```
%DEFINITION OF THE REAL SPACE GRIDS
%Grid spacing values
delta_x=0.1;

%Lower and upper values of the x_grid.
%The total length of the x_grid is 2*(-lower_x)/delta_x = N, which must be a multiple↙
of 4.
%This will be important when doing Fourier Transforms!
lower_x=-10;
upper_x=-lower_x - delta_x;

%Definition_of the x_grid
x_grid=lower_x:delta_x:upper_x;
x_grid_size=size(x_grid);
x_grid_size=x_grid_size(2);

%-------------------------------------------------------------------------

%DEFINITION OF THE K SPACE GRIDS
%k-space grid spacing. Needs to fullfill the Discrete Fourier Transform
%value.
delta_k_x=2*pi/(x_grid_size*delta_x);

%Respective reciprocal grids.
kx_grid=-pi/delta_x:delta_k_x:pi/delta_x-delta_k_x;


%-------------------------------------------------------------------------

%Time step
%This might be much smaller than necessary. See below in "steps_collect".
delta_t=0.05;

%-------------------------------------------------------------------------
%Time independent Schrodinger equation step

%These are the normalization of the harmonic oscillator wavefunctions for
%each of the oscillators.
%mass=1
omega=1;
quanta_1=0;
Normalization=((omega/pi)^(1/4))/sqrt((2^quanta_1)*factorial(quanta_1));

%At t=0 start at one of the eigenstates of the initial Hamiltonian with
%V=V_0

%Starting potential
%V=(omega_1)^2*(x_grid).^2/2;

V_0=-0.2*x_grid.^2 + x_grid.^4/60 -x_grid;
```

```
V_f=-0.2*x_grid.^2 + x_grid.^4/60 +x_grid;

% x0=3;
% %delta_E=0;
% delta_E=3;
% V1_0=(x_grid-x0).^2/2-delta_E;
% V2_0=(x_grid+x0).^2/2;
% J=0.01;
% V_0=(V1_0+V2_0)/2 - sqrt(((V1_0-V2_0)/2).^2+J^2);
%
% V1_f=(x_grid-x0).^2/2;
% V2_f=(x_grid+x0).^2/2-delta_E;
% V_f=(V1_f+V2_f)/2 - sqrt(((V1_f-V2_f)/2).^2+J^2);

H=zeros(x_grid_size,x_grid_size);
t=-1/(2*delta_x^2);

for n=1:1:x_grid_size-1
    H(n,n+1)=t;
    H(n+1,n)=t;
end

for n=1:1:x_grid_size
    H(n,n)=V_0(n)-2*t;
end

[eigvecH,eigvalH]=eig(H);

initial_wavepacket=(eigvecH(:,1)).';

%-------------------------------------------------------------------------
%Time dependent Schrodinger equation step



%Kinetic energy operator
K=kx_grid.^2/2;


%Kinetic energy propagator for a single time step delta_t;
expK=exp(-1i*K*delta_t);


%Bulb function for Fourier transform. See my notes. The bulb acts in 2D in
%this case.
bulb=zeros(1,x_grid_size);
for r=1:1:x_grid_size
       bulb(1,r)=(-1)^(r);
end
```

```
%Time of evolution

%For adiabatic, make final_time=3000; for nonadiabatic make final_time~300
final_time=3000; %000;
total_steps=floor(final_time/delta_t);
%Maximum frequency you want to resolve (so that your spectrum will go from
%-max_freq to +max_freq.
max_freq=20;

%Collect at every steps_collect number of time_steps

%According to Fourier theory, if you want to collect spectra in the
%mentioned range, your time step of collection must be ~pi/max_freq,
%which in turn gives
%steps_collect=floor(pi/(max_freq*delta_t));
steps_collect=floor(total_steps/100);
%as the number of "smaller" steps (in our case, about 3) defined by delta_t you want↙
to skip before collecting. If you
%collect more often, then you will get a broader spectral domain, which may
%make visualization and data storage difficult. Alternatively, it is
%possible that delta_t has been chosen to be too small and you might be able to make↙
the propagation
%faster and collect data from every single time step you evolve.

%Signal matrix will be a N x 2 matrix where N is the number of time points
%where you will collect data. The first column will be the time and the
%second will be the value of the autocorrelation function of the dipole.


%The counter registers the number of time points where we collected data
counter=0;

psi_t=initial_wavepacket;

for step=0:1:total_steps

  time=step*delta_t;
  fraction_trajectory=(step/total_steps);

  coeff=2*(step)/total_steps;
  V=(1-fraction_trajectory)*V_0+fraction_trajectory*V_f;
  %V=V_0+2*x_grid;



                %Tfem=time*(1e5/(3*2*pi))/100;
                expV=exp(-1i*V*delta_t/2);
```

```
                psi_t=expV.*psi_t;
                psi_t=bulb.*fft(bulb.*psi_t);
                psi_t=expK.*psi_t;
                psi_t=bulb.*ifft(bulb.*psi_t);
                psi_t=expV.*psi_t;

                if mod(step,steps_collect)==0
                    for n=1:1:x_grid_size
                           H(n,n)=V(n)-2*t;
                    end

                [eigvecH,eigvalH]=eig(H);


                figure(1);
                subplot(2,1,1);
                plot(x_grid,3*abs(eigvecH(:,1)),'Color','r','Linewidth',3);
                hold on;
                plot(x_grid,3*abs(psi_t),'Color','b','Linewidth',3);
                hold on;

                plot(x_grid,0.1*V,'Color','g','Linewidth',3);
                xlabel('Position x');
                ylabel('Energy');
                axis([-10 20 -1 1]);
                legend('Adiabatic eigenstate','Actual TD state','TD Potential');
                set(gca,'Fontsize',25);
                hold off;


                subplot(2,1,2);
                plot(time,eigvalH(1,1),'+','Color','r','Linewidth',3);
                hold on;
                plot(time,eigvalH(2,2),'+','Color','b','Linewidth',3);
                xlabel('Time');
                ylabel('Adiabatic eigenenergies','Fontsize',20);
                hold on;

                figure(1);
                set(gcf,'color','w');
                set(gca,'Fontsize',25);

                pause;
                end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%



end
```