PHYS 516: Methods of Computational Physics
ASSIGNMENT 5- Quantum Dynamics Basics

Anup V Kanale
March 8, 2017

1 Exponentiation of the Kinetic Energy operator

From the lecture notes, the Kinetic energy operator is given as

$$T = \begin{bmatrix} 2a & b & & & & & b \\ b & 2a & b & & & & \\ & b & 2a & b & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & b & 2a & b & \\ & & & & b & 2a & b \\ b & & & & & b & 2a \end{bmatrix} \tag{1}$$

$$= \frac{1}{2}\begin{bmatrix} a & b & & & & & \\ b & a & & & & & \\ & & a & b & & & \\ & & b & a & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & & a & b \\ & & & & & b & a \end{bmatrix} + \begin{bmatrix} a & & & & & & b \\ & a & b & & & & \\ & b & a & & & & \\ & & & a & b & & \\ & & & b & a & & \\ & & & & \ddots & \ddots & \ddots \\ & & & & & a & b \\ b & & & & & b & a \end{bmatrix} + \frac{1}{2}\begin{bmatrix} a & b & & & & & \\ b & a & & & & & \\ & & a & b & & & \\ & & b & a & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & & a & b \\ & & & & & b & a \end{bmatrix} \tag{2}$$

We can exponentiate this kinetic energy operator using the Trotter expansion as follows

$$e^{-iT\Delta t} = U_X^{half} U_x^{full} U_X^{half} + O(\Delta t^3) \tag{3}$$

Since T is a sum of block diagonal matrices, each block may be exponentiated individually. Consider the diagonal block $A = \begin{bmatrix} a & b \\ b & a \end{bmatrix}$. From HW1, we know that A can be diagonalized and expressed as

$$A = \begin{bmatrix} a & b \\ b & a \end{bmatrix} = 0.5 \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a+b & 0 \\ 0 & a-b \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{4}$$

Using telescoping,

$$A^n = 0.5 \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} (a+b)^n & 0 \\ 0 & (a-b)^n \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{5}$$

$$= 0.5 \begin{bmatrix} (a+b)^n + (a-b)^n & (a+b)^n - (a-b)^n \\ (a+b)^n - (a-b)^n & (a+b)^n + (a-b)^n \end{bmatrix} \tag{6}$$

Using the definition of exponential, we can write

$$e^{iA\Delta t} = \sum_n \frac{i\Delta t}{n!} A^n \tag{7}$$

$$= \sum_n \frac{i\Delta t}{n!} \frac{1}{2} \begin{bmatrix} (a+b)^n + (a-b)^n & (a+b)^n - (a-b)^n \\ (a+b)^n - (a-b)^n & (a+b)^n + (a-b)^n \end{bmatrix} \tag{8}$$

$$= \frac{1}{2} \begin{bmatrix} \sum_n \frac{i\Delta t}{n!}(a+b)^n + \sum_n \frac{i\Delta t}{n!}(a-b)^n & \sum_n \frac{i\Delta t}{n!}(a+b)^n - \sum_n \frac{i\Delta t}{n!}(a-b)^n \\ \sum_n \frac{i\Delta t}{n!}(a+b)^n - \sum_n \frac{i\Delta t}{n!}(a-b)^n & \sum_n \frac{i\Delta t}{n!}(a+b)^n + \sum_n \frac{i\Delta t}{n!}(a-b)^n \end{bmatrix} \tag{9}$$

$$= \frac{1}{2} \begin{bmatrix} e^{i\Delta t(a+b)} + e^{i\Delta t(a-b)} & e^{i\Delta t(a+b)} - e^{i\Delta t(a-b)} \\ e^{i\Delta t(a+b)} - e^{i\Delta t(a-b)} & e^{i\Delta t(a+b)+e^{i\Delta t(a-b)}} \end{bmatrix} \tag{10}$$

The time step Δt can be modified for dull or half step. Let the general time step be $\frac{\Delta t}{n}, n = 1, 2$. Now, using

$$\epsilon_n^+ = \frac{1}{2}[e^{\frac{i\Delta t}{n}(a+b)} + e^{\frac{i\Delta t}{n}(a-b)}] \tag{11}$$

$$\epsilon_n^- = \frac{1}{2}[e^{\frac{i\Delta t}{n}(a+b)} - e^{\frac{i\Delta t}{n}(a-b)}] \tag{12}$$

where n denotes full step or half step, in eqn (3), it can be written as

$$\begin{bmatrix} \epsilon_2^+ & \epsilon_2^- & & & & & & \\ \epsilon_2^- & \epsilon_2^+ & & & & & & \\ & & \epsilon_2^+ & \epsilon_2^- & & & & \\ & & \epsilon_2^- & \epsilon_2^+ & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & & & & \epsilon_2^+ & \epsilon_2^- \\ & & & & & & \epsilon_2^- & \epsilon_2^+ \end{bmatrix} \begin{bmatrix} \epsilon_1^+ & & & & & & & \epsilon_1^- \\ & \epsilon_1^+ & \epsilon_1^- & & & & & \\ & \epsilon_1^- & \epsilon_1^+ & & & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & \epsilon_1^+ & \epsilon_1^- & & \\ & & & & \epsilon_1^- & \epsilon_1^+ & & \\ \epsilon_1^- & & & & & & \epsilon_1^+ & \end{bmatrix} \begin{bmatrix} \epsilon_2^+ & \epsilon_2^- & & & & & & \\ \epsilon_2^- & \epsilon_2^+ & & & & & & \\ & & \epsilon_2^+ & \epsilon_2^- & & & & \\ & & \epsilon_2^- & \epsilon_2^+ & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & & & & \epsilon_2^+ & \epsilon_2^- \\ & & & & & & \epsilon_2^- & \epsilon_2^+ \end{bmatrix} \tag{13}$$

which is the required exponentiation of the Kinetic Energy operator as in space splitting method.

2 Quantum Dynamics using Spectral Method

In this problem, the dynamics were simulated using the Spectral method. The total energy of the system is conserved, as shown in the figure below.

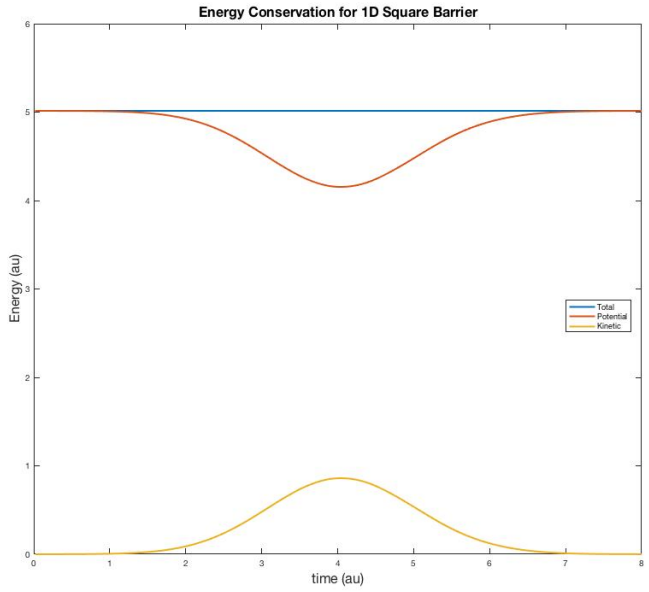

Figure 1:

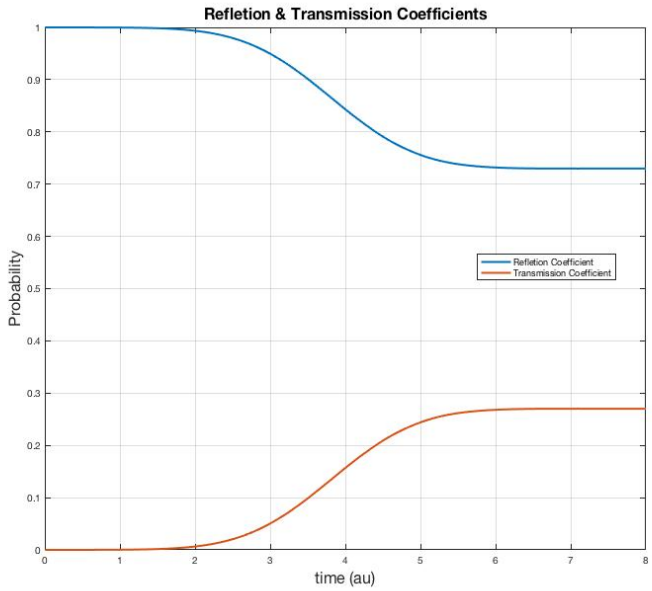

Figure 2:

The reflection and transmission coefficients were plotted as shown in figure 2.

The source code is as shown below.

```
/*************************************************************
Quantum dynamics (QD) simulation of an electron in one dimension.

USAGE
%cc -o qd1 qd1.c -lm
%qd1 < qd1.in (see qd1.h for the input-file format)
*************************************************************/
#include <stdio.h>
#include <math.h>
#include "qd1Spectral.h"

int main(int argc, char **argv) {
  int step; /* Simulation loop iteration index */
  int i,m;

  init_param();  /* Read input parameters */
  init_prop();   /* Initialize the kinetic & potential propagators */
  init_wavefn(); /* Initialize the electron wave function */

  double refl[Nstep];
  double tran[Nstep];

  for (step=1; step<=Nstep; step++) {
    single_step(); /* Time propagation for one step, DT */

    refl[step] = 0;    tran[step] = 0;
    for (i=0; i<NX/2; i++)
      refl[step] += (psi[2*i+0]*psi[2*i+0] + psi[2*i+1]*psi[2*i+1])*dx;
    for (i=NX/2; i<NX; i++)
      tran[step] += (psi[2*i+0]*psi[2*i+0] + psi[2*i+1]*psi[2*i+1])*dx;
    // printf("%le %le %le\n", DT*step, refl[step], tran[step]);

    if (step%NECAL==0) {
      calc_energy();
      printf("%le %le %le %le\n",DT*step,ekin,epot,etot);
    }
  }
  return 0;
}

void init_param() {
/*-------------------------------------------------------------
  Initializes parameters by reading them from standard input.
-------------------------------------------------------------*/
  /* Read control parameters */
  scanf("%le",&LX);
```

```
47    scanf("%le",&DT);
48    scanf("%d",&Nstep);
49    scanf("%d",&NECAL);
50    scanf("%le%le%le",&X0,&S0,&E0);
51    scanf("%le%le",&BH,&BW);
52    scanf("%le",&EH);
53
54    /* Calculate the mesh size */
55    dx = LX/NX;
56  }
57
58  void init_prop() {
59  /*-------------------------------------------------------------
60    Initializes the kinetic & potential propagators.
61  -------------------------------------------------------------*/
62    int stp,s,i,up,lw,m;
63    double x;
64
65    /* Set up kinetic propagators */
66    for (m=0; m<NX; m++){
67      if (m<=NX/2)
68        km[m] = 2*M_PI*m/LX;
69      else
70        km[m] = 2*M_PI*(m-NX)/LX;
71      ut[2*m+0] = cos(-0.5*DT*km[m]*km[m]);
72      ut[2*m+1] = sin(-0.5*DT*km[m]*km[m]);
73    }
74
75    /* Set up potential propagator */
76    for (i=0; i<NX; i++) {
77      x = dx*i;
78      /* Construct the edge potential */
79      if (i==0 || i==NX-1)
80        v[i] = EH;
81      /* Construct the barrier potential */
82      else if (0.5*(LX-BW)<x && x<0.5*(LX+BW))
83        v[i] = BH;
84      else
85        v[i] = 0.0;
86      /* Half-step potential propagator */
87      uv[2*i+0] = cos(-0.5*DT*v[i]);
88      uv[2*i+1] = sin(-0.5*DT*v[i]);
89    }
90  }
91
92  /*-------------------------------------------------------------*/
93  void init_wavefn() {
94  /*-------------------------------------------------------------
```

```
95    Initializes the wave function as a traveling Gaussian wave packet.
96  ------------------------------------------------------------------*/
97    int sx,s;
98    double x,gauss,psisq,norm_fac;
99
100   /* Calculate the the wave function value mesh point-by-point */
101   for (sx=0; sx<NX; sx++) {
102     x = dx*sx-X0;
103     gauss = exp(-0.25*x*x/(S0*S0));
104     psi[2*sx+0] = gauss*cos(sqrt(2.0*E0)*x);
105     psi[2*sx+1] = gauss*sin(sqrt(2.0*E0)*x);
106   }
107
108   /* Normalize the wave function */
109   psisq=0.0;
110   for (sx=0; sx<NX; sx++)
111     for (s=0; s<2; s++)
112       psisq += psi[2*sx+s]*psi[2*sx+s];
113   psisq *= dx;
114   norm_fac = 1.0/sqrt(psisq);
115   for (sx=0; sx<NX; sx++)
116     for (s=0; s<2; s++)
117       psi[2*sx+s] *= norm_fac;
118 }
119
120 /*------------------------------------------------------------------*/
121 void prop( double u[]) {
122   int sx;
123   double wr,wi;
124
125   for (sx=0; sx<NX; sx++) {
126     wr=u[2*sx+0]*psi[2*sx+0]-u[2*sx+1]*psi[2*sx+1];
127     wi=u[2*sx+0]*psi[2*sx+1]+u[2*sx+1]*psi[2*sx+0];
128     psi[2*sx+0]=wr;
129     psi[2*sx+1]=wi;
130   }
131 }
132
133 /*------------------------------------------------------------------*/
134 void single_step() {
135 /*------------------------------------------------------------------
136   Propagates the electron wave function for a unit time step, DT.
137 ------------------------------------------------------------------*/
138   int j;
139
140   prop(uv);  /* half step potential propagation */
141   four1(psi-1, (unsigned int)NX, -1);
142   for (j=0; j<2*NX; j++) {
```

```
143       psi[j] /= NX;
144     }
145     prop(ut); /* full step kinetic propagation    */
146     four1(psi-1, (unsigned int)NX, 1);
147     prop(uv);  /* half step potential propagation */
148 }
149 
150 /*------------------------------------------------------------------*/
151 
152 
153 /*------------------------------------------------------------------*/
154 void calc_energy() {
155 /*------------------------------------------------------------------
156     Calculates the kinetic, potential & total energies, EKIN, EPOT &
          ETOT.
157 ------------------------------------------------------------------*/
158     int sx,s,j;
159     double a,bx;
160 
161     /* Tridiagonal kinetic-energy operators */
162     a =   1.0/(dx*dx);
163     bx = -0.5/(dx*dx);
164 
165     four1(psi-1, (unsigned int)NX, -1);
166     for (j=0; j<2*NX; j++)
167       psi[j] /= NX;
168 
169     /* kinetic energy */
170     ekin = 0.0;
171     for (sx=0; sx<NX; sx++)
172       ekin += 0.5*km[sx]*km[sx]*(psi[2*sx+0]*psi[2*sx+0] +
              psi[2*sx+1]*psi[2*sx+1]);
173     ekin *= NX*dx;
174 
175     four1(psi-1, (unsigned int)NX, 1);
176     /* Potential energy */
177     epot = 0.0;
178     for (sx=0; sx<NX; sx++)
179       epot += v[sx]*(psi[2*sx+0]*psi[2*sx+0] + psi[2*sx+1]*psi[2*sx+1]);
180     epot *= dx;
181 
182     /* Total energy */
183     etot = ekin+epot;
184 }
```

qd1Spectral.c