PHYS 410 Project 2

Gavin Pringle, 56401938

November 30, 2024

Introduction

In this project, the time-dependent Schrödinger equation is solved numerically in both one dimension and two dimensions. In both cases, solutions account for a time-independent potential term V, which takes the form of a rectangular barrier or well, a double slit (in 2d only), or is zero everywhere. The MATLAB script `sch_1d_cn.m` implements the Crank-Nicolson discretization approach to solve the Schrödinger equation in 1d, while the script `sch_2d_adi.m` implements the alternating-direction-implicit (ADI) method to solve the equation in 2d.

The 1d case is tested by conducting convergence testing in the file `ctest_1d.m` which checks for solution convergence among increasing discretization levels. A similar convergence test is done for two dimensions in the file `ctest_2d.m`. For the 1d case, the solution "excess fractional probability" is also examined in the files `barrier_survey.m` and `well_survey.m`, which provides insights into how much time the quantum particle is spending in a certain location. Lastly, videos of the 2d wave function scattering off a rectangular barrier or well, and producing self-interference through a double slit are created using the scripts `video_rec_bar.m`, `video_rec_well.m`, and `video_double_slit.m`.

Review of Theory

1d Schrödinger Equation

The 1d Schrödinger Equation PDE is given by the following equation:

$$i\psi(x,t)_t = -\psi_{xx} + V(x,t)\psi \tag{1}$$

where the wave function, $\psi(x,t)$, is complex. The equation is to be solved on the domain

$$0 \le x \le 1, \quad 0 \le t \le t_{\max}$$

subject to initial and boundary conditions

$$\psi(x,0) = \psi_0(x) \tag{2}$$

$$\psi(0,t) = \psi(1,t) = 0 \tag{3}$$

The family of exact solutions to (1) is

$$\psi(x,t) = e^{-im^2\pi^2 t}\sin(m\pi x) \tag{4}$$

where m is a positive integer.

Since the modulus squared of the wave function represents the probability density, $\rho = |\psi|^2 = \psi\psi^*$, the "running integral" of the probability density represents the probability that the particle is to the left of x at any given time t:

$$P(x,t) = \int_0^x \psi(\tilde{x},t)\psi^*(\tilde{x},t)d\tilde{x} \tag{5}$$

Note that equation (5) only computes the correct probability if the wave function is properly normalized such that $P(1,t) = 1$. Even if it is not so normalized, we should have

$$P(1,t) = \text{conserved to level of solution error}$$

2d Schrödinger Equation

Numerical Approach

1d Schrödinger Equation

2d Schrödinger Equation

$$\left(1 - i\frac{\Delta t}{2}\partial_{xx}^h\right)\psi_{i,j}^{n+\frac{1}{2}} = \left(1 + i\frac{\Delta t}{2}\partial_{xx}^h\right)\left(1 + i\frac{\Delta t}{2}\partial_{yy}^h - i\frac{\Delta t}{2}V_{i,j}\right)\psi_{i,j}^n,$$
$$i = 2,3,\ldots,n_x - 1, \quad j = 2,3,\ldots,n_y - 1, \quad n = 1,2,\ldots,n_t - 1. \tag{6}$$

$$\left(1 - i\frac{\Delta t}{2}\partial_{yy}^h + i\frac{\Delta t}{2}V_{i,j}\right)\psi_{i,j}^{n+1} = \psi_{i,j}^{n+\frac{1}{2}},$$
$$i = 2,3,\ldots,n_x - 1, \quad j = 2,3,\ldots,n_y - 1, \quad n = 1,2,\ldots,n_t - 1. \tag{7}$$

Implementation

1d Tridiagonal System

2d Tridiagonal System

Results

1d Schrödinger Equation

2d Schrödinger Equation

Conclusions

Generative AI was used to help with understanding how to use MATLAB's `contourf` function for making videos of numerical experiments in 2d. It was also used for help with typesetting this document.

Appendix A - sch_1d_cn.m Code

```
% sch_1d_cn: Solves 1D Schrödinger equation using O(dt^2,dx^2)
% Crank-Nicolson implicit scheme.
%
% Inputs:
%
%   tmax:   Maximum integration time
%   level:  Discretization level
%   lambda: dt/dx
%   idtype: Selects initial data type
%   idpar:  Vector of initial data parameters
%   vtype:  Selects potential type
%   vpar:   Vector of potential parameters
%
% Outputs:
%
%   x:      Column vector of x coordinates [nx]
%   t:      Column vector of t coordinates [nt]
%   psi:    Array of computed psi values [nt x nx]
%   psire:  Array of computed psi_re values [nt x nx]
%   psiim:  Array of computed psi_im values [nt x nx]
%   psimod: Array of computed sqrt(psi psi*) values [nt x nx]
%   prob:   Array of computed running integral values [nt x nx]
%   v:      Column vector of potential values [nx]
function [x t psi psire psiim psimod prob v] = ...
    sch_1d_cn(tmax, level, lambda, idtype, idpar, vtype, vpar)

    % Define mesh and derived parameters
    nx = 2^level + 1;
    x = linspace(0.0, 1.0, nx);
    dx = x(2) - x(1);
    dt = lambda * dx;
    nt = round(tmax / dt) + 1;
    t = (0 : nt-1) * dt;

    % Initialize solution, and set initial data
    psi = zeros(nt, nx);
    if idtype == 0
        % Exact family
        psi(1, :) = sin(idpar(1) * pi * x);
    elseif idtype == 1
        % Boosted Gaussian
        psi(1, :) = exp(1i * idpar(3) * x) .* ...
                    exp(-((x - idpar(1)) ./ idpar(2)) .^ 2);
    else
        fprintf('sch_1d_cn: Invalid idtype=%d\n', idtype);
        return
    end
    % Set first and last values of initial data to zero
    psi(1, 1) = 0;
    psi(1, nx) = 0;

    % Initial storage for prob and calculate for initial time
    prob = zeros(nt, nx);
    for j = 2 : nx
        prob(1, j) = trapz(x(1:j), abs(psi(1, 1:j)).^2);
    end
```

```
57
58      % Initialize potential
59      v = zeros(1,nx);
60      if vtype == 0
61          % No potential - leave unchanged
62      elseif vtype == 1
63          % Rectangular barrier or well
64          v(x > vpar(1) & x < vpar(2)) = vpar(3);
65      else
66          fprintf('sch_1d_cn: Invalid vtype=%d\n', vtype);
67          return
68      end
69
70      % Initialize storage for RHS
71      f  = zeros(nx,1);
72
73      % Set up tridiagonal system
74      dl = 0.5/dx^2 * ones(nx, 1);
75      d  = (1i/dt - 1/dx^2 - 0.5*v.') .* ones(nx,1);
76      du = dl;
77      % Fix up boundary cases
78      d(1) = 1.0;
79      du(2) = 0.0;
80      dl(nx-1) = 0.0;
81      d(nx) = 1.0;
82      % Define sparse matrix
83      A = spdiags([dl d du], -1:1, nx, nx);
84
85      % Compute solution using CN scheme
86      for n = 1 : nt-1
87          % Define RHS of linear system
88          f(2:nx-1) = psi(n, 2:nx-1) .* (1i/dt + 1/dx^2 + 0.5*v(2:nx-1)) ...
89                      + (-0.5/dx^2) * (psi(n, 1:nx-2) + psi(n, 3:nx));
90          f(1) = 0.0;
91          f(nx) = 0.0;
92          % Solve system, thus updating approximation to next time step
93          psi(n+1, :) = A \ f;
94          % Set first and last values to zero
95          psi(n+1, 1) = 0;
96          psi(n+1, nx) = 0;
97
98          % Calculate prob each time step
99          for j = 2 : nx
100             prob(n+1, j) = trapz(x(1:j), abs(psi(n+1, 1:j)).^2);
101         end
102     end
103
104     % Compute real, imaginary, and modulus of each entry in psi
105     psire  = real(psi);
106     psiim  = imag(psi);
107     psimod = abs(psi);
108
109     % Convert to column vectors
110     x = x.';
111     t = t.';
112     v = v.';
113 end
```

Appendix B - ctest_1d.m Code

```
%% 1.4 - 1d Convergence Testing

close all;
clear; clc;
format long;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convergence Test #1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Simulation maximum time
tmax = 0.25;
% Discretization levels
minlevel = 6;
maxlevel = 9;
% Delta t by Delta x ratio
lambda = 0.1;

% idtype = 0   -> Exact family (sine wave)
% idtype = 1   -> Boosted Gaussian
idtype = 0;
idpar = [3]; % m = 3

% vtype = 0   -> No potential
% vtype = 1   -> Rectangular barrier or well
vtype = 0;
vpar = zeros(1,3);

% Perform computation at various levels of discretization, store
% results in cell arrays ...
for l = minlevel : maxlevel
    % Compute the solution
    [x{l} t{l} psi{l} psire{l} psiim{l} psimod{l} prob{l} v{l}] ...
        = sch_1d_cn(tmax, l, lambda, idtype, idpar, vtype, vpar)

    [nt{l}, nx{l}] = size(psi{l});

    % Since idtype == 0, compute exact solution
    psixct{l} = zeros(nt{l}, nx{l});
    for n = 1 : nt{l}
        psixct{l}(n,:) = exp(-1i * idpar(1)^2 * pi^2 * t{l}(n)) ...
                         * sin(idpar(1) * pi * x{l});
    end
    % Compute exact errors and their rms values for later
    Epsi{l} = psixct{l} - psi{l};
    rms_Epsi{l} = rms(abs(Epsi{l}), 2);
end

% Calculating the level-to-level differences, taking every second
% value of the larger length array
dpsi6 = downsample(downsample(psi{7}, 2).', 2).' - psi{6};
dpsi7 = downsample(downsample(psi{8}, 2).', 2).' - psi{7};
dpsi8 = downsample(downsample(psi{9}, 2).', 2).' - psi{8};

% Compute l-2 norm of each dpsi, resulting in functions of t
rms_dpsi6 = rms(abs(dpsi6), 2);
```

```
rms_dpsi7 = rms(abs(dpsi7), 2);
rms_dpsi8 = rms(abs(dpsi8), 2);

% Plot scaled errors for different discretization levels
fig1 = figure;
rho = 4;
hold on
plot(t{6}, rms_dpsi6, 'LineWidth', 2);
plot(t{7}, rho*rms_dpsi7, 'LineWidth', 2);
plot(t{8}, rho^2*rms_dpsi8, 'LineWidth', 2);
xlabel("Time");
ylabel("l-2 norm of difference between level");
legend('||dΨ^6||', '4 * ||dΨ^7||', '4^2 * ||dΨ^8||', 'Location', 'best');
title({"1d Schrodinger equation convergence test - Exact family"
       "l-2 norm of difference between level l solutions"
       "idtype = 0, vtype = 0, tmax = 0.25, lambda = 0.1, 6 <= l <= 9"});
ax = gca;
ax.FontSize = 12;

% Plot scaled exact errors for different discretization levels
fig2 = figure;
rho = 4;
hold on
plot(t{6}, rms_Epsi{6}, 'LineWidth', 2);
plot(t{7}, rho*rms_Epsi{7}, 'LineWidth', 2);
plot(t{8}, rho^2*rms_Epsi{8}, 'LineWidth', 2);
plot(t{9}, rho^3*rms_Epsi{9}, 'LineWidth', 2);
xlabel("Time");
ylabel("l-2 norm of exact error");
legend('||EΨ^6||', '4 * ||EΨ^7||', '4^2 * ||EΨ^8||', '4^3 * ||EΨ^9||'...
        , 'Location', 'best');
title({"1d Schrodinger equation convergence test - Exact family"
       "l-2 norm of exact error for each level l"
       "idtype = 0, vtype = 0, tmax = 0.25, lambda = 0.1, 6 <= l <= 9"});
ax = gca;
ax.FontSize = 12;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convergence Test #2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

% Simulation maximum time
tmax = 0.01;
% Discretization levels
minlevel = 6;
maxlevel = 9;
% Delta t by Delta x ratio
lambda = 0.01;

% idtype = 0   ->  Exact family (sine wave)
% idtype = 1   ->  Boosted Gaussian
idtype = 1;
idpar = [0.50  0.075  0.0];

% vtype = 0   ->  No potential
```

```
% vtype = 1   ->  Rectangular barrier or well
vtype = 0;
vpar = zeros(1,3);

% Perform computation at various levels of discretization, store
% results in cell arrays ...
for l = minlevel : maxlevel
    % Compute the solution
    [x{l} t{l} psi{l} psire{l} psiim{l} psimod{l} prob{l} v{l}] ...
        = sch_1d_cn(tmax, l, lambda, idtype, idpar, vtype, vpar);

    [nt{l}, nx{l}] = size(psi{l});
end

% Calculating the level-to-level differences, taking every second
% value of the larger length array
dpsi6 = downsample(downsample(psi{7}, 2).', 2).' - psi{6};
dpsi7 = downsample(downsample(psi{8}, 2).', 2).' - psi{7};
dpsi8 = downsample(downsample(psi{9}, 2).', 2).' - psi{8};

% Compute l-2 norm of each dpsi, resulting in functions of t
rms_dpsi6 = rms(abs(dpsi6), 2);
rms_dpsi7 = rms(abs(dpsi7), 2);
rms_dpsi8 = rms(abs(dpsi8), 2);

% Plot scaled errors for different discretization levels
fig3 = figure;
rho = 4;
hold on
plot(t{6}, rms_dpsi6, 'LineWidth', 2);
plot(t{7}, rho*rms_dpsi7, 'LineWidth', 2);
plot(t{8}, rho^2*rms_dpsi8, 'LineWidth', 2);
xlabel("Time");
ylabel("l-2 norm of difference between level");
legend('||dΨ^6||', '4 * ||dΨ^7||', '4^2 * ||dΨ^8||', 'Location', 'best');
title({"1d Schrodinger equation convergence test - Boosted Gaussian"
       "l-2 norm of difference between level l solutions"
       "idtype = 1, vtype = 0, tmax = 0.01, lambda = 0.01, 6 <= l <= 9"});
ax = gca;
ax.FontSize = 12;
```

Appendix C - barrier_survey.m Code

```
%% 1.5.1 - 1d Barrier Survey

close all;
clear; clc;
format long;

% Simulation maximum time
tmax = 0.10;
% Discretization level
level = 9;
% Delta t by Delta x ratio
lambda = 0.01;

% idtype = 0   ->  Exact family (sine wave)
% idtype = 1   ->  Boosted Gaussian
idtype = 1;
idpar = [0.40, 0.075, 20.0];

% vtype = 0   ->  No potential
% vtype = 1   ->  Rectangular barrier or well
vtype = 1;
xmin = 0.6;
xmax = 0.8;
lnV0 = linspace(-2, 5, 251);

% Survey range
x1 = 0.8;
x2 = 1.0;

for idx = 1 : length(lnV0)
    % Get this iteration's vpar
    vpar = [xmin, xmax, exp(lnV0(idx))];

    % Compute the solution
    [x{idx} t{idx} psi{idx} psire{idx} psiim{idx} psimod{idx} prob{idx} v{
        idx}] ...
       = sch_1d_cn(tmax, level, lambda, idtype, idpar, vtype, vpar);

    % Compute temporal average of probability matrix
    P_bar{idx} = mean(prob{idx});
    % Normalize the temporal average
    P_bar{idx} = P_bar{idx} / P_bar{idx}(end);

    % Compute indices of x1 and x2 in the array x
    % This only needs to be done once
    if idx == 1
        [~, x1_loc] = min(abs(x{idx} - x1));
        [~, x2_loc] = min(abs(x{idx} - x2));
    end

    % Compute excess fractional probability and its logarithm
    Fe_bar{idx} = (P_bar{idx}(x2_loc) - P_bar{idx}(x1_loc)) / ...
                  (x{idx}(x2_loc) - x{idx}(x1_loc));
    lnFe_bar{idx} = log(Fe_bar{idx})
end

```

```
56 fig1 = figure;
57 plot(lnV0, cell2mat(lnFe_bar), 'LineWidth', 2)
58 title({"Barrier Survey - Log-log plot of excess fractional probablity vs. V_0"
59         "Barrier between x = 0.6 and x = 0.8. x_1 = 0.8, x_2 = 1.0"})
60 xlabel('$$\mathbf{\ln(V_0)}$$', 'interpreter', 'latex')
61 ylabel('$$\mathbf{\ln(\overline{F_e}(x_1, x_2))}$$', 'interpreter', 'latex')
62 ax = gca;
63 ax.FontSize = 12;
```

Appendix D - well_survey.m Code

```
%% 1.5.2 - 1d Well Survey

close all;
clear; clc;
format long;

% Simulation maximum time
tmax = 0.10;
% Discretization level
level = 9;
% Delta t by Delta x ratio
lambda = 0.01;

% idtype = 0   ->  Exact family (sine wave)
% idtype = 1   ->  Boosted Gaussian
idtype = 1;
idpar = [0.40, 0.075, 0.0];

% vtype = 0   ->  No potential
% vtype = 1   ->  Rectangular barrier or well
vtype = 1;
xmin = 0.6;
xmax = 0.8;
lnV0 = linspace(2, 10, 251);

% Survey range
x1 = 0.6;
x2 = 0.8;

for idx = 1 : length(lnV0)
    % Get this iteration's vpar
    vpar = [xmin, xmax, -exp(lnV0(idx))];

    % Compute the solution
    [x{idx} t{idx} psi{idx} psire{idx} psiim{idx} psimod{idx} prob{idx} v{
        idx}] ...
       = sch_1d_cn(tmax, level, lambda, idtype, idpar, vtype, vpar);

    % Compute temporal average of probability matrix
    P_bar{idx} = mean(prob{idx});
    % Normalize the temporal average
    P_bar{idx} = P_bar{idx} / P_bar{idx}(end);

    % Compute indices of x1 and x2 in the array x
    % This only needs to be done once
    if idx == 1
        [~, x1_loc] = min(abs(x{idx} - x1));
        [~, x2_loc] = min(abs(x{idx} - x2));
    end

    % Compute excess fractional probability and its logarithm
    Fe_bar{idx} = (P_bar{idx}(x2_loc) - P_bar{idx}(x1_loc)) / ...
                  (x{idx}(x2_loc) - x{idx}(x1_loc));
    lnFe_bar{idx} = log(Fe_bar{idx})
end

```

```
fig2 = figure;
plot(lnV0, cell2mat(lnFe_bar), 'LineWidth', 2)
title({"Well Survey - Log-log plot of excess fractional probablity vs. V_0"
       "Well between x = 0.6 and x = 0.8. x_1 = 0.6, x_2 = 0.8"})
xlabel('$$\mathbf{\ln(V_0)}$$', 'interpreter', 'latex')
ylabel('$$\mathbf{\ln(\overline{F_e}(x_1, x_2))}$$', 'interpreter', 'latex'
    )
ax = gca;
ax.FontSize = 12;
```

Appendix E - sch_2d_adi.m Code

```
% sch_2d_adi: Solves 2D Schrödinger equation using ADI scheme.
%
% Inputs:
%
%   tmax:   Maximum integration time
%   level:  Discretization level
%   lambda: dt/dx
%   idtype: Selects initial data type
%   idpar:  Vector of initial data parameters
%   vtype:  Selects potential type
%   vpar:   Vector of potential parameters
%
% Outputs:
%
%   x:      Column vector of x coordinates [nx]
%   y:      Column vector of y coordinates [ny]
%   t:      Column vector of t coordinates [nt]
%   psi:    Array of computed psi values [nt x nx x ny]
%   psire:  Array of computed psi_re values [nt x nx x ny]
%   psiim:  Array of computed psi_im values [nt x nx x ny]
%   psimod: Array of computed sqrt(psi psi*) values [nt x nx x ny]
%   v:      Array of potential values [nx x ny]
function [x y t psi psire psiim psimod v] = ...
    sch_2d_adi(tmax, level, lambda, idtype, idpar, vtype, vpar)

    % Define mesh and derived parameters
    nx = 2^level + 1;                ny = nx;
    x  = linspace(0.0, 1.0, nx);     y  = x;
    dx = x(2) - x(1);                dy = dx;
    dt = lambda * dx;
    nt = round(tmax / dt) + 1;
    t  = (0 : nt-1) * dt;

    % Define meshgrid for populating psi(x,y,0) and V(x,y)
    [X, Y] = meshgrid(x, y);

    % Initialize solution, and set initial data
    psi = zeros(nt, nx, ny);
    if idtype == 0
        % Exact family
        psi(1, :, :) = sin(idpar(1)*pi*x)' * sin(idpar(2)*pi*y);
    elseif idtype == 1
        % Boosted Gaussian
        % Create variable names for function parameters
        x0      = idpar(1);      y0 = idpar(2);
        delta_x = idpar(3); delta_y = idpar(4);
        p_x     = idpar(5);     p_y = idpar(6);

        % Calculate psi(x, y, 0)
        psi_0 = exp(1i*p_x*X) .* exp(1i*p_y*Y) ...
              .* exp(-(((X - x0).^2)/delta_x^2 + ((Y - y0).^2)/delta_y^2));
        psi(1, :, :) = reshape(psi_0, [1, nx, ny]);
    else
        fprintf('sch_2d_adi: Invalid idtype=%d\n', idtype);
        return
    end
```

```
57      % Set boundary conditions of initial data to zero
58      % t = 0:   psi(0,y,t) = psi(1,y,t) = psi(x,0,t) = psi(x,1,t) = 0
59      psi(1, 1, :)  = 0;
60      psi(1, :, 1)  = 0;
61      psi(1, nx, :) = 0;
62      psi(1, :, ny) = 0;
63
64      % Initialize time-independent potential
65      v = zeros(nx,ny);
66      if vtype == 0
67          % No potential - leave unchanged
68      elseif vtype == 1
69          % Rectangular barrier or well
70          % Create variable names for function parameters
71          x_min = vpar(1);    x_max = vpar(2);
72          y_min = vpar(3);    y_max = vpar(4);
73          Vc    = vpar(5);
74
75          % Calculate V(x, y)
76          v((X >= x_min & X <= x_max) & (Y >= y_min & Y <= y_max)) = Vc;
77      elseif vtype == 2
78          % Double slit
79          % Create variable names for function parameters
80          x1 = vpar(1);    x2 = vpar(2);
81          x3 = vpar(3);    x4 = vpar(4);
82          Vc = vpar(5);
83          j_prime = (ny - 1)/4 + 1;
84
85          % Calculate V(x, y)
86          Vc_indices = (x <= x1) | (x >= x2 & x <= x3) | (x >= x4);
87          v(j_prime, Vc_indices) = Vc;
88          v(j_prime + 1, Vc_indices) = Vc;
89      else
90          fprintf('sch_2d_adi: Invalid vtype=%d\n', vtype);
91          return
92      end
93
94      % Define sparse matrix diagonals for first ADI eqn
95      dl = (-1i*dt/(2*dx^2)) * ones(nx, 1);
96      d  = (1 + 1i*dt/(dx^2)) * ones(nx, 1);
97      du = dl;
98      % Impose boundary conditions
99      d(1)    = 1.0;
100     du(2)   = 0.0;
101     dl(nx-1) = 0.0;
102     d(nx)   = 1.0;
103     % Compute sparse matrix for first ADI eqn
104     A_half = spdiags([dl d du], -1:1, nx, nx);
105
106     % Loop that iterates each time step
107     for n = 1:nt-1
108         % reshape psi to create a 2d matrix at this timestep
109         psi_n = reshape(psi(n,:,:), nx, ny);
110         % Create matrix for psi^(n+1/2)
111         psi_half = zeros(nx,ny);
112
113         % Solve tridiagonal system for each j (row)
114         for j = 2:ny-1
```

```
115             % Array for holding the RHS of the first ADI eqn
116             f = zeros(nx,1);
117 
118             % Compute RHS of first ADI eqn in stages.
119             f(2:nx-1) = (1i*dt/(2*dy^2)) * (psi_n(2:nx-1, j+1) + psi_n(2:nx
                    -1, j-1)) ...
120                     + (1 - 1i*dt*(1/dy^2 + v(2:nx-1,j)/2)) .* psi_n(2:nx
                        -1, j);
121             f(2:nx-1) = (1i*dt/(2*dx^2)) * (f(1:nx-2) + f(3:nx)) + ...
122                       (1 - 1i*dt/dx^2) * f(2:nx-1);
123 
124             % Impose boundary conditions
125             f(1) = 0.0;
126             f(nx) = 0.0;
127 
128             % Solve first ADI system
129             psi_half(:,j) = A_half \ f;
130             % Impose boundary conditions
131             psi_half(1,:)  = 0.0;
132             psi_half(:,1)  = 0.0;
133             psi_half(nx,:) = 0.0;
134             psi_half(:,ny) = 0.0;
135         end
136 
137         % Define upper and lower sparse matrix diagonals for second ADI eqn
138         dl = (-1i*dt/(2*dy^2)) * ones(ny, 1);
139         du = dl;
140         % Impose boundary conditions
141         du(2)    = 0.0;
142         dl(ny-1) = 0.0;
143 
144         % Solve tridiagonal system for each i (column)
145         for i = 2:nx-1
146             % Define middle sparse matrix diagonal for second ADI eqn
147             v_i = reshape(v(i,:), ny, 1);
148             d = 1 + 1i*dt/dy^2 + (1i*dt/2)*v_i;
149             % Impose boundary conditions
150             d(1)  = 1.0;
151             d(ny) = 1.0;
152 
153             % Compute sparse matrix for second ADI eqn
154             A_full = spdiags([dl d du], -1:1, ny, ny);
155 
156             % Compute RHS of second ADI eqn. BCs already imposed previously
157             f = reshape(psi_half(i,:), ny, 1);
158 
159             % Solve second ADI system
160             psi(n+1, i, :) = A_full \ f;
161             % Impose boundary conditions
162             psi(n+1, 1, :)  = 0.0;
163             psi(n+1, :, 1)  = 0.0;
164             psi(n+1, nx, :) = 0.0;
165             psi(n+1, :, ny) = 0.0;
166         end
167     end
168 
169     % Compute real, imaginary, and modulus of each entry in psi
170     psire = real(psi);
```

```
171         psiim = imag(psi);
172         psimod = abs(psi);
173 
174         % Convert to column vectors
175         x = x.';
176         y = y.';
177         t = t.';
178 end
```

Appendix F - ctest_2d.m Code

```
%% 2.4 - 2d Convergence Testing

close all;
clear; clc;
format long;

% Simulation maximum time
tmax = 0.05;
% Discretization levels
minlevel = 6;
maxlevel = 9;
% Delta t by Delta x ratio
lambda = 0.05;

% idtype = 0   -> Exact family (sine wave)
% idtype = 1   -> Boosted Gaussian
idtype = 0;
idpar = [2, 3];
mx = idpar(1); my = idpar(2);

% vtype = 0   -> No potential
% vtype = 1   -> Rectangular barrier or well
vtype = 0;
vpar = zeros(1,5);

% Perform computation at various levels of discretization, store
% results in cell arrays ...
for l = minlevel : maxlevel
    % Compute the solution
    [x{l} y{l} t{l} psi{l} psire{l} psiim{l} psimod{l} v{l}] = ...
        sch_2d_adi(tmax, l, lambda, idtype, idpar, vtype, vpar)

    [nt{l}, nx{l}, ny{l}] = size(psi{l});

    % Define meshgrid for computing exact psi(x,y,t)
    [X{l}, Y{l}] = meshgrid(x{l}.', y{l}.');

    % Compute exact solution
    psixct{l} = zeros(nt{l}, nx{l}, ny{l});
    for n = 1 : nt{l}
        psixct_n{l} = exp(-1i*(mx^2 + my^2)*pi^2*t{l}(n)) * ...
                      sin(mx*pi*X{l}) .* sin(my*pi*Y{l});
        % Enforce boundary conditions
        psixct_n{l}(1, :)     = 0.0;
        psixct_n{l}(:, 1)     = 0.0;
        psixct_n{l}(nx{l}, :) = 0.0;
        psixct_n{l}(:, ny{l}) = 0.0;
        psixct{l}(n, :, :) = reshape(psixct_n{l}, [1, nx{l}, ny{l}]);
    end

    % Compute exact errors and their rms values for later
    Epsi{l} = psixct{l} - psi{l};
    Epsi_2d{l} = reshape(Epsi{l}, nt{l}, nx{l}*ny{l});
    rms_Epsi{l} = rms(abs(Epsi_2d{l}), 2);

    % Downsample each psi for differencing
```

```
57      psi_ds{l} = psi{l}(1:2:end, 1:2:end, 1:2:end);
58
59      % Flatten each 3d array into a 2d array
60      psi_2d{l} = reshape(psi{l}, nt{l}, nx{l}*ny{l});
61      psi_ds_2d{l} = reshape(psi_ds{l}, (nt{l}-1)/2+1, ...
62                             ((nx{l}-1)/2+1)*((ny{l}-1)/2+1));
63  end
64
65  % Calculating the level-to-level differences, taking every second
66  % value of the larger length array
67  dpsi6 = psi_ds_2d{7} - psi_2d{6};
68  dpsi7 = psi_ds_2d{8} - psi_2d{7};
69  dpsi8 = psi_ds_2d{9} - psi_2d{8};
70
71  % Compute l-2 norm of each dpsi, resulting in functions of t
72  rms_dpsi6 = rms(abs(dpsi6), 2);
73  rms_dpsi7 = rms(abs(dpsi7), 2);
74  rms_dpsi8 = rms(abs(dpsi8), 2);
75
76  % Plot scaled errors for different discretization levels
77  fig1 = figure;
78  rho = 4;
79  hold on
80  plot(t{6}, rms_dpsi6, 'LineWidth', 2);
81  plot(t{7}, rho*rms_dpsi7, 'LineWidth', 2);
82  plot(t{8}, rho^2*rms_dpsi8, 'LineWidth', 2);
83  xlabel("Time");
84  ylabel("l-2 norm of difference between level");
85  legend('||dΨ^6||', '4 * ||dΨ^7||', '4^2 * ||dΨ^8||', 'Location', 'best');
86  title({"2d Schrodinger equation convergence test - Exact family"
87         "l-2 norm of difference between level l solutions"
88         "idtype = 0, vtype = 0, tmax = 0.05, lambda = 0.05, 6 <= l <= 9"});
89  ax = gca;
90  ax.FontSize = 12;
91
92  % Plot scaled exact errors for different discretization levels
93  fig2 = figure;
94  rho = 4;
95  hold on
96  plot(t{6}, rms_Epsi{6}, 'LineWidth', 2);
97  plot(t{7}, rho*rms_Epsi{7}, 'LineWidth', 2);
98  plot(t{8}, rho^2*rms_Epsi{8}, 'LineWidth', 2);
99  plot(t{9}, rho^3*rms_Epsi{9}, 'LineWidth', 2);
100 xlabel("Time");
101 ylabel("l-2 norm of exact error");
102 legend('||EΨ^6||', '4 * ||EΨ^7||', '4^2 * ||EΨ^8||', '4^3 * ||EΨ^9||'...
103         , 'Location', 'best');
104 title({"2d Schrodinger equation convergence test - Exact family"
105        "l-2 norm of exact error for each level l"
106        "idtype = 0, vtype = 0, tmax = 0.05, lambda = 0.05, 6 <= l <= 9"});
107 ax = gca;
108 ax.FontSize = 12;
```

Appendix G - video_rec_bar.m Code

```
%% 2.5 - 2d Video of Scattering off Rectangular Barrier

close all;
clear; clc;
format long;

% Simulation maximum time
tmax = 0.05;
% Discretization level
level = 8;
% Delta t by Delta x ratio
lambda = 0.05;

% idtype = 0   -> Exact family (sine wave)
% idtype = 1   -> Boosted Gaussian
idtype = 1;
%x0      = idpar(1);      y0 = idpar(2);
%delta_x = idpar(3); delta_y = idpar(4);
%p_x     = idpar(5);      p_y = idpar(6);
idpar = [0.5, 0.3, 0.1, 0.1, 0.0, 30];

% vtype = 0   -> No potential
% vtype = 1   -> Rectangular barrier or well
% vtype = 2   -> Double Slit
vtype = 1;
x_min = 0.2;   x_max = 0.8;
y_min = 0.7;   y_max = 0.8;
%Vc    = vpar(5);
vpar = [x_min, x_max, y_min, y_max, 1e8];

% Compute solution
[x y t psi psire psiim psimod v] = ...
    sch_2d_adi(tmax, level, lambda, idtype, idpar, vtype, vpar);

% Dimensions of matrix
[nt, nx, ny] = size(psimod);

% Create a VideoWriter object
video = VideoWriter('../../output/problem2/rec_bar.avi');
video.FrameRate = 30;
open(video);

% Create a figure and define where the rectangle indicating where
% the potential barrier is will be drawn
figure;
rect_xmin = x_min * (nx - 1) + 1;
rect_ymin = y_min * (ny - 1) + 1;
rect_xlen = (x_max - x_min) * (nx - 1);
rect_ylen = (y_max - y_min) * (ny - 1);

% Loop over time steps
for n = 1:nt
    % reshape psi to create a 2d matrix at this timestep
    psi_n = reshape(psimod(n,:,:), nx, ny);

    % Create filled contour plot
```

```
57      contourf(psi_n, 20, 'LineStyle', 'none');
58      colormap("default");
59      xlabel('x');
60      ylabel('y');
61      title({'2d Schrödinger Equation Simulation'
62             '|ψ| Scattering off a Rectangular Barrier'
63             ['tmax = ', num2str(tmax), ', level = ', num2str(level), ...
64              ', lambda = ', num2str(lambda), ', idpar = [', ...
65              num2str(idpar(1)), ' ', num2str(idpar(2)), ' ', ...
66              num2str(idpar(3)), ' ', num2str(idpar(4)), ' ', ...
67              num2str(idpar(5)), ' ', num2str(idpar(6)), ']']
68            ['Time Step n = ', num2str(n)]});
69      ax = gca;
70      ax.FontSize = 12;
71
72      % Set axis limits for consistency
73      axis([1 nx 1 ny]);
74      % Set color axis limits to match data range
75      clim([min(psimod(:)) max(psimod(:))]);
76
77      % Scale the axes tick labels to range from 0 to 1
78      xticks(linspace(1, nx, 11));
79      yticks(linspace(1, ny, 11));
80      xticklabels(linspace(0, 1, 11));
81      yticklabels(linspace(0, 1, 11));
82
83      % Draw rectangle where the potential is
84      rectangle('Position', [rect_xmin, rect_ymin, rect_xlen, rect_ylen], ...
85                'EdgeColor', 'black', 'LineWidth', 1);
86
87      % Write to video file
88      frame = getframe(gcf);
89      writeVideo(video, frame);
90  end
91
92  % Close the video file and figure
93  close(video);
```

Appendix H - video_rec_well.m Code

```
%% 2.5 - 2d Video of Scattering off Rectangular Well

close all;
clear; clc;
format long;

% Simulation maximum time
tmax = 0.05;
% Discretization level
level = 8;
% Delta t by Delta x ratio
lambda = 0.05;

% idtype = 0   -> Exact family (sine wave)
% idtype = 1   -> Boosted Gaussian
idtype = 1;
%x0      = idpar(1);      y0 = idpar(2);
%delta_x = idpar(3); delta_y = idpar(4);
%p_x     = idpar(5);     p_y = idpar(6);
idpar = [0.5, 0.3, 0.1, 0.1, 0.0, 30];

% vtype = 0   -> No potential
% vtype = 1   -> Rectangular barrier or well
% vtype = 2   -> Double Slit
vtype = 1;
x_min = 0.2;   x_max = 0.8;
y_min = 0.7;   y_max = 0.8;
%Vc    = vpar(5);
vpar = [x_min, x_max, y_min, y_max, -1e8];

% Compute solution
[x y t psi psire psiim psimod v] = ...
    sch_2d_adi(tmax, level, lambda, idtype, idpar, vtype, vpar);

% Dimensions of matrix
[nt, nx, ny] = size(psimod);

% Create a VideoWriter object
video = VideoWriter('../../output/problem2/well_bar.avi');
video.FrameRate = 30;
open(video);

% Create a figure and define where the rectangle indicating where
% the potential barrier is will be drawn
figure;
rect_xmin = x_min * (nx - 1) + 1;
rect_ymin = y_min * (ny - 1) + 1;
rect_xlen = (x_max - x_min) * (nx - 1);
rect_ylen = (y_max - y_min) * (ny - 1);

% Loop over time steps
for n = 1:nt
    % reshape psi to create a 2d matrix at this timestep
    psi_n = reshape(psimod(n,:,:), nx, ny);

    % Create filled contour plot
```

```
57      contourf(psi_n, 20, 'LineStyle', 'none');
58      colormap("default");
59      xlabel('x');
60      ylabel('y');
61      title({'2d Schrödinger Equation Simulation'
62             '|ψ| Scattering off a Rectangular Well'
63             ['tmax = ', num2str(tmax), ', level = ', num2str(level), ...
64              ', lambda = ', num2str(lambda), ', idpar = [', ...
65              num2str(idpar(1)), ' ', num2str(idpar(2)), ' ', ...
66              num2str(idpar(3)), ' ', num2str(idpar(4)), ' ', ...
67              num2str(idpar(5)), ' ', num2str(idpar(6)), ']']
68            ['Time Step n = ', num2str(n)]});
69      ax = gca;
70      ax.FontSize = 12;
71
72      % Set axis limits for consistency
73      axis([1 nx 1 ny]);
74      % Set color axis limits to match data range
75      clim([min(psimod(:)) max(psimod(:))]);
76
77      % Scale the axes tick labels to range from 0 to 1
78      xticks(linspace(1, nx, 11));
79      yticks(linspace(1, ny, 11));
80      xticklabels(linspace(0, 1, 11));
81      yticklabels(linspace(0, 1, 11));
82
83      % Draw rectangle where the potential is
84      rectangle('Position', [rect_xmin, rect_ymin, rect_xlen, rect_ylen], ...
85                'EdgeColor', 'black', 'LineWidth', 1);
86
87      % Write to video file
88      frame = getframe(gcf);
89      writeVideo(video, frame);
90  end
91
92  % Close the video file and figure
93  close(video);
```

Appendix I - video_double_slit.m Code

```
%% 2.5 - 2d Video of Scattering through a Double Slit

close all;
clear; clc;
format long;

% Simulation maximum time
tmax = 0.05;
% Discretization level
level = 8;
% Delta t by Delta x ratio
lambda = 0.05;

% idtype = 0   -> Exact family (sine wave)
% idtype = 1   -> Boosted Gaussian
idtype = 1;
%x0      = idpar(1);      y0 = idpar(2);
%delta_x = idpar(3); delta_y = idpar(4);
%p_x     = idpar(5);     p_y = idpar(6);
idpar = [0.5, 0.0, 0.08, 0.08, 0.0, 60];

% vtype = 0   -> No potential
% vtype = 1   -> Rectangular barrier or well
% vtype = 2   -> Double Slit
vtype = 2;
x1 = 0.48;   x2 = 0.49;
x3 = 0.51;   x4 = 0.52;
%Vc    = vpar(5);
vpar = [x1, x2, x3, x4, 1e8];

% Compute solution
[x y t psi psire psiim psimod v] = ...
    sch_2d_adi(tmax, level, lambda, idtype, idpar, vtype, vpar);

% Dimensions of matrix
[nt, nx, ny] = size(psimod);

% Create a VideoWriter object
video = VideoWriter('../../output/problem2/double_slit.avi');
video.FrameRate = 30;
open(video);

% Create a figure
figure;

% Loop over time steps
for n = 1:nt
    % reshape ψ to create a 2d matrix at this timestep
    psi_n = reshape(psimod(n,:,:), nx, ny);

    % Create filled contour plot
    contourf(psi_n, 20, 'LineStyle', 'none');
    colormap("default");
    xlabel('x');
    ylabel('y');
    title({'2d Schrödinger Equation Simulation'
```

```
57              '|ψ| Scattering through Double Slits'
58              ['tmax = ', num2str(tmax), ', level = ', num2str(level), ...
59               ', lambda = ', num2str(lambda), ', idpar = [', ...
60               num2str(idpar(1)), ' ', num2str(idpar(2)), ' ', ...
61               num2str(idpar(3)), ' ', num2str(idpar(4)), ' ', ...
62               num2str(idpar(5)), ' ', num2str(idpar(6)), ']']
63             ['Time Step n = ', num2str(n)]});
64      ax = gca;
65      ax.FontSize = 12;
66
67      % Set axis limits for consistency
68      axis([1 nx 1 ny]);
69      % Set color axis limits to match data range
70      clim([min(psimod(:)) max(psimod(:))]);
71
72      % Scale the axes tick labels to range from 0 to 1
73      xticks(linspace(1, nx, 11));
74      yticks(linspace(1, ny, 11));
75      xticklabels(linspace(0, 1, 11));
76      yticklabels(linspace(0, 1, 11));
77
78      % Draw rectangles where the double slit potential is
79      hold on
80      rectangle('Position', [1, (ny - 1)/4 + 1, floor(x1*nx), 1], ...
81                'EdgeColor', 'black', 'LineWidth', 1);
82      rectangle('Position', [ceil(x2*nx), (ny - 1)/4 + 1, ceil((x3 - x2)*nx),
            1], ...
83                'EdgeColor', 'black', 'LineWidth', 1);
84      rectangle('Position', [ceil(x4*nx), (ny - 1)/4 + 1, floor((1 - x4)*nx),
            1], ...
85                'EdgeColor', 'black', 'LineWidth', 1);
86      hold off
87
88      % Write to video file
89      frame = getframe(gcf);
90      writeVideo(video, frame);
91  end
92
93  % Close the video file and figure
94  close(video);
```