```
function [x,y, t, psi, psire, psiim, psimod, v] = sch_2d_adi(tmax,
 level, lambda, idtype, idpar, vtype, vpar)
% 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: Vector of x coordinates [nx]
% y: Vector of y coordinates [ny]
% t: 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]
   nx = 2^level + 1;
   x = linspace(0.0, 1.0, nx);
   dx = x(2) - x(1);

   ny = nx;
   y =  linspace(0.0, 1.0, ny);
   dy = y(2) - y(1);


   dt = lambda * dx;
   nt = round(tmax / dt) + 1;
   t = [0 : nt-1] * dt;

   % solution storage and boundary conditions
   psi = zeros(nx, ny, nt);

   % Set up parameters
   if idtype == 0
      m_x = idpar(1);
      m_y = idpar(2);

      psi(: , :, 1) = sin(m_x*pi*x)'*sin(m_y*pi*y);

   elseif idtype == 1
      x0 = idpar(1);
      y0 = idpar(2);

      deltax = idpar(3);
      deltay = idpar(4);
      px = idpar(5);
```

```
        py = idpar(6);

        ys = exp(-1i*py.*y).*exp(-((y - y0)./deltay).^2);
        xs = exp(-1i*px.*x).*(exp(-((x - x0)./deltax).^2));
        % Boundary conditions
        psi( : , :, 1) = ys.'*xs;
        psi(1,:, 1) = 0.0;
        psi(:,1,1) = 0.0;
        psi(nx, :, 1) = 0.0;
        psi(:,nx, 1) = 0.0;
%
    else
        fprintf('sch_1d_cn: Invalid idtype=%d\n', idtype);
        return
    end

    if vtype == 0
        V = zeros(nx, ny);
    elseif vtype == 1
        x_mn = vpar(1);  %get x_min
        index_x_low = round(x_mn/dx); % produce index of x_min

        x_mx = vpar(2); % get x_max
        index_x_high = round(x_mx/dx); %produce index of x_max

        y_mn = vpar(3);  %get y_min
        index_y_low = round(y_mn/dx); % produce index of y_min

        y_mx = vpar(4);  %get y_max
        index_y_high = round(y_mx/dx); % produce index of y_min

        potential = vpar(5);

        V = zeros(nx, ny);
        for xx = index_x_low: index_x_high
            for yy =  index_y_low:index_y_high
                  V(xx, yy) = potential; % assign potential
            end
        end

    elseif vtype == 2
        j = round((ny-1)/4) + 1;
        x1 = vpar(1);
        index_x1 = round(x1/dx);

        x2 =  vpar(2);
        index_x2 = round(x2/dx);

        x3 = vpar(3);
        index_x3 = round(x3/dx);

        x4 = vpar(4);
        index_x4 = round(x4/dx);
```

```
    Vc = vpar(5);

    V = zeros(nx, ny);
    V(:, j) = Vc;
    for xx = index_x1:index_x2
        V(xx, j) = 0;
    end
    for xx = index_x3:index_x4
        V(xx, j) = 0;
    end

else
   fprintf('sch_1d_cn: Invalid idtype=%d\n', idtype);
   return
end




% Set up first triagonal system

% 1. Triagonal system for psi^{n+1/2}

% Set up tridiagonal system ...
cplus = -1i*dt/(2*dx^2)* ones(nx, 1);
c0 = (1+1i*dt/dx^2)* ones(nx, 1);
cminus = cplus;

% Fix up boundary cases ...
c0(1) = 1.0;
cplus(2) = 0.0;
cminus(nx-1) = 0.0;
cplus(nx) = 1.0;

% Define sparse matrix ... use for x coordinates then y coordinates
H = spdiags([cminus c0 cplus], -1:1, nx, nx);
H(1, 1) = 1.0;
H(1, 2) = 0.0;
H(nx, nx-1) = 0.0;
H(nx,nx) = 1.0;



% iterate over time
for n = 1 : nt-1
    PSI = psi(:, :, n);

    psi_n = PSI; % Get matrix for timestep

    psi_res = zeros(nx, nx);
    for y_crd = 2:nx-1
```

```
            % step 1. calculate psi n minus half
            dyy = (psi_n( 2:nx-1, y_crd+1) -2*psi_n( 2:nx-1,
y_crd)+psi_n( 2:nx-1, y_crd-1))/dy^2;

            potential = V(2:nx-1, y_crd).*psi_n( 2:nx-1, y_crd+1);

            psi_minushalf = zeros(1,  nx);
            psi_minushalf(1, 2:nx-1) = psi_n(2:nx-1,
y_crd)+0.5*1i*dt*dyy -0.5*1i*dt.*potential;

            % step 2. calculate the RHS using psi n-0.5
            dxx = zeros(1, nx);
            dxx(1, 2:nx-1) = (psi_minushalf(1, 3:nx)
-2*psi_minushalf(1, 2:nx-1) + psi_minushalf(1, 1:nx-2))/dx^2;

            RHS = zeros(1, nx);

            RHS = psi_minushalf + 1i*dt/2*dxx;




            psi_half = H \ RHS.';

            % Step 3
            % Set up tridiagonal matrix for a +halfstep
            Potential =V(:, y_crd); % POTENTIAL


            aplus = -1i*dt/(2*dy^2)* ones(nx, 1);

            a0 = (1+1i*dt/dy^2)* ones(nx, 1)+1i*dt/2*Potential;
            aminus = aplus;

            % Fix up boundary cases ...
            a0(1) = 1.0;
            aplus(2) = 0.0;
            aminus(nx-1) = 0.0;
            aplus(nx) = 1.0;

            % Define sparse matrix ...
            A = spdiags([aminus a0 aplus], -1:1, nx, nx);
            A(1, 1) = 1.0;
            A(1, 2) = 0.0;
            A(nx, nx-1) = 0.0;
            A(nx,nx) = 1.0;

            psi_res(y_crd, :) = A \ psi_half;

        end
```

```
        psi(:, :, n+1) = psi_res;




    end
    v = V;
    psire= real(psi);
    psiim = imag(psi);
    psimod = abs(psi).^2;


end
```

```
Not enough input arguments.

Error in sch_2d_adi (line 22)
    nx = 2^level + 1;
```

Published with MATLAB® R2018a