig'); %Figure name
73          print('Figure_at_Time_1', '-dpng');
74      end
75
76      U_old = U_new; %Update the vector for further calculation
77  end
78
79  % Close video and display info
80  %close(v);
81  disp(['Video saved as ', videoFile]);
82  disp('Maximum error at any timestep:');
83  max(errors)
84  figure;
85  plot(errors);
86  title('Error over Time');
87  xlabel('Time Step');
88  ylabel('Max Absolute Error');
```