Simulation using the Finite Difference method (FDM)

This script numerically solves the time-dependent Schrödinger equation: $i\hbar\frac{\partial\psi}{\partial t}(x,t) = \frac{-\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2} + V(x,t)\psi(x,t)$

using the Finite Difference method. In the entire scripts, the units of:

- positions are in nm
- times are in ps
- energies are in meV

The fundamental constants $\hbar, m_e, \ldots$ reflect this choice of units. The conversions are made numerically at initialization (see section "physical constants").

```
close all
clear all
```

Simulation parameters

The variables below determine the spatial and time ranges of the simulation.

```
f = 4e9          % Pumping frequency (in Hz)
```

```
f = 4.0000e+09
```

```
T = 1/f * 1e12;  % Pumping period (ps)

t_start = 000    % Start time of the simulation
```

```
t_start = 0
```

```
t_end   = T/2    % End time of the simulation
```

```
t_end = 125.0000
```

```

x_start = -60    % Position to start the computations from
```

```
x_start = -60
```

```
x_end   = 150    % Position at which to end the computations
```

```
x_end = 150
```

```

dx = 1           % Space precision in nm
```

```
dx = 1
```

```
C1 = 1/10        % Constant linking dt and dx
```

```
C1 = 0.1000
```

Then choose some options about what to plot. The less plots required, the less memory/time is used.

```
show_wave_packet = 0;               % Non-zero to show the wavepacket animation in
time
wave_packet_update_step = 6.1e3;    % How often to update the figure
wave_packet_animation_delay = 0.5;  % Delay between GIF frames

show_probability = 1;               % Non-zero to show the probability density in
time and space
show_energy = 0;
show_occupation_probs = 0;
energy_levels = 4;

save_workspace = 0;
save_figures = 1;
```

Simulation variables

These should not require modification.

Physical constants

We first start by defining the original values of the constants used.

```
me_SI = 9.10938291e-31;    % Electron mass
hbar_SI = 1.054571726e-34; % Reduced Planck's constant
e_SI = 1.602176565e-19;    % Elementary charge
c_SI = 2.99792458e8;
```

Now, we convert these into our preferred units: meV for energies, ps for time and nm for length.

```
meV = 1/e_SI*1e3;
nm = 1e9;
ps = 1e12;

c = c_SI * nm/ps;               % 3e5 nm/ps
hbar = hbar_SI * meV * ps;      % 0.6582 meV.ps
me = me_SI*c_SI^2 * meV / c^2;  % 0.0057 meV.ps^2/nm^2

meff = 0.19*me;                  % Electron effective mass
m = meff;                        % 0.0011 meV.ps^2/nm^2
```

Helper variables

```
Nx = round((x_end-x_start) / dx + 1)    % Number of grid points
```

```
Nx = 211
```

```
x = linspace(x_start, x_end, Nx);       % Position vector
dt = C1 * 2 * m * dx^2 / hbar           % Time resolution
```

```
dt = 3.2824e-04
```

```
Nt = round((t_end-t_start) / dt + 1)
```

```
Nt = 380816
```

```
C2 = dt/hbar;                               % Constant used in the differential equation

t = linspace(t_start, t_end, Nt);           % Number of time steps
sim_time = t_end-t_start                    % Total simulation time in ps
```

```
sim_time = 125.0000
```

```
time_range = (t_start + (0:Nt-1)*dt);   % Simulation time range in ps

qd_min = zeros(Nt, 1);
```

File settings

```
suffix = sprintf("fdm-dx-%0.2f-%dGHz-time-%0.1f-%0.1fps/", dx, f/1e9, t_start,
t_end);
path = "data/" + suffix;                    % Output folder
mkdir(path); % Create folder for the experiment
```

```
Warning: Directory already exists.
```

```

wave_packet_file = sprintf(path + "wave-packet.gif");
prob_density_file = sprintf(path + "probability-density.fig");
occupation_prob_file = sprintf(path + "occupation-prob.fig");
energy_file = sprintf(path + "energy.fig");
```

Potential profile

```
% Alpha coefficients from supp. material section II (see table at the end)
alpha_ent_barr = 0.49;
alpha_ent_exit_barr = 0.037;
alpha_exit_barr = 0.48;
alpha_exit_ent_bar = 0.052;

V_ent = -0.7;   % Entrance gate voltage. From supp. material section III
V_exit = -0.7;  % Exit gate voltage. From supp. material section III
V_ac = 1.415;   % Entrance gate AC amplitude voltage. From supp. material section
III

x_ent = 0;      % Position of the entrance gate. From supp. material section III
x_exit = 100;   % Position of the exit gate. From supp. material section III

U_scr = 1;      % From supp. material section III
L_ent = 100;    % Entrace gate length. Set as the same as the exit gate
L_exit = 100;   % Exit gate length. From Fig.2 of main paper. Modified for sim
L_scr = 1;      % From supp. material section III

U_ent_x = -alpha_ent_barr * (alpha_ent_barr / alpha_ent_exit_barr) ...
          .^ ( -abs(x-x_ent) / abs(x_exit-x_ent) );
U_exit = -alpha_exit_barr * V_exit ...
```

```
         * (alpha_exit_barr / alpha_exit_ent_bar) ...
         .^ ( -abs(x-x_exit) / abs(x_ent-x_exit) );

U_upper = U_scr .* exp(-(x-x_ent)/L_scr .* (x-x_ent>0) ) ...
          .* exp(-(x_ent-L_ent-x)/L_scr .* (x_ent-L_ent-x>0)) ...
          + U_scr .* exp(-(x-x_exit-L_exit)/L_scr .* (x-x_exit-L_exit>0)) ...
          .* exp(-(x_exit-x)/L_scr .* (x_exit-x>0));


U_ent = (V_ent + V_ac * cos(2*pi*f*t_start*1e-12)) * U_ent_x;
U0 = U_ent + U_exit + U_upper;
```

The following are the parameters for the parabolic approximation $V(x,t)$.

```
DeltaE = 1;               % Level spacing of the QD (meV)
omega = DeltaE / hbar;    % ps^-1
```

Once the parameters of V are known, we can compute the initial value of this potential.

```
[~, qd_min0] = min(U0);
x0 = x_start + qd_min0 * dx;
V0 = 1/2 * m * omega^2 * (x - x0).^2;
qd_min(1) = x0;
```

Wave packet initialization

We initialize the wavefunction with a wavepacket corresponding to the ground state of the harmonic oscillator centered in x_0:

$$\psi(x, t=0) = \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} e^{-\frac{1m\omega}{2\hbar}(x-x_0)^2}$$

The intial energy of that packet is then $E_0 = \frac{\hbar\omega}{2}$ and the initial position x_0 is taken as the minimum of the potential.

The `prob` variable will contain $|\psi(t,x)|^2$ for all t and x. It is computed from y_R and y_I.

```
yR = harmonic_oscillator_state(0, hbar, m, omega, x0, x_start, x_end, Nx);
yI = zeros(1, Nx);


%% TODO: single at init of after ? Update CNM as well
prob = zeros(Nt, Nx, "single");
prob(1, :) = (yR.^2 + yI.^2) * dx;
```

We now define two arrays: `psi_n` and `amp_psi_n`. The first contains the eigenstates of the harmonic oscillators. The second contains the amplitude of each of these states. The amplitude is computed using a scalar product.

```
if show_occupation_probs
```

```
    phi_n = zeros(energy_levels, Nx);
    amp_phi_n = zeros(Nt, energy_levels);

    for n = 1 : energy_levels
        phi_n(n, :) = harmonic_oscillator_state(n-1, hbar, m, omega, x0, x_start,
x_end, Nx);
        amp_phi_n(1, n) = trapz((yR-1i*yI) .* phi_n(n,:) * dx);
    end
end
```

We define below arrays to keep track of the different energies with time.

```
if show_energy
    energy = zeros(Nt, 1, "single");
    kinetic_energy = zeros(Nt, 1, "single");
    potential_energy = zeros(Nt, 1, "single");
    [energy(1), kinetic_energy(1), potential_energy(1)] =
wave_packet_energy((yR+1i*yI)', V0, dx, m, hbar);
end
```

Main loop

Shouldn't require modification either. The progress is displayed by chunks of 10%.

```
cur_frame = 0; % Frame counter when plotting the wave packet
progress = 0
```

```
progress = 0
```

```
tic
for nt = 2 : Nt
    if nt >= (10+progress)*Nt/100
        progress = round(nt/Nt * 100)
    end

    % Time-dependent potential U_ent(x,t)
    U_ent = (V_ent + V_ac * cos(2*pi*f*(t_start+(nt-1)*dt)*1e-12)) * U_ent_x;

    % Total potential U(x,t)
    U = U_ent + U_exit + U_upper;
    [~, qd_min_new] = min(U);
    qd_min(nt) = x_start + qd_min_new * dx;
    V = 1/2 * m * omega^2 * (x - qd_min(nt)).^2;

    % Compute the wavefunction: Ian Cooper's method vectorized
    yR(2:Nx-1) = yR(2:Nx-1) - C1 * (yI(3:Nx) - 2*yI(2:Nx-1) + yI(1:Nx-2)) +
C2*V(2:Nx-1).*yI(2:Nx-1);
    yI(2:Nx-1) = yI(2:Nx-1) + C1 * (yR(3:Nx) - 2*yR(2:Nx-1) + yR(1:Nx-2)) -
C2*V(2:Nx-1).*yR(2:Nx-1);
```

```
    % Energy
    if show_energy
        [energy(nt), kinetic_energy(nt), potential_energy(nt)] =
wave_packet_energy((yR+1i*yI)', V, dx, m, hbar);
    end

    if show_occupation_probs
        for n = 1 : energy_levels
            amp_phi_n(nt, n) = trapz((yR-1i*yI) .* phi_n(n,:) * dx);
        end
    end

    % Normalize and save the probability
    prob(nt, :) = (yR.^2 + yI.^2) * dx;

    % Stop simulation if result is unphysical
    if max(prob(nt, :)) > 1e10
        fprintf("Divergence detected. Stopping...")
        nt                                          % Output when things went wrong
        break
    end


    % If first frame or a frame that we should show...
    if show_wave_packet && ( (mod(nt,wave_packet_update_step) == 0) || (nt == 2) )
        cur_frame = plot_wave_packet(x, nt, dt, t_start, prob(nt,:), V, ...
                                     wave_packet_animation_delay, wave_packet_file,
cur_frame);
    end
end
```

```
progress = 10
progress = 20
progress = 30
progress = 40
progress = 50
progress = 60
progress = 70
progress = 80
progress = 90
progress = 100
```

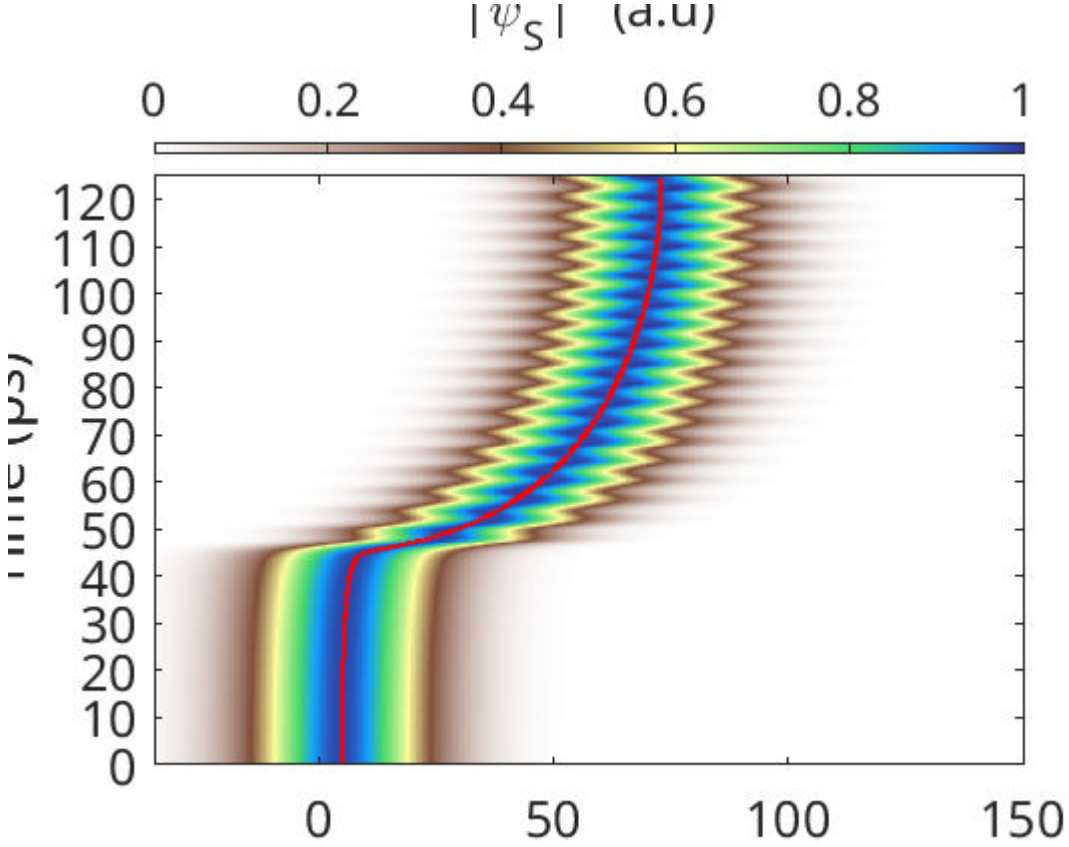
```
toc
```

```
Elapsed time is 24.950694 seconds.
```

Results

```
if save_workspace
    save(path + "workspace.mat", "-v7.3");
end
```

```
% Probability density. Plot to compare to Fig.1 of the main paper.
if show_probability
    prob_fig = figure('Name', 'Probability density evolution in time and space');
    im = imagesc(x, time_range, prob/max(prob(:)));

    hold on;
    color_white = [1, 1, 1];
    color_red = [1, 0, 0];
    plot(qd_min, time_range, 'LineWidth', 2, 'Color', color_red);

    load('CustomColormap','custommap')
    colormap(custommap)

    axis xy;

    c = colorbar('Limits',[0, 1], 'Location', 'northoutside');
    c.Label.String = '|\psi_S|^2 (a.u)';
    c.Ticks = 0:0.2:1.0;
    ax = gca;
    axpos = ax.Position;
    c.Position(4) = 0.25*c.Position(4);
    ax.Position = axpos;

    xticks([0, 50, 100, 150]);
    xlabel('Position (nm)');
    yticks(0:10:120);
    ylabel('Time (ps)');

    % Increase font size
    fontsize = 17;
    c.Label.FontSize = fontsize;
    a = get(gca,'XTickLabel');
    set(gca,'XTickLabel',a,'fontsize',fontsize)
    a = get(gca,'YTickLabel');
    set(gca,'YTickLabel',a,'fontsize',fontsize)

    xlim([-35, 150]);

    if save_figures
%         savefig(prob_fig, prob_density_file);
    end
end
```




```
if show_occupation_probs
    occupation_prob_fig = figure('Name', 'Occupation probabilities evolution in
time');
    hold on;

    for n = 1 : energy_levels
        legend_str = "State of energy E_" + num2str(n-1);
        plot(time_range, abs(amp_phi_n(:, n)).^2, 'DisplayName', [legend_str]);
    end

    xlim([time_range(1), time_range(end)]);
    xlabel('Time (ps)');
    ylabel('State probability');
    legend("show");
    title("Occupation probabilities of states");

    if save_figures
        savefig(occupation_prob_fig, occupation_prob_file);
    end
end
```

```
if show_energy
    % Compute the first two energy levels
    E0 = hbar*omega/2;
```

```
    E1 = E0 + DeltaE;

    % Plot them
    energy_fig = figure('Name', 'Energy evolution in time');
    hold on;

    plot(time_range, ones(1, Nt) * E0, 'LineStyle', '--')
    plot(time_range, ones(1, Nt) * E1, 'LineStyle', '--')
    plot(time_range, energy);
    plot(time_range, kinetic_energy)
    plot(time_range, potential_energy)

    xlim([time_range(1), time_range(end)])
    xlabel('Time (ps)');
    ylabel('Energy (meV)');
    yticks([0.25, 0.5, 1.5]);
    title('Energy of the electron wave packet');
    legend("E_0", "E_1", "<E>(t)", "<Ec>(t)", "<V>(t)");

    if save_figures
        savefig(energy_fig, energy_file);
    end
end
```

Scratchpad

```
```

Acknowledgments

Original file designed by Ian Cooper (School of Physics, University of Sydney), 12 dec 2015. Title: Solving the Time Dependent Schrodinger Equation using the FDTD Method.

Documentation: www.physics.usyd.edu.au/teach_res/mp/mphome.htm

Mscripts: www.physics.usyd.edu.au/teach_res/mp/mscripts

Modified 20 February 2020 by Alan Gardin and Giuseppe Tettamanzi, University of Adelaide