

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

2d Schrödinger Equation

Numerical Approach

1d Schrödinger Equation

2d Schrödinger Equation

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

```

1 % sch_1d_cn: Solves 1D Schrödinger equation using  $O(dt^2, dx^2)$ 
2 % Crank–Nicolson implicit scheme.
3 %
4 % Inputs:
5 %
6 %   tmax:    Maximum integration time
7 %   level:   Discretization level
8 %   lambda:  dt/dx
9 %   idtype:  Selects initial data type
10 %   idpar:   Vector of initial data parameters
11 %   vtype:   Selects potential type
12 %   vpar:    Vector of potential parameters
13 %
14 % Outputs:
15 %
16 %   x:       Column vector of x coordinates [nx]
17 %   t:       Column vector of t coordinates [nt]
18 %   psi:     Array of computed psi values [nt x nx]
19 %   psire:   Array of computed psi_re values [nt x nx]
20 %   psiim:   Array of computed psi_im values [nt x nx]
21 %   psimod:  Array of computed sqrt(psi psi*) values [nt x nx]
22 %   prob:    Array of computed running integral values [nt x nx]
23 %   v:       Column vector of potential values [nx]
24 function [x t psi psire psiim psimod prob v] = ...
25     sch_1d_cn(tmax, level, lambda, idtype, idpar, vtype, vpar)
26
27 % Define mesh and derived parameters
28 nx = 2^level + 1;
29 x = linspace(0.0, 1.0, nx);
30 dx = x(2) - x(1);
31 dt = lambda * dx;
32 nt = round(tmax / dt) + 1;
33 t = (0 : nt-1) * dt;
34
35 % Initialize solution, and set initial data
36 psi = zeros(nt, nx);
37 if idtype == 0
38     % Exact family
39     psi(1, :) = sin(idpar(1) * pi * x);
40 elseif idtype == 1
41     % Boosted Gaussian
42     psi(1, :) = exp(1i * idpar(3) * x) .* ...
43         exp(-((x - idpar(1)) ./ idpar(2)).^ 2);
44 else
45     fprintf('sch_1d_cn: Invalid idtype=%d\n', idtype);
46     return
47 end
48 % Set first and last values of initial data to zero
49 psi(1, 1) = 0;
50 psi(1, nx) = 0;
51
52 % Initial storage for prob and calculate for initial time
53 prob = zeros(nt, nx);
54 for j = 2 : nx
55     prob(1, j) = trapz(x(1:j), abs(psi(1, 1:j)).^2);
56 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 %% 1.4 - 1d Convergence Testing
2
3 close all;
4 clear; clc;
5 format long;
6
7 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8 % Convergence Test #1
9 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10
11 % Simulation maximum time
12 tmax = 0.25;
13 % Discretization levels
14 minlevel = 6;
15 maxlevel = 9;
16 % Delta t by Delta x ratio
17 lambda = 0.1;
18
19 % idtype = 0    -> Exact family (sine wave)
20 % idtype = 1    -> Boosted Gaussian
21 idtype = 0;
22 idpar = [3]; % m = 3
23
24 % vtype = 0     -> No potential
25 % vtype = 1     -> Rectangular barrier or well
26 vtype = 0;
27 vpar = zeros(1,3);
28
29 % Perform computation at various levels of discretization, store
30 % results in cell arrays ...
31 for l = minlevel : maxlevel
32     % Compute the solution
33     [x{l} t{l} psi{l} psire{l} psiim{l} psimod{l} prob{l} v{l}] ...
34     = sch_1d_cn(tmax, l, lambda, idtype, idpar, vtype, vpar)
35
36     [nt{l}, nx{l}] = size(psi{l});
37
38     % Since idtype == 0, compute exact solution
39     psixct{l} = zeros(nt{l}, nx{l});
40     for n = 1 : nt{l}
41         psixct{l}(n,:) = exp(-1i * idpar(1)^2 * pi^2 * t{l}(n)) ...
42             * sin(idpar(1) * pi * x{l});
43     end
44     % Compute exact errors and their rms values for later
45     Epsi{l} = psixct{l} - psi{l};
46     rms_Epsi{l} = rms(abs(Epsi{l}), 2);
47 end
48
49 % Calculating the level-to-level differences, taking every second
50 % value of the larger length array
51 dpsi6 = downsample(downsample(psi{7}, 2).', 2).' - psi{6};
52 dpsi7 = downsample(downsample(psi{8}, 2).', 2).' - psi{7};
53 dpsi8 = downsample(downsample(psi{9}, 2).', 2).' - psi{8};
54
55 % Compute l-2 norm of each dpsi, resulting in functions of t
56 rms_dpsi6 = rms(abs(dpsi6), 2);

```

```

57 rms_dpsi7 = rms(abs(dpsi7), 2);
58 rms_dpsi8 = rms(abs(dpsi8), 2);
59
60 % Plot scaled errors for different discretization levels
61 fig1 = figure;
62 rho = 4;
63 hold on
64 plot(t{6}, rms_dpsi6, 'LineWidth', 2);
65 plot(t{7}, rho*rms_dpsi7, 'LineWidth', 2);
66 plot(t{8}, rho^2*rms_dpsi8, 'LineWidth', 2);
67 xlabel("Time");
68 ylabel("l-2 norm of difference between level");
69 legend('||dΨ^6||', '4 * ||dΨ^7||', '4^2 * ||dΨ^8||', 'Location', 'best');
70 title("1d Schrodinger equation convergence test - Exact family"
71       "l-2 norm of difference between level 1 solutions"
72       "idtype = 0, vtype = 0, tmax = 0.25, lambda = 0.1, 6 <= l <= 9");
73 ax = gca;
74 ax.FontSize = 12;
75
76 % Plot scaled exact errors for different discretization levels
77 fig2 = figure;
78 rho = 4;
79 hold on
80 plot(t{6}, rms_Epsi{6}, 'LineWidth', 2);
81 plot(t{7}, rho*rms_Epsi{7}, 'LineWidth', 2);
82 plot(t{8}, rho^2*rms_Epsi{8}, 'LineWidth', 2);
83 plot(t{9}, rho^3*rms_Epsi{9}, 'LineWidth', 2);
84 xlabel("Time");
85 ylabel("l-2 norm of exact error");
86 legend('||EΨ^6||', '4 * ||EΨ^7||', '4^2 * ||EΨ^8||', '4^3 * ||EΨ^9||', ...
87       , 'Location', 'best');
88 title("1d Schrodinger equation convergence test - Exact family"
89       "l-2 norm of exact error for each level l"
90       "idtype = 0, vtype = 0, tmax = 0.25, lambda = 0.1, 6 <= l <= 9");
91 ax = gca;
92 ax.FontSize = 12;
93
94
95 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
96 % Convergence Test #2
97 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
98
99 clear;
100
101 % Simulation maximum time
102 tmax = 0.01;
103 % Discretization levels
104 minlevel = 6;
105 maxlevel = 9;
106 % Delta t by Delta x ratio
107 lambda = 0.01;
108
109 % idtype = 0    -> Exact family (sine wave)
110 % idtype = 1    -> Boosted Gaussian
111 idtype = 1;
112 idpar = [0.50 0.075 0.0];
113
114 % vtype = 0     -> No potential

```

```

115 % vtype = 1    -> Rectangular barrier or well
116 vtype = 0;
117 vpar = zeros(1,3);
118
119 % Perform computation at various levels of discretization, store
120 % results in cell arrays ...
121 for l = minlevel : maxlevel
122     % Compute the solution
123     [x{1} t{1} psi{1} psire{1} psiim{1} psimod{1} prob{1} v{1}] ...
124     = sch_1d_cn(tmax, l, lambda, idtype, idpar, vtype, vpar);
125
126     [nt{1}, nx{1}] = size(psi{1});
127 end
128
129 % Calculating the level-to-level differences, taking every second
130 % value of the larger length array
131 dpsi6 = downsample(downsample(psi{7}, 2).', 2).' - psi{6};
132 dpsi7 = downsample(downsample(psi{8}, 2).', 2).' - psi{7};
133 dpsi8 = downsample(downsample(psi{9}, 2).', 2).' - psi{8};
134
135 % Compute l-2 norm of each dpsi, resulting in functions of t
136 rms_dpsi6 = rms(abs(dpsi6), 2);
137 rms_dpsi7 = rms(abs(dpsi7), 2);
138 rms_dpsi8 = rms(abs(dpsi8), 2);
139
140 % Plot scaled errors for different discretization levels
141 fig3 = figure;
142 rho = 4;
143 hold on
144 plot(t{6}, rms_dpsi6, 'LineWidth', 2);
145 plot(t{7}, rho*rms_dpsi7, 'LineWidth', 2);
146 plot(t{8}, rho^2*rms_dpsi8, 'LineWidth', 2);
147 xlabel("Time");
148 ylabel("l-2 norm of difference between level");
149 legend('||dΨ^6||', '4 * ||dΨ^7||', '4^2 * ||dΨ^8||', 'Location', 'best');
150 title({"1d Schrodinger equation convergence test - Boosted Gaussian"
151        "l-2 norm of difference between level l solutions"
152        "idtype = 1, vtype = 0, tmax = 0.01, lambda = 0.01, 6 <= l <= 9"});
153 ax = gca;
154 ax.FontSize = 12;

```

Appendix C - barrier_survey.m Code

```

1 %% 1.5.1 - 1d Barrier Survey
2
3 close all;
4 clear; clc;
5 format long;
6
7 % Simulation maximum time
8 tmax = 0.10;
9 % Discretization level
10 level = 9;
11 % Delta t by Delta x ratio
12 lambda = 0.01;
13
14 % idtype = 0    -> Exact family (sine wave)
15 % idtype = 1    -> Boosted Gaussian
16 idtype = 1;
17 idpar = [0.40, 0.075, 20.0];
18
19 % vtype = 0     -> No potential
20 % vtype = 1     -> Rectangular barrier or well
21 vtype = 1;
22 xmin = 0.6;
23 xmax = 0.8;
24 lnV0 = linspace(-2, 5, 251);
25
26 % Survey range
27 x1 = 0.8;
28 x2 = 1.0;
29
30 for idx = 1 : length(lnV0)
31     % Get this iteration's vpar
32     vpar = [xmin, xmax, exp(lnV0(idx))];
33
34     % Compute the solution
35     [x{idx} t{idx} psi{idx} psire{idx} psiim{idx} psimod{idx} prob{idx} v{
36         idx}] ...
37         = sch_1d_cn(tmax, level, lambda, idtype, idpar, vtype, vpar);
38
39     % Compute temporal average of probability matrix
40     P_bar{idx} = mean(prob{idx});
41     % Normalize the temporal average
42     P_bar{idx} = P_bar{idx} / P_bar{idx}(end);
43
44     % Compute indices of x1 and x2 in the array x
45     % This only needs to be done once
46     if idx == 1
47         [~, x1_loc] = min(abs(x{idx} - x1));
48         [~, x2_loc] = min(abs(x{idx} - x2));
49     end
50
51     % Compute excess fractional probability and its logarithm
52     Fe_bar{idx} = (P_bar{idx}(x2_loc) - P_bar{idx}(x1_loc)) / ...
53         (x{idx}(x2_loc) - x{idx}(x1_loc));
54     lnFe_bar{idx} = log(Fe_bar{idx})
55 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 )
63 ax = gca;
64 ax.FontSize = 12;

```

Appendix D - well_survey.m Code

```

1 %% 1.5.2 - 1d Well Survey
2
3 close all;
4 clear; clc;
5 format long;
6
7 % Simulation maximum time
8 tmax = 0.10;
9 % Discretization level
10 level = 9;
11 % Delta t by Delta x ratio
12 lambda = 0.01;
13
14 % idtype = 0  -> Exact family (sine wave)
15 % idtype = 1  -> Boosted Gaussian
16 idtype = 1;
17 idpar = [0.40, 0.075, 0.0];
18
19 % vtype = 0  -> No potential
20 % vtype = 1  -> Rectangular barrier or well
21 vtype = 1;
22 xmin = 0.6;
23 xmax = 0.8;
24 lnV0 = linspace(2, 10, 251);
25
26 % Survey range
27 x1 = 0.6;
28 x2 = 0.8;
29
30 for idx = 1 : length(lnV0)
31     % Get this iteration's vpar
32     vpar = [xmin, xmax, -exp(lnV0(idx))];
33
34     % Compute the solution
35     [x{idx} t{idx} psi{idx} psire{idx} psiim{idx} psimod{idx} prob{idx} v{
        idx}] ...
36     = sch_1d_cn(tmax, level, lambda, idtype, idpar, vtype, vpar);
37
38     % Compute temporal average of probability matrix
39     P_bar{idx} = mean(prob{idx});
40     % Normalize the temporal average
41     P_bar{idx} = P_bar{idx} / P_bar{idx}(end);
42
43     % Compute indices of x1 and x2 in the array x
44     % This only needs to be done once
45     if idx == 1
46         [~, x1_loc] = min(abs(x{idx} - x1));
47         [~, x2_loc] = min(abs(x{idx} - x2));
48     end
49
50     % Compute excess fractional probability and its logarithm
51     Fe_bar{idx} = (P_bar{idx}(x2_loc) - P_bar{idx}(x1_loc)) / ...
52                 (x{idx}(x2_loc) - x{idx}(x1_loc));
53     lnFe_bar{idx} = log(Fe_bar{idx})
54 end
55

```

```

56 fig2 = figure;
57 plot(lnV0, cell2mat(lnFe_bar), 'LineWidth', 2)
58 title({"Well Survey – Log–log plot of excess fractional probablity vs. V_0"
59       "Well between x = 0.6 and x = 0.8. x_1 = 0.6, x_2 = 0.8"})
60 xlabel('$$\mathbf{\ln(V_0)}$$', 'interpreter', 'latex')
61 ylabel('$$\mathbf{\ln(\overline{F_e}(x_1, x_2))}$$', 'interpreter', 'latex'
62        )
63 ax = gca;
64 ax.FontSize = 12;

```

Appendix E - sch_2d_adi.m Code

```

1 % sch_2d_adi: Solves 2D Schrödinger equation using ADI scheme.
2 %
3 % Inputs:
4 %
5 %   tmax:    Maximum integration time
6 %   level:   Discretization level
7 %   lambda:  dt/dx
8 %   idtype:  Selects initial data type
9 %   idpar:   Vector of initial data parameters
10 %  vtype:   Selects potential type
11 %  vpar:    Vector of potential parameters
12 %
13 % Outputs:
14 %
15 %   x:       Column vector of x coordinates [nx]
16 %   y:       Column vector of y coordinates [ny]
17 %   t:       Column vector of t coordinates [nt]
18 %   psi:     Array of computed psi values [nt x nx x ny]
19 %   psire:   Array of computed psi_re values [nt x nx x ny]
20 %   psiim:   Array of computed psi_im values [nt x nx x ny]
21 %   psimod:  Array of computed sqrt(psi psi*) values [nt x nx x ny]
22 %   v:       Array of potential values [nx x ny]
23 function [x y t psi psire psiim psimod v] = ...
24     sch_2d_adi(tmax, level, lambda, idtype, idpar, vtype, vpar)
25
26 % Define mesh and derived parameters
27 nx = 2^level + 1;          ny = nx;
28 x = linspace(0.0, 1.0, nx); y = x;
29 dx = x(2) - x(1);         dy = dx;
30 dt = lambda * dx;
31 nt = round(tmax / dt) + 1;
32 t = (0 : nt-1) * dt;
33
34 % Define meshgrid for populating psi(x,y,0) and V(x,y)
35 [X, Y] = meshgrid(x, y);
36
37 % Initialize solution, and set initial data
38 psi = zeros(nt, nx, ny);
39 if idtype == 0
40     % Exact family
41     psi(1, :, :) = sin(idpar(1)*pi*x)' * sin(idpar(2)*pi*y);
42 elseif idtype == 1
43     % Boosted Gaussian
44     % Create variable names for function parameters
45     x0 = idpar(1); y0 = idpar(2);
46     delta_x = idpar(3); delta_y = idpar(4);
47     p_x = idpar(5); p_y = idpar(6);
48
49     % Calculate psi(x, y, 0)
50     psi_0 = exp(1i*p_x*X) .* exp(1i*p_y*Y) ...
51         .* exp(-(((X - x0).^2)/delta_x^2 + ((Y - y0).^2)/delta_y^2));
52     psi(1, :, :) = reshape(psi_0, [1, nx, ny]);
53 else
54     fprintf('sch_2d_adi: Invalid idtype=%d\n', idtype);
55     return
56 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) = (1*dt/(2*dy^2)) * (psi_n(2:nx-1, j+1) + psi_n(2:nx
120 -1, j-1)) ...
121 + (1 - 1*dt*(1/dy^2 + v(2:nx-1,j)/2)) .* psi_n(2:nx
122 -1, j);
123 f(2:nx-1) = (1*dt/(2*dx^2)) * (f(1:nx-2) + f(3:nx)) + ...
124 (1 - 1*dt/dx^2) * f(2:nx-1);
125
126 % Impose boundary conditions
127 f(1) = 0.0;
128 f(nx) = 0.0;
129
130 % Solve first ADI system
131 psi_half(:,j) = A_half \ f;
132 % Impose boundary conditions
133 psi_half(1,:) = 0.0;
134 psi_half(:,1) = 0.0;
135 psi_half(nx,:) = 0.0;
136 psi_half(:,ny) = 0.0;
137 end
138
139 % Define upper and lower sparse matrix diagonals for second ADI eqn
140 dl = (-1*dt/(2*dy^2)) * ones(ny, 1);
141 du = dl;
142 % Impose boundary conditions
143 du(2) = 0.0;
144 dl(ny-1) = 0.0;
145
146 % Solve tridiagonal system for each i (column)
147 for i = 2:nx-1
148 % Define middle sparse matrix diagonal for second ADI eqn
149 v_i = reshape(v(i,:), ny, 1);
150 d = 1 + 1*dt/dy^2 + (1*dt/2)*v_i;
151 % Impose boundary conditions
152 d(1) = 1.0;
153 d(ny) = 1.0;
154
155 % Compute sparse matrix for second ADI eqn
156 A_full = spdiags([dl d du], -1:1, ny, ny);
157
158 % Compute RHS of second ADI eqn. BCs already imposed previously
159 f = reshape(psi_half(i,:), ny, 1);
160
161 % Solve second ADI system
162 psi(n+1, i, :) = A_full \ f;
163 % Impose boundary conditions
164 psi(n+1, 1, :) = 0.0;
165 psi(n+1, :, 1) = 0.0;
166 psi(n+1, nx, :) = 0.0;
167 psi(n+1, :, ny) = 0.0;
168 end
169 end
170
171 % Compute real, imaginary, and modulus of each entry in psi
172 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

```

1 %% 2.4 – 2d Convergence Testing
2
3 close all;
4 clear; clc;
5 format long;
6
7 % Simulation maximum time
8 tmax = 0.05;
9 % Discretization levels
10 minlevel = 6;
11 maxlevel = 9;
12 % Delta t by Delta x ratio
13 lambda = 0.05;
14
15 % idtype = 0    -> Exact family (sine wave)
16 % idtype = 1    -> Boosted Gaussian
17 idtype = 0;
18 idpar = [2, 3];
19 mx = idpar(1); my = idpar(2);
20
21 % vtype = 0     -> No potential
22 % vtype = 1     -> Rectangular barrier or well
23 vtype = 0;
24 vpar = zeros(1,5);
25
26 % Perform computation at various levels of discretization, store
27 % results in cell arrays ...
28 for l = minlevel : maxlevel
29     % Compute the solution
30     [x{l} y{l} t{l} psi{l} psire{l} psiim{l} psimod{l} v{l}] = ...
31         sch_2d_adi(tmax, l, lambda, idtype, idpar, vtype, vpar)
32
33     [nt{l}, nx{l}, ny{l}] = size(psi{l});
34
35     % Define meshgrid for computing exact psi(x,y,t)
36     [X{l}, Y{l}] = meshgrid(x{l}.' , y{l}.' );
37
38     % Compute exact solution
39     psixct{l} = zeros(nt{l}, nx{l}, ny{l});
40     for n = 1 : nt{l}
41         psixct_n{l} = exp(-1i*(mx^2 + my^2)*pi^2*t{l}(n)) * ...
42             sin(mx*pi*X{l}) .* sin(my*pi*Y{l});
43         % Enforce boundary conditions
44         psixct_n{l}(1, :) = 0.0;
45         psixct_n{l}(:, 1) = 0.0;
46         psixct_n{l}(nx{l}, :) = 0.0;
47         psixct_n{l}(:, ny{l}) = 0.0;
48         psixct{l}(n, :, :) = reshape(psixct_n{l}, [1, nx{l}, ny{l}]);
49     end
50
51     % Compute exact errors and their rms values for later
52     Epsi{l} = psixct{l} - psi{l};
53     Epsi_2d{l} = reshape(Epsi{l}, nt{l}, nx{l}*ny{l});
54     rms_Epsi{l} = rms(abs(Epsi_2d{l}), 2);
55
56     % Downsample each psi for differencing

```



```

57     psi_ds{1} = psi{1}(1:2:end, 1:2:end, 1:2:end);
58
59     % Flatten each 3d array into a 2d array
60     psi_2d{1} = reshape(psi{1}, nt{1}, nx{1}*ny{1});
61     psi_ds_2d{1} = reshape(psi_ds{1}, (nt{1}-1)/2+1, ...
62                             ((nx{1}-1)/2+1)*((ny{1}-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 1 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

```

1 %% 2.5 - 2d Video of Scattering off Rectangular Barrier
2
3 close all;
4 clear; clc;
5 format long;
6
7 % Simulation maximum time
8 tmax = 0.05;
9 % Discretization level
10 level = 8;
11 % Delta t by Delta x ratio
12 lambda = 0.05;
13
14 % idtype = 0  -> Exact family (sine wave)
15 % idtype = 1  -> Boosted Gaussian
16 idtype = 1;
17 %x0      = idpar(1);      y0 = idpar(2);
18 %delta_x = idpar(3); delta_y = idpar(4);
19 %p_x     = idpar(5);      p_y = idpar(6);
20 idpar = [0.5, 0.3, 0.1, 0.1, 0.0, 30];
21
22 % vtype = 0  -> No potential
23 % vtype = 1  -> Rectangular barrier or well
24 % vtype = 2  -> Double Slit
25 vtype = 1;
26 x_min = 0.2;   x_max = 0.8;
27 y_min = 0.7;   y_max = 0.8;
28 %Vc     = vpar(5);
29 vpar = [x_min, x_max, y_min, y_max, 1e8];
30
31 % Compute solution
32 [x y t psi psire psiim psimod v] = ...
33     sch_2d_adi(tmax, level, lambda, idtype, idpar, vtype, vpar);
34
35 % Dimensions of matrix
36 [nt, nx, ny] = size(psimod);
37
38 % Create a VideoWriter object
39 video = VideoWriter(' ../output/problem2/rec_bar.avi ');
40 video.FrameRate = 30;
41 open(video);
42
43 % Create a figure and define where the rectangle indicating where
44 % the potential barrier is will be drawn
45 figure;
46 rect_xmin = x_min * (nx - 1) + 1;
47 rect_ymin = y_min * (ny - 1) + 1;
48 rect_xlen = (x_max - x_min) * (nx - 1);
49 rect_ylen = (y_max - y_min) * (ny - 1);
50
51 % Loop over time steps
52 for n = 1:nt
53     % reshape  $\psi$  to create a 2d matrix at this timestep
54     psi_n = reshape(psimod(n, :, :), nx, ny);
55
56     % 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           '| $\psi$ | 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

```

1 %% 2.5 - 2d Video of Scattering off Rectangular Well
2
3 close all;
4 clear; clc;
5 format long;
6
7 % Simulation maximum time
8 tmax = 0.05;
9 % Discretization level
10 level = 8;
11 % Delta t by Delta x ratio
12 lambda = 0.05;
13
14 % idtype = 0  -> Exact family (sine wave)
15 % idtype = 1  -> Boosted Gaussian
16 idtype = 1;
17 %x0      = idpar(1);      y0 = idpar(2);
18 %delta_x = idpar(3); delta_y = idpar(4);
19 %p_x     = idpar(5);      p_y = idpar(6);
20 idpar = [0.5, 0.3, 0.1, 0.1, 0.0, 30];
21
22 % vtype = 0  -> No potential
23 % vtype = 1  -> Rectangular barrier or well
24 % vtype = 2  -> Double Slit
25 vtype = 1;
26 x_min = 0.2;   x_max = 0.8;
27 y_min = 0.7;   y_max = 0.8;
28 %Vc     = vpar(5);
29 vpar = [x_min, x_max, y_min, y_max, -1e8];
30
31 % Compute solution
32 [x y t psi psire psiim psimod v] = ...
33     sch_2d_adi(tmax, level, lambda, idtype, idpar, vtype, vpar);
34
35 % Dimensions of matrix
36 [nt, nx, ny] = size(psimod);
37
38 % Create a VideoWriter object
39 video = VideoWriter(' ../output/problem2/well_bar.avi ');
40 video.FrameRate = 30;
41 open(video);
42
43 % Create a figure and define where the rectangle indicating where
44 % the potential barrier is will be drawn
45 figure;
46 rect_xmin = x_min * (nx - 1) + 1;
47 rect_ymin = y_min * (ny - 1) + 1;
48 rect_xlen = (x_max - x_min) * (nx - 1);
49 rect_ylen = (y_max - y_min) * (ny - 1);
50
51 % Loop over time steps
52 for n = 1:nt
53     % reshape  $\psi$  to create a 2d matrix at this timestep
54     psi_n = reshape(psimod(n, :, :), nx, ny);
55
56     % 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           '| $\psi$ | 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

```

1 %% 2.5 – 2d Video of Scattering through a Double Slit
2
3 close all;
4 clear; clc;
5 format long;
6
7 % Simulation maximum time
8 tmax = 0.05;
9 % Discretization level
10 level = 8;
11 % Delta t by Delta x ratio
12 lambda = 0.05;
13
14 % idtype = 0  -> Exact family (sine wave)
15 % idtype = 1  -> Boosted Gaussian
16 idtype = 1;
17 %x0      = idpar(1);      y0 = idpar(2);
18 %delta_x = idpar(3); delta_y = idpar(4);
19 %p_x     = idpar(5);      p_y = idpar(6);
20 idpar = [0.5, 0.0, 0.08, 0.08, 0.0, 60];
21
22 % vtype = 0  -> No potential
23 % vtype = 1  -> Rectangular barrier or well
24 % vtype = 2  -> Double Slit
25 vtype = 2;
26 x1 = 0.48;   x2 = 0.49;
27 x3 = 0.51;   x4 = 0.52;
28 %Vc      = vpar(5);
29 vpar = [x1, x2, x3, x4, 1e8];
30
31 % Compute solution
32 [x y t psi psire psiim psimod v] = ...
33     sch_2d_adi(tmax, level, lambda, idtype, idpar, vtype, vpar);
34
35 % Dimensions of matrix
36 [nt, nx, ny] = size(psimod);
37
38 % Create a VideoWriter object
39 video = VideoWriter(' ../output/problem2/double_slit.avi ');
40 video.FrameRate = 30;
41 open(video);
42
43 % Create a figure
44 figure;
45
46 % Loop over time steps
47 for n = 1:nt
48     % reshape  $\psi$  to create a 2d matrix at this timestep
49     psi_n = reshape(psimod(n,:,:), nx, ny);
50
51     % Create filled contour plot
52     contourf(psi_n, 20, 'LineStyle', 'none');
53     colormap("default");
54     xlabel('x');
55     ylabel('y');
56     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),
83             1], ...
84             'EdgeColor', 'black', 'LineWidth', 1);
85     rectangle('Position', [ceil(x4*nx), (ny - 1)/4 + 1, floor((1 - x4)*nx),
86             1], ...
87             'EdgeColor', 'black', 'LineWidth', 1);
88     hold off
89
90     % Write to video file
91     frame = getframe(gcf);
92     writeVideo(video, frame);
93 end
94
95 % Close the video file and figure
96 close(video);

```