Tarea 6

Física Computacional

Diego Sarceño

201900109

18 de noviembre de 2022

Los códigos tanto de *c++* como de *gnuplot*, se pueden encontrar en la carpeta de Github.

Problema 1

Utilizamos el código para $N = 64$ y $N = 128$ utilizando el delta dado ($\mathrm{d}t = 5 \times 10^{-18}$)

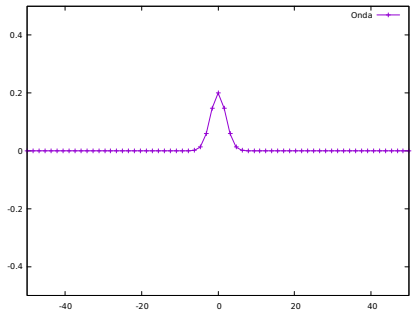

Figura 1: Amplitud de onda para $N = 64$.

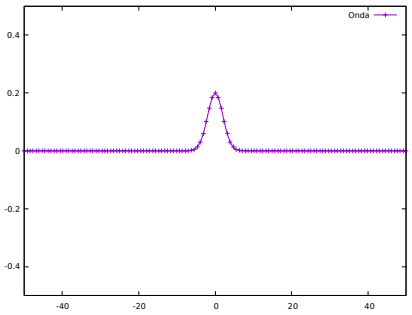

Figura 2: Amplitud de onda para $N = 128$.

```
// Librerias
#include <cmath>
#include <iostream>
#include <fstream>
#include <complex>
#include <iomanip>


using namespace std;



void output( complex<double> *u, double *x, double tiempo, int N,
    ostream &of );
void fourier( complex<double> *ftrans, complex<double> *f, int n );
void fourierInversa( complex<double> *f, complex<double> *ftrans, int n
    );


ofstream solucion;
complex<double> I(0.0, 1.0);
```

```
20
21 int main()
22 {
23   int N = 128;
24   int Niter = 100;
25   int outCada = 1;
26   double tiempo = 0.0;
27   double L = 50.0;
28   double k0 = 2*M_PI/(N+1);
29   double hbar = sqrt(7.6199682);
30   double masa = 1.0;
31   double dx    = 2*L/N;
32   double dt    = 5e-18;
33   double delta_x = 2.0; // Ancho del paquete
34   double k0momentum = sqrt(2*masa*2)/hbar;  // T=p^2/(2m), p=hbar*k
35   solucion.open( "solucion.dat", ios::out );
36
37   // Cantidades complejas
38   complex<double> *psi, *trans, *phi, *expV, *expT;
39   psi    = new complex<double>[ N+1 ];
40   phi    = new complex<double>[ N+1 ];
41   trans  = new complex<double>[ N+1 ];
42   expV   = new complex<double>[ N+1 ];
43   expT   = new complex<double>[ N+1 ];
44
45   // Cantidades reales
46   double *x, *k, *V;
47   k = new double[ N+1 ];
48   x = new double[ N+1 ];
49   V = new double[ N+1 ];
50
51
52
53   // Inicializar coordenada x
54   for(int i=0; i<N+1; i++)
55     x[i] = -L + i*dx;
56
57   // Inicializar k
58   for(int i=0; i<(N+1)/2; i++)
59     k[i] = i*k0;
60
61   for(int i=(N+1)/2; i<N+1; i++)
62     k[i] = -k[N+1-i];
63
64
65   // Inicializar Potencial
66   for(int i=0; i<N+1; i++){
67     if ( 20<=x[i] && x[i]<=30 )
```

```
68        V[i] = 10.0;
69      else
70        V[i] = 0.0;
71
72    }
73
74    // Inicializar expomenciales de T y V
75    for(int i=0; i<N+1; i++){
76      expV[i] = exp( -I*V[i]*dt/(2*hbar) );
77      expT[i] = exp( -I*hbar*k[i]*k[i]*dt/(2*masa) );
78    }
79
80
81
82
83
84    // condiciones iniciales
85    for(int i=0; i<N+1; i++)
86      psi[i] = exp(I*k0momentum*x[i] - x[i]*x[i]/pow(2*delta_x,2) )/pow(2*
           M_PI*pow(delta_x,2),0.25);
87
88
89    // ciclo principal
90    for(int j=0; j<=Niter; j++){
91
92      if ( j%outCada==0 ){
93        cout << "it␣=␣" << j << "␣/␣" << Niter << endl;;
94        output( psi, x, tiempo, N, solucion );
95      }
96
97
98      // Aplicacion de los operadores
99      for(int i=0; i<N+1; i++)
100       phi[i] = expV[i] * psi[i];
101
102     fourier( trans, phi, N+1 );
103
104     for(int i=0; i<N+1; i++)
105       phi[i] = expT[i] * trans[i];
106
107     fourierInversa( psi, phi, N+1 );
108
109     for(int i=0; i<N+1; i++)
110       psi[i] = expV[i] * psi[i];
111
112
113     // condiciones de frontera
114     psi[0] = 0.0;
```

```
115     psi[N] = 0.0;
116
117
118     tiempo += dt;
119
120   }
121
122   return 0;
123 }
124
125
126
127 /*************************************************************************/


128
129
130
131 void output( complex<double> *psi, double *x, double tiempo, int N,
        ostream &of )
132 {
133   for(int i=0; i<N+1; i++)
134     of << tiempo << "\t" << x[i] << "\t"  << real(psi[i]) << "\t" <<
          imag(psi[i]) << endl;
135
136   of << endl << endl;
137 }
138
139
140
141 void fourier( complex<double> *ftrans, complex<double> *f, int n )
142 {
143   for( int i=0; i<n+1; i++ ){
144     ftrans[i] = 0.0;
145     for( int j=0; j<n+1; j++ )
146       ftrans[i] += f[j] * exp(-2*M_PI*I * (double)j * (double)i / (
            double)n);
147
148     ftrans[i] /= sqrt(n);
149   }
150 }
151
152
153 void fourierInversa( complex<double> *f, complex<double> *ftrans, int n
        )
154 {
155   for( int i=0; i<n+1; i++ ){
156     f[i] = 0.0;
157     for( int j=0; j<n+1; j++ )
```

```
158       f[i] += ftrans[j] * exp(2*M_PI*I*(double)j*(double)i/(double)n);
159
160     f[i] /= sqrt(n);
161   }
162 }
```

Problema 2

Utilizando el mismo código, pero con energía inicial $E_o = 150eV$ y realizamos 100 pasos en el tiempo, con lo que tenemos las siguientes gráficas.

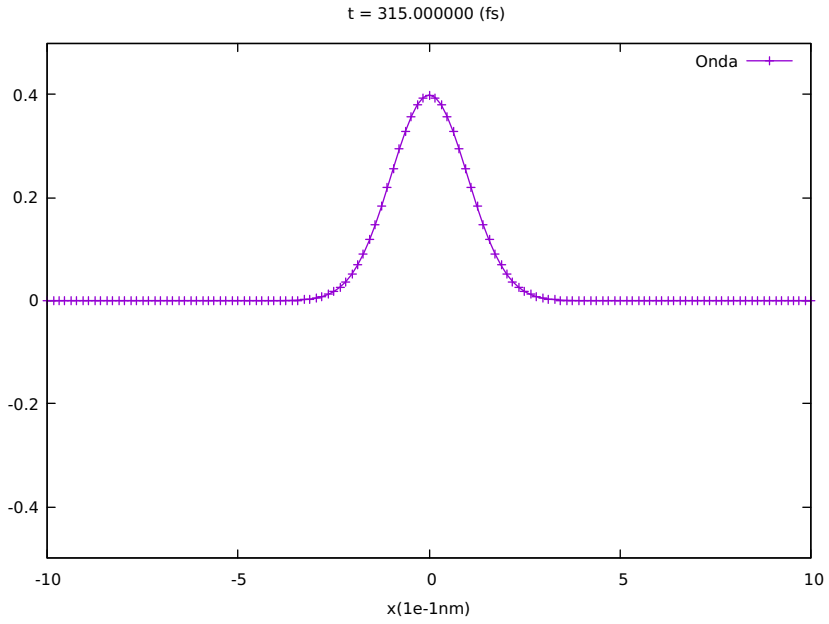

Figura 3: Onda para $E_o = 150eV$.

```
1  // Librerias
2  #include <cmath>
3  #include <iostream>
4  #include <fstream>
5  #include <complex>
6  #include <iomanip>
7
8
9  using namespace std;
10
11
12
13 void output( complex<double> *u, double *x, double tiempo, int N,
       ostream &of );
14 void fourier( complex<double> *ftrans, complex<double> *f, int n );
```

```
15 void fourierInversa( complex<double> *f, complex<double> *ftrans, int n
       );
16
17
18 ofstream solucion;
19 complex<double> I(0.0, 1.0);
20
21 int main()
22 {
23   int N = 128;
24   int Niter = 100;
25   int outCada = 1;
26   double tiempo = 0.0;
27   double L = 10.0;
28   double k0 = 2*M_PI/(N+1);
29   double hbar = sqrt(7.6199682);
30   double masa = 1.0;
31   double dx    = 2*L/N;
32   double dt    = 5e-18;
33   double delta_x = 1.0; // Ancho del paquete
34   double k0momentum = sqrt(2*masa*150)/hbar;  // T=p^2/(2m), p=hbar*k
35   solucion.open( "solucion.dat", ios::out );
36
37   // Cantidades complejas
38   complex<double> *psi, *trans, *phi, *expV, *expT;
39   psi    = new complex<double>[ N+1 ];
40   phi    = new complex<double>[ N+1 ];
41   trans  = new complex<double>[ N+1 ];
42   expV   = new complex<double>[ N+1 ];
43   expT   = new complex<double>[ N+1 ];
44
45   // Cantidades reales
46   double *x, *k, *V;
47   k = new double[ N+1 ];
48   x = new double[ N+1 ];
49   V = new double[ N+1 ];
50
51
52
53   // Inicializar coordenada x
54   for(int i=0; i<N+1; i++)
55     x[i] = -L + i*dx;
56
57   // Inicializar k
58   for(int i=0; i<(N+1)/2; i++)
59     k[i] = i*k0;
60
61   for(int i=(N+1)/2; i<N+1; i++)
```

```
62      k[i] = -k[N+1-i];
63
64
65    // Inicializar Potencial
66    for(int i=0; i<N+1; i++){
67      if ( 20<=x[i] && x[i]<=30 )
68        V[i] = 0.0;
69      else
70        V[i] = 0.0;
71
72    }
73
74    // Inicializar expomenciales de T y V
75    for(int i=0; i<N+1; i++){
76      expV[i] = exp( -I*V[i]*dt/(2*hbar) );
77      expT[i] = exp( -I*hbar*k[i]*k[i]*dt/(2*masa) );
78    }
79
80
81
82    /* condiciones de frontera */
83    //psi[0] = 0;
84    //psi[N] = 0;
85
86
87    // condiciones iniciales
88    for(int i=0; i<N+1; i++)
89      psi[i] = exp(I*k0momentum*x[i] - x[i]*x[i]/pow(2*delta_x,2) )/pow(2*
           M_PI*pow(delta_x,2),0.25);
90
91
92    // ciclo principal
93    for(int j=0; j<=Niter; j++){
94
95      if ( j%outCada==0 ){
96        cout << "it␣=␣" << j << "␣/␣" << Niter << endl;;
97        output( psi, x, tiempo, N, solucion );
98      }
99
100
101     // Aplicacion de los operadores
102     for(int i=0; i<N+1; i++)
103       phi[i] = expV[i] * psi[i];
104
105     fourier( trans, phi, N+1 );
106
107     for(int i=0; i<N+1; i++)
108       phi[i] = expT[i] * trans[i];
```

```
109
110     fourierInversa( psi, phi, N+1 );
111
112     for(int i=0; i<N+1; i++)
113       psi[i] = expV[i] * psi[i];
114
115
116
117
118
119     tiempo += dt;
120
121   }
122
123   return 0;
124 }
125
126
127
128 /*************************************************************************/

129
130
131
132 void output( complex<double> *psi, double *x, double tiempo, int N,
      ostream &of )
133 {
134   for(int i=0; i<N+1; i++)
135     of << tiempo << "\t" << x[i] << "\t"  << real(psi[i]) << "\t" <<
          imag(psi[i]) << endl;
136
137   of << endl << endl;
138 }
139
140
141
142 void fourier( complex<double> *ftrans, complex<double> *f, int n )
143 {
144   for( int i=0; i<n+1; i++ ){
145     ftrans[i] = 0.0;
146     for( int j=0; j<n+1; j++ )
147       ftrans[i] += f[j] * exp(-2*M_PI*I * (double)j * (double)i / (
            double)n);
148
149     ftrans[i] /= sqrt(n);
150   }
151 }
152
```

```
153
154 void fourierInversa( complex<double> *f, complex<double> *ftrans, int n
    )
155 {
156   for( int i=0; i<n+1; i++ ){
157     f[i] = 0.0;
158     for( int j=0; j<n+1; j++ )
159       f[i] += ftrans[j] * exp(2*M_PI*I*(double)j*(double)i/(double)n);
160
161     f[i] /= sqrt(n);
162   }
163 }
```

Problema 3

Dada la barrera de potencial

$$V(x) = V_o, \qquad \forall \quad x \geq 0,$$

y energía $E_o = 100eV$. Se tienen los siguientes resultados

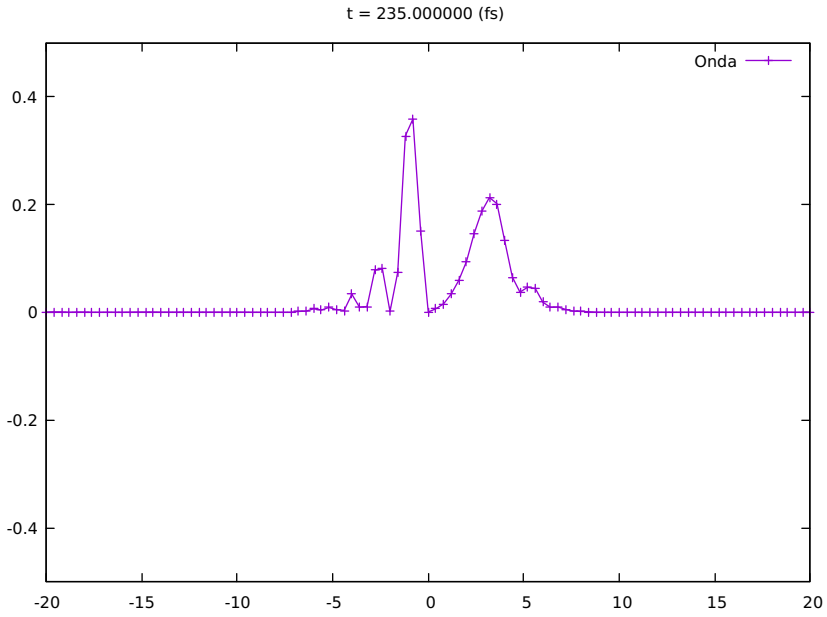

Figura 4: Onda con una barrera depotencial $V_o = 200eV$.

```
1 // Librerias
2 #include <cmath>
3 #include <iostream>
4 #include <fstream>
5 #include <complex>
6 #include <iomanip>
7
```

```
8
9  using namespace std;
10
11
12
13 void output( complex<double> *u, double *x, double tiempo, int N,
       ostream &of );
14 void fourier( complex<double> *ftrans, complex<double> *f, int n );
15 void fourierInversa( complex<double> *f, complex<double> *ftrans, int n
       );
16
17
18 ofstream solucion;
19 complex<double> I(0.0, 1.0);
20
21 int main()
22 {
23   int N = 100;
24   int Niter = 100;
25   int outCada = 1;
26   double tiempo = 0.0;
27   double L = 20.0;
28   double k0 = 2*M_PI/(N+1);
29   double hbar = sqrt(7.6199682);
30   double masa = 1.0;
31   double dx    = 2*L/N;
32   double dt    = 5e-2;
33   double delta_x = 2.0; // Ancho del paquete
34   double k0momentum = sqrt(2*masa*10)/hbar;  // T=p^2/(2m), p=hbar*k
35   solucion.open( "solucion.dat", ios::out );
36
37   // Cantidades complejas
38   complex<double> *psi, *trans, *phi, *expV, *expT;
39   psi    = new complex<double>[ N+1 ];
40   phi    = new complex<double>[ N+1 ];
41   trans  = new complex<double>[ N+1 ];
42   expV   = new complex<double>[ N+1 ];
43   expT   = new complex<double>[ N+1 ];
44
45   // Cantidades reales
46   double *x, *k, *V;
47   k = new double[ N+1 ];
48   x = new double[ N+1 ];
49   V = new double[ N+1 ];
50
51
52
53   // Inicializar coordenada x
```

```
54   for(int i=0; i<N+1; i++)
55     x[i] = -L + i*dx;
56
57   // Inicializar k
58   for(int i=0; i<(N+1)/2; i++)
59     k[i] = i*k0;
60
61   for(int i=(N+1)/2; i<N+1; i++)
62     k[i] = -k[N+1-i];
63
64
65   // Inicializar Potencial
66   for(int i=0; i<N+1; i++){
67     if ( 0<=x[i] && x[i]<=20 )
68       V[i] = 200.0;
69     else
70       V[i] = 0.0;
71
72   }
73
74   // Inicializar expomenciales de T y V
75   for(int i=0; i<N+1; i++){
76     expV[i] = exp( -I*V[i]*dt/(2*hbar) );
77     expT[i] = exp( -I*hbar*k[i]*k[i]*dt/(2*masa) );
78   }
79
80
81
82   /* condiciones de frontera */
83   //psi[0] = 0;
84   //psi[N] = 0;
85
86
87   // condiciones iniciales
88   for(int i=0; i<N+1; i++)
89     psi[i] = exp(I*k0momentum*x[i] - x[i]*x[i]/pow(2*delta_x,2) )/pow(2*
          M_PI*pow(delta_x,2),0.25);
90
91
92   // ciclo principal
93   for(int j=0; j<=Niter; j++){
94
95     if ( j%outCada==0 ){
96       cout << "it␣=␣" << j << "␣/␣" << Niter << endl;;
97       output( psi, x, tiempo, N, solucion );
98     }
99
100
```

```
101     // Aplicacion de los operadores
102     for(int i=0; i<N+1; i++)
103       phi[i] = expV[i] * psi[i];
104 
105     fourier( trans, phi, N+1 );
106 
107     for(int i=0; i<N+1; i++)
108       phi[i] = expT[i] * trans[i];
109 
110     fourierInversa( psi, phi, N+1 );
111 
112     for(int i=0; i<N+1; i++)
113       psi[i] = expV[i] * psi[i];
114 
115 
116     // condiciones de frontera
117     psi[0] = 0.0;
118     psi[N] = 0.0;
119 
120 
121     tiempo += dt;
122 
123   }
124 
125   return 0;
126 }
127 
128 
129 
130 /*************************************************************************/
131 
132 
133 
134 void output( complex<double> *psi, double *x, double tiempo, int N,
       ostream &of )
135 {
136   for(int i=0; i<N+1; i++)
137     of << tiempo << "\t" << x[i] << "\t"  << real(psi[i]) << "\t" <<
          imag(psi[i]) << endl;
138 
139   of << endl << endl;
140 }
141 
142 
143 
144 void fourier( complex<double> *ftrans, complex<double> *f, int n )
145 {
```

```
146   for( int i=0; i<n+1; i++ ){
147     ftrans[i] = 0.0;
148     for( int j=0; j<n+1; j++ )
149       ftrans[i] += f[j] * exp(-2*M_PI*I * (double)j * (double)i / (
              double)n);
150
151     ftrans[i] /= sqrt(n);
152   }
153 }
154
155
156 void fourierInversa( complex<double> *f, complex<double> *ftrans, int n
      )
157 {
158   for( int i=0; i<n+1; i++ ){
159     f[i] = 0.0;
160     for( int j=0; j<n+1; j++ )
161       f[i] += ftrans[j] * exp(2*M_PI*I*(double)j*(double)i/(double)n);
162
163     f[i] /= sqrt(n);
164   }
165 }
```

Problema 4

Dados los mismos datos que el ejercicio anterior, pero con $E_o = 225eV$.

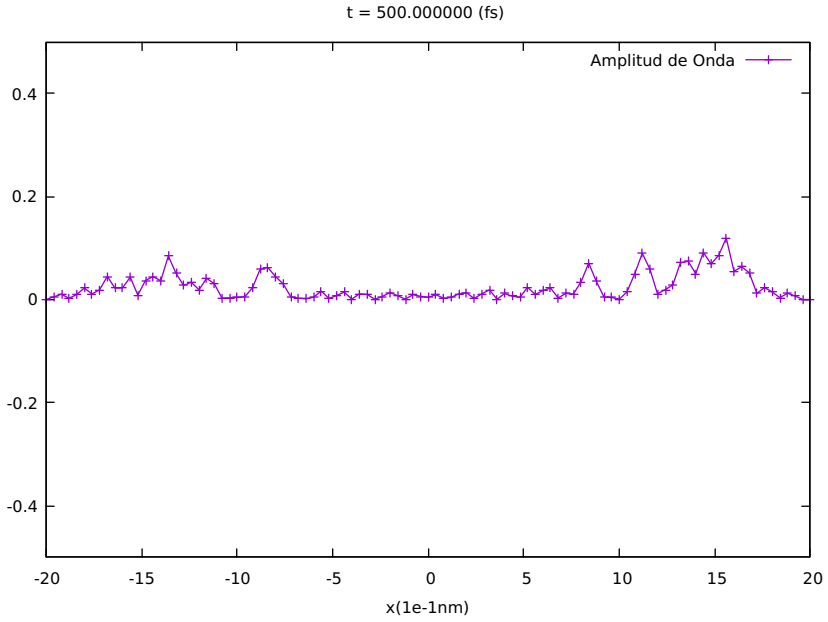

Figura 5: Onda con una barrera depotencial $V_o = 200eV$, pero con una energía inicial de $225eV$. Con esto se ve claro que parte de la onda supera dicha barrera de potencial.

```
// Librerias
#include <cmath>
#include <iostream>
#include <fstream>
#include <complex>
#include <iomanip>


using namespace std;



void output( complex<double> *u, double *x, double tiempo, int N,
    ostream &of );
void fourier( complex<double> *ftrans, complex<double> *f, int n );
void fourierInversa( complex<double> *f, complex<double> *ftrans, int n
    );


ofstream solucion;
complex<double> I(0.0, 1.0);

int main()
{
  int N = 100;
  int Niter = 100;
  int outCada = 1;
  double tiempo = 0.0;
  double L = 20.0;
  double k0 = 2*M_PI/(N+1);
  double hbar = sqrt(7.6199682);
  double masa = 1.0;
  double dx    = 2*L/N;
  double dt    = 5e-2;
  double delta_x = 1.0; // Ancho del paquete
  double k0momentum = sqrt(2*masa*225)/hbar;  // T=p^2/(2m), p=hbar*k
  solucion.open( "solucion.dat", ios::out );

  // Cantidades complejas
  complex<double> *psi, *trans, *phi, *expV, *expT;
  psi    = new complex<double>[ N+1 ];
  phi    = new complex<double>[ N+1 ];
  trans  = new complex<double>[ N+1 ];
  expV   = new complex<double>[ N+1 ];
  expT   = new complex<double>[ N+1 ];

  // Cantidades reales
  double *x, *k, *V;
```

```
47   k = new double[ N+1 ];
48   x = new double[ N+1 ];
49   V = new double[ N+1 ];
50
51
52
53   // Inicializar coordenada x
54   for(int i=0; i<N+1; i++)
55     x[i] = -L + i*dx;
56
57   // Inicializar k
58   for(int i=0; i<(N+1)/2; i++)
59     k[i] = i*k0;
60
61   for(int i=(N+1)/2; i<N+1; i++)
62     k[i] = -k[N+1-i];
63
64
65   // Inicializar Potencial
66   for(int i=0; i<N+1; i++){
67     if ( 0<=x[i] && x[i]<=20 )
68       V[i] = 200.0;
69     else
70       V[i] = 0.0;
71
72   }
73
74   // Inicializar expomenciales de T y V
75   for(int i=0; i<N+1; i++){
76     expV[i] = exp( -I*V[i]*dt/(2*hbar) );
77     expT[i] = exp( -I*hbar*k[i]*k[i]*dt/(2*masa) );
78   }
79
80
81
82   /* condiciones de frontera */
83   //psi[0] = 0;
84   //psi[N] = 0;
85
86
87   // condiciones iniciales
88   for(int i=0; i<N+1; i++)
89     psi[i] = exp(I*k0momentum*x[i] - x[i]*x[i]/pow(2*delta_x,2) )/pow(2*
         M_PI*pow(delta_x,2),0.25);
90
91
92   // ciclo principal
93   for(int j=0; j<=Niter; j++){
```

```
94
95      if ( j%outCada==0 ){
96        cout << "it␣=␣" << j << "␣/␣" << Niter << endl;;
97        output( psi, x, tiempo, N, solucion );
98      }
99
100
101     // Aplicacion de los operadores
102     for(int i=0; i<N+1; i++)
103       phi[i] = expV[i] * psi[i];
104
105     fourier( trans, phi, N+1 );
106
107     for(int i=0; i<N+1; i++)
108       phi[i] = expT[i] * trans[i];
109
110     fourierInversa( psi, phi, N+1 );
111
112     for(int i=0; i<N+1; i++)
113       psi[i] = expV[i] * psi[i];
114
115
116
117
118
119     tiempo += dt;
120
121   }
122
123   return 0;
124 }
125
126
127
128 /*************************************************************************/


129
130
131
132
133
134
135 void output( complex<double> *psi, double *x, double tiempo, int N,
       ostream &of )
136 {
137   for(int i=0; i<N+1; i++)
138     of << tiempo << "\t" << x[i] << "\t"  << real(psi[i]) << "\t" <<
          imag(psi[i]) << endl;
```

```
139
140   of << endl << endl;
141 }
142
143
144
145 void fourier( complex<double> *ftrans, complex<double> *f, int n )
146 {
147   for( int i=0; i<n+1; i++ ){
148     ftrans[i] = 0.0;
149     for( int j=0; j<n+1; j++ )
150       ftrans[i] += f[j] * exp(-2*M_PI*I * (double)j * (double)i / (
              double)n);
151
152     ftrans[i] /= sqrt(n);
153   }
154 }
155
156
157 void fourierInversa( complex<double> *f, complex<double> *ftrans, int n
      )
158 {
159   for( int i=0; i<n+1; i++ ){
160     f[i] = 0.0;
161     for( int j=0; j<n+1; j++ )
162       f[i] += ftrans[j] * exp(2*M_PI*I*(double)j*(double)i/(double)n);
163
164     f[i] /= sqrt(n);
165   }
166 }
```

Problema 5

Ahora, se tiene un pozo de potencial con $V_o = -200eV$ en el rango de $0 \leq x \leq a$, $a = 1.963A$ y $-30A \leq x \leq 30A$. Con esto se tienen las siguientes gráficas de la onda pasando por el pozo

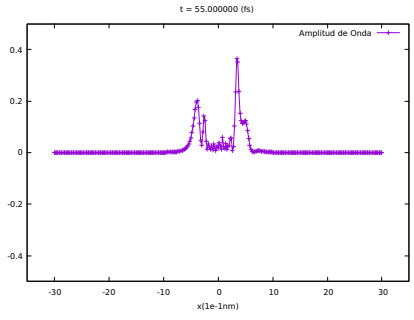

Figura 6: Onda con $E_o = 100eV$.

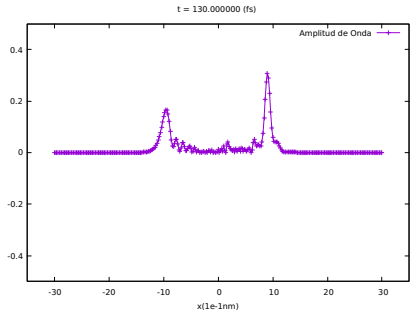

Figura 7: Onda luego de pasar el pozo.

```
// Librerias
#include <cmath>
#include <iostream>
#include <fstream>
#include <complex>
#include <iomanip>


using namespace std;



void output( complex<double> *u, double *x, double tiempo, int N,
    ostream &of );
void fourier( complex<double> *ftrans, complex<double> *f, int n );
void fourierInversa( complex<double> *f, complex<double> *ftrans, int n
    );


ofstream solucion;
complex<double> I(0.0, 1.0);

int main()
{
  int N = 300;
  int Niter = 30;
  int outCada = 1;
  double tiempo = 0.0;
  double L = 30.0;
  double k0 = 2*M_PI/(N+1);
  double hbar = sqrt(7.6199682);
  double masa = 8.0;
  double dx    = 2*L/N;
  double dt    = 2;
  double delta_x = 1.0; // Ancho del paquete
  double k0momentum = sqrt(2*masa*100)/hbar;  // T=p^2/(2m), p=hbar*k
  double a_potencial = 1.963; //
  solucion.open( "solucion.dat", ios::out );

  // Cantidades complejas
  complex<double> *psi, *trans, *phi, *expV, *expT;
  psi    = new complex<double>[ N+1 ];
  phi    = new complex<double>[ N+1 ];
  trans  = new complex<double>[ N+1 ];
  expV   = new complex<double>[ N+1 ];
  expT   = new complex<double>[ N+1 ];

  // Cantidades reales
```

```
47   double *x, *k, *V;
48   k = new double[ N+1 ];
49   x = new double[ N+1 ];
50   V = new double[ N+1 ];
51
52
53
54   // Inicializar coordenada x
55   for(int i=0; i<N+1; i++)
56     x[i] = -L + i*dx;
57
58   // Inicializar k
59   for(int i=0; i<(N+1)/2; i++)
60     k[i] = i*k0;
61
62   for(int i=(N+1)/2; i<N+1; i++)
63     k[i] = -k[N+1-i];
64
65
66   // Inicializar Potencial
67   for(int i=0; i<N+1; i++){
68     if ( 0<=x[i] && x[i]<=a_potencial )
69       V[i] = -200.0;
70     else
71       V[i] = 0.0;
72
73   }
74
75   // Inicializar expomenciales de T y V
76   for(int i=0; i<N+1; i++){
77     expV[i] = exp( -I*V[i]*dt/(2*hbar) );
78     expT[i] = exp( -I*hbar*k[i]*k[i]*dt/(2*masa) );
79   }
80
81
82
83   /* condiciones de frontera */
84   //psi[0] = 0;
85   //psi[N] = 0;
86
87
88   // condiciones iniciales
89   for(int i=0; i<N+1; i++)
90     psi[i] = exp(I*k0momentum*x[i] - x[i]*x[i]/pow(2*delta_x,2) )/pow(2*
         M_PI*pow(delta_x,2),0.25);
91
92
93   // ciclo principal
```

```
94    for(int j=0; j<=Niter; j++){
95
96      if ( j%outCada==0 ){
97        cout << "it␣=␣" << j << "␣/␣" << Niter << endl;;
98        output( psi, x, tiempo, N, solucion );
99      }
100
101
102     // Aplicacion de los operadores
103     for(int i=0; i<N+1; i++)
104       phi[i] = expV[i] * psi[i];
105
106     fourier( trans, phi, N+1 );
107
108     for(int i=0; i<N+1; i++)
109       phi[i] = expT[i] * trans[i];
110
111     fourierInversa( psi, phi, N+1 );
112
113     for(int i=0; i<N+1; i++)
114       psi[i] = expV[i] * psi[i];
115
116
117
118     tiempo += dt;
119
120   }
121
122   return 0;
123 }
124
125
126
127 /*********************************************************************/


128
129
130
131
132 void output( complex<double> *psi, double *x, double tiempo, int N,
        ostream &of )
133 {
134   for(int i=0; i<N+1; i++)
135     of << tiempo << "\t" << x[i] << "\t"  << real(psi[i]) << "\t" <<
          imag(psi[i]) << endl;
136
137   of << endl << endl;
138 }
```

```
139
140
141
142 void fourier( complex<double> *ftrans, complex<double> *f, int n )
143 {
144   for( int i=0; i<n+1; i++ ){
145     ftrans[i] = 0.0;
146     for( int j=0; j<n+1; j++ )
147       ftrans[i] += f[j] * exp(-2*M_PI*I * (double)j * (double)i / (
              double)n);
148
149     ftrans[i] /= sqrt(n);
150   }
151 }
152
153
154 void fourierInversa( complex<double> *f, complex<double> *ftrans, int n
     )
155 {
156   for( int i=0; i<n+1; i++ ){
157     f[i] = 0.0;
158     for( int j=0; j<n+1; j++ )
159       f[i] += ftrans[j] * exp(2*M_PI*I*(double)j*(double)i/(double)n);
160
161     f[i] /= sqrt(n);
162   }
163 }
```