Tarea#6

Solución de la ecuación de Schrödinger para un paquete Gausiano utilizando el método pseudo espectral.

Ejercicio 7.16:

Nos piden resolver la ecuación propuesta del libro para 2 configuraciones de redes distintas en el caso estático con, N= 64 y N= 128.

Código cpp:

```
//============================================================
//
// Ecuacion de Schrodinger con metodo pseudo espectral
//
//============================================================

#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 = 64;
  int Niter = 100;
  int outCada =1;
  double tiempo = 0.0;
  double hbar = sqrt(7.6199682);
  double L = 10.0; //vamos a escalar todo al A circulado
  double k0 = 2*M_PI/(N+1);

  double masa = 1.0;
  double dx    = 2*L/N;
```

```
double dt    = 5e-1; //conversiones
double delta_x = 1; // Ancho del paquete
double k0momentum = 0;  // 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;
k = new double[ N+1 ];
x = new double[ N+1 ];
V = new double[ N+1 ];



// Inicializar coordenada x
for(int i=0; i<N+1; i++)
  x[i] = -L + i*dx;

// Inicializar k
for(int i=0; i<(N+1)/2; i++)
  k[i] = i*k0;

for(int i=(N+1)/2; i<N+1; i++)
  k[i] = -k[N+1-i];


// Inicializar Potencial
for(int i=0; i<N+1; i++){
  if ( 20<=x[i] && x[i]<=30 )
    V[i] = 0.0;
  else
    V[i] = 0.0;

}

// Inicializar expomenciales de T y V
for(int i=0; i<N+1; i++){
  expV[i] = exp( -I*V[i]*dt/(2*hbar) );
  expT[i] = exp( -I*hbar*k[i]*k[i]*dt/(2*masa) );
}



/* condiciones de frontera */
```

```
  //psi[0] = 0;
  //psi[N] = 0;


  // condiciones iniciales
  for(int i=0; i<N+1; i++)
    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);


  // ciclo principal
  for(int j=0; j<=Niter; j++){

    if ( j%outCada==0 ){
      cout << "it = " << tiempo << " / " << Niter << endl;;
      output( psi, x, tiempo, N, solucion );
    }


    // Aplicacion de los operadores
    for(int i=0; i<N+1; i++)
      phi[i] = expV[i] * psi[i];

    fourier( trans, phi, N+1 );

    for(int i=0; i<N+1; i++)
      phi[i] = expT[i] * trans[i];

    fourierInversa( psi, phi, N+1 );

    for(int i=0; i<N+1; i++)
      psi[i] = expV[i] * psi[i];


    // condiciones de frontera
    psi[0] = 0.0;
    psi[N] = 0.0;


    tiempo += dt;

  }

  return 0;
}



/*****************************************************************/
```

```
//double f( double x )
//{
  //return sqrt(2.0)*sin(5*x*M_PI);
  //return cexp(-50*pow(x-0.5,2) -I*100*x);
//}


void output( complex<double> *psi, double *x, double tiempo, int N, ostream &of )
{
  for(int i=0; i<N+1; i++)
    of << tiempo << "\t" << x[i] << "\t"  << real(psi[i]) << "\t" << imag(psi[i])
<< endl;

  of << endl << endl;
}



void fourier( complex<double> *ftrans, complex<double> *f, int n )
{
  for( int i=0; i<n+1; i++ ){
    ftrans[i] = 0.0;
    for( int j=0; j<n+1; j++ )
      ftrans[i] += f[j] * exp(-2*M_PI*I * (double)j * (double)i / (double)n);

    ftrans[i] /= sqrt(n);
  }
}


void fourierInversa( complex<double> *f, complex<double> *ftrans, int n )
{
  for( int i=0; i<n+1; i++ ){
    f[i] = 0.0;
    for( int j=0; j<n+1; j++ )
      f[i] += ftrans[j] * exp(2*M_PI*I*(double)j*(double)i/(double)n);

    f[i] /= sqrt(n);
  }
}
```

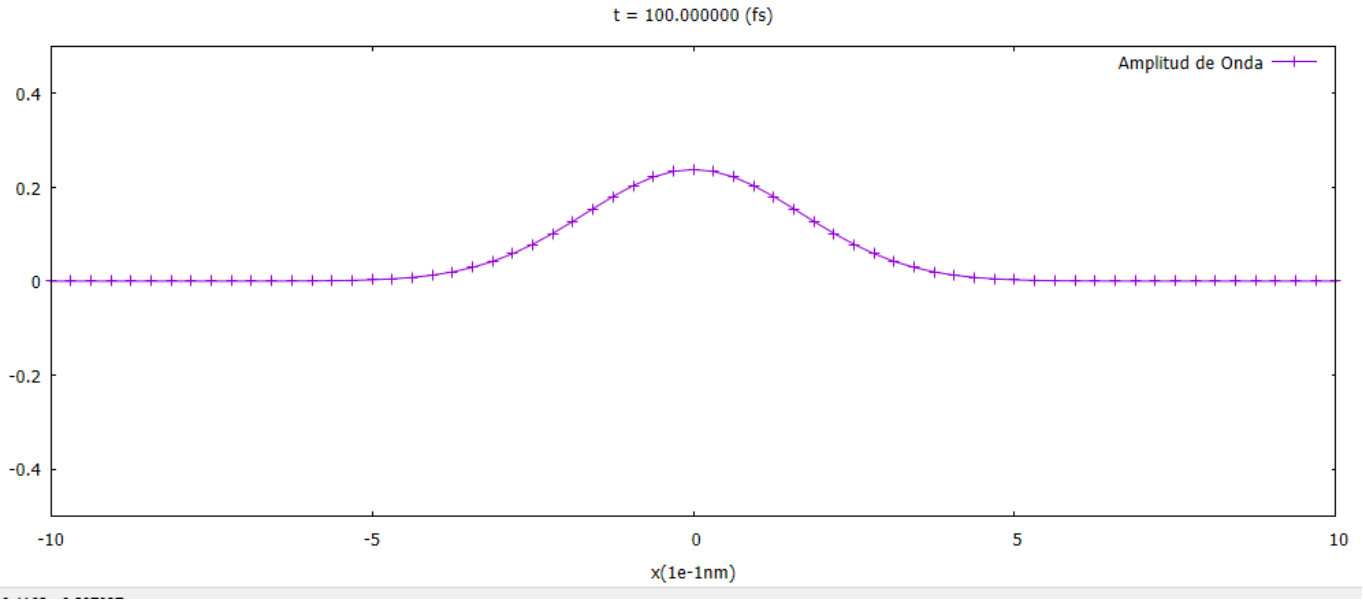
Código gp:

```
set yrange [-0.5:0.5]
set xlabel "x(1e-1nm)"
set xrange [-10:10]
dt = 5
do for [i=0:80] {
```

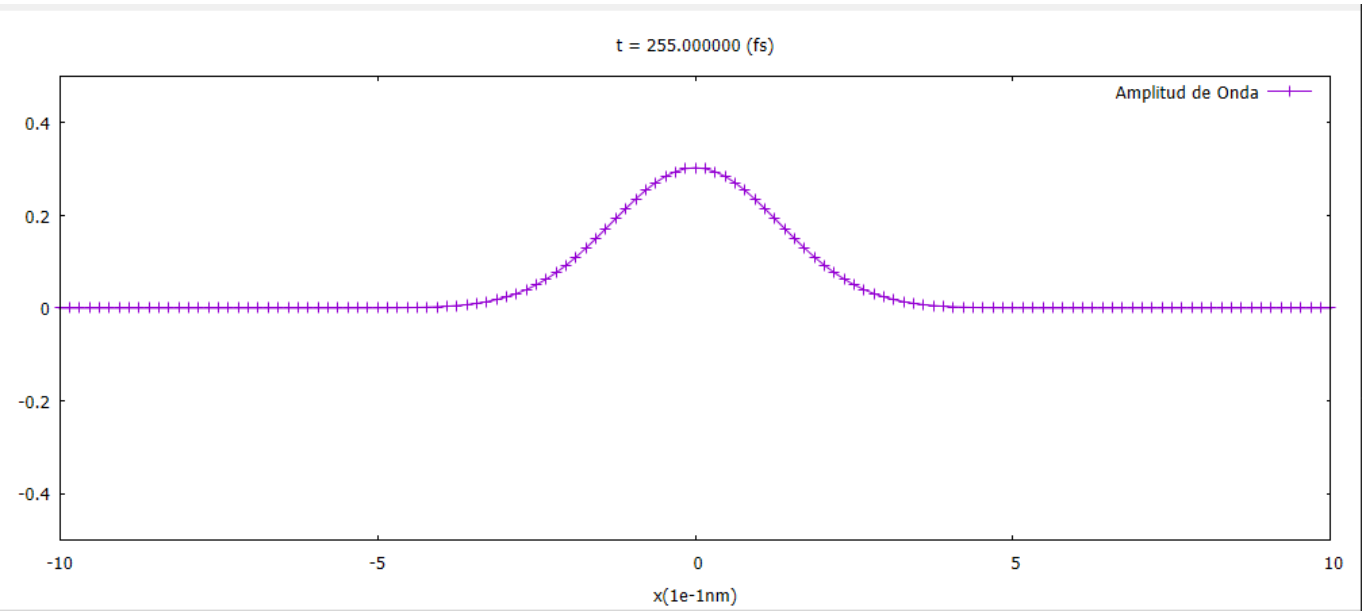
```
    set title sprintf( "t = %f (fs)", i*dt )
  #  plot 'solucion.dat' index i u 2:3 w lp, '' index i u 2:4 w lp
    plot 'solucion.dat' index i u 2:($3**2+$4**2) w lp title "Amplitud de Onda"

    pause .09
}
```

Solución:

Debemos realizar las conversiones a las unidades utilizadas y traducir los parámetros del libro a los de nuestro programa. Al calibrar los parámetros correctamente guardamos los resultados en un archivo .dat y graficamos la solución de la amplitud al cuadrado utilizando gnuplot.



Expansión del paquete de onda con N64



Expansión del paquete de onda con N128

Ejercicio 7.17:

Nos piden resolver la ecuación propuesta del libro para una E0= 150eV con la red más fina y 100 saltos de tiempo.

Código cpp:

```
//============================================================
//
// Ecuacion de Schrodinger con metodo pseudo espectral
//
//============================================================

#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 = 128;
  int Niter = 100;
  int outCada =1;
  double tiempo = 0.0;
  double hbar = sqrt(7.6199682);
  double L = 10.0;
  double k0 = 2*M_PI/(N+1);

  double masa = 1.0;
  double dx    = 2*L/N;
  double dt    = 5e-1;
  double delta_x = 1; // Ancho del paquete
  double k0momentum = sqrt(2*150)/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;
  k = new double[ N+1 ];
  x = new double[ N+1 ];
  V = new double[ N+1 ];



  // Inicializar coordenada x
  for(int i=0; i<N+1; i++)
    x[i] = -L + i*dx;

  // Inicializar k
  for(int i=0; i<(N+1)/2; i++)
    k[i] = i*k0;

  for(int i=(N+1)/2; i<N+1; i++)
    k[i] = -k[N+1-i];


  // Inicializar Potencial
  for(int i=0; i<N+1; i++){
    if ( 20<=x[i] && x[i]<=30 )
      V[i] = 0.0;
    else
      V[i] = 0.0;

  }

  // Inicializar expomenciales de T y V
  for(int i=0; i<N+1; i++){
    expV[i] = exp( -I*V[i]*dt/(2*hbar) );
    expT[i] = exp( -I*hbar*k[i]*k[i]*dt/(2*masa) );
  }



  /* condiciones de frontera */
  //psi[0] = 0;
  //psi[N] = 0;


  // condiciones iniciales
  for(int i=0; i<N+1; i++)
    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);
```

```
  // ciclo principal
  for(int j=0; j<=Niter; j++){

    if ( j%outCada==0 ){
      cout << "it = " << tiempo << " / " << Niter << endl;;
      output( psi, x, tiempo, N, solucion );
    }


    // Aplicacion de los operadores
    for(int i=0; i<N+1; i++)
      phi[i] = expV[i] * psi[i];

    fourier( trans, phi, N+1 );

    for(int i=0; i<N+1; i++)
      phi[i] = expT[i] * trans[i];

    fourierInversa( psi, phi, N+1 );

    for(int i=0; i<N+1; i++)
      psi[i] = expV[i] * psi[i];


    // condiciones de frontera
    psi[0] = 0.0;
    psi[N] = 0.0;


    tiempo += dt;

  }

  return 0;
}



/*************************************************************************/



//double f( double x )
//{
  //return sqrt(2.0)*sin(5*x*M_PI);
  //return cexp(-50*pow(x-0.5,2) -I*100*x);
//}
```

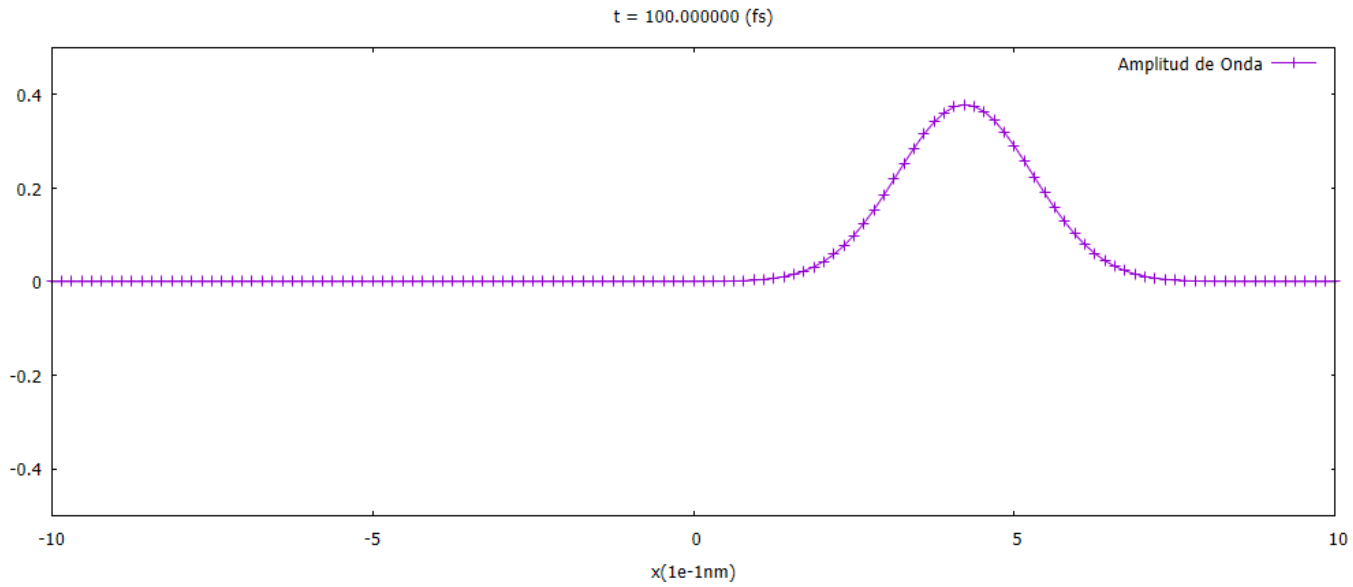
```
void output( complex<double> *psi, double *x, double tiempo, int N, ostream &of )
{
  for(int i=0; i<N+1; i++)
    of << tiempo << "\t" << x[i] << "\t"  << real(psi[i]) << "\t" << imag(psi[i])
<< endl;

  of << endl << endl;
}



void fourier( complex<double> *ftrans, complex<double> *f, int n )
{
  for( int i=0; i<n+1; i++ ){
    ftrans[i] = 0.0;
    for( int j=0; j<n+1; j++ )
      ftrans[i] += f[j] * exp(-2*M_PI*I * (double)j * (double)i / (double)n);

    ftrans[i] /= sqrt(n);
  }
}


void fourierInversa( complex<double> *f, complex<double> *ftrans, int n )
{
  for( int i=0; i<n+1; i++ ){
    f[i] = 0.0;
    for( int j=0; j<n+1; j++ )
      f[i] += ftrans[j] * exp(2*M_PI*I*(double)j*(double)i/(double)n);

    f[i] /= sqrt(n);
  }
}
```

Código gp:

```
set yrange [-0.5:0.5]
set xrange [-10:10]
dt = 5
set xlabel "x(1e-1nm)"
do for [i=0:80] {
    set title sprintf( "t = %f (fs)", i*dt )
  #  plot 'solucion.dat' index i u 2:3 w lp, '' index i u 2:4 w lp
    plot 'solucion.dat' index i u 2:($3**2+$4**2) w lp title "Amplitud de Onda"

    pause .01
}
```

Solución:

El procedimiento de este problema es equivalente al anterior.



Ahora el paquete de onda se mueve hacia la derecha

Ejercicio 7.20:

Nos piden resolver la ecuación propuesta del libro para una barrera de potencial V= 200ev y 50 saltos de tiempo.

Código cpp:

```
//============================================================
//
// Ecuacion de Schrodinger con metodo pseudo espectral
//
//============================================================

#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 = 128;
  int Niter = 100;
  int outCada =1;
  double tiempo = 0.0;
  double hbar = sqrt(7.6199682);
  double L = 20.0;
  double k0 = 2*M_PI/(N+1);

  double masa = 1.0;
  double dx    = 2*L/N;
  double dt    = 5e-1;
  double delta_x = 1; // Ancho del paquete
  double k0momentum = sqrt(2*100)/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;
  k = new double[ N+1 ];
  x = new double[ N+1 ];
  V = new double[ N+1 ];



  // Inicializar coordenada x
  for(int i=0; i<N+1; i++)
    x[i] = -L + i*dx;

  // Inicializar k
  for(int i=0; i<(N+1)/2; i++)
    k[i] = i*k0;

  for(int i=(N+1)/2; i<N+1; i++)
    k[i] = -k[N+1-i];


  // Inicializar Potencial
  for(int i=0; i<N+1; i++){
    if ( 0<=x[i] && x[i]<=20 )
```

```
      V[i] = 200.0;
    else
      V[i] = 0.0;

  }

  // Inicializar expomenciales de T y V
  for(int i=0; i<N+1; i++){
    expV[i] = exp( -I*V[i]*dt/(2*hbar) );
    expT[i] = exp( -I*hbar*k[i]*k[i]*dt/(2*masa) );
  }



  /* condiciones de frontera */
  //psi[0] = 0;
  //psi[N] = 0;


  // condiciones iniciales
  for(int i=0; i<N+1; i++)
    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);


  // ciclo principal
  for(int j=0; j<=Niter; j++){

    if ( j%outCada==0 ){
      cout << "it = " << tiempo << " / " << Niter << endl;;
      output( psi, x, tiempo, N, solucion );
    }


    // Aplicacion de los operadores
    for(int i=0; i<N+1; i++)
      phi[i] = expV[i] * psi[i];

    fourier( trans, phi, N+1 );

    for(int i=0; i<N+1; i++)
      phi[i] = expT[i] * trans[i];

    fourierInversa( psi, phi, N+1 );

    for(int i=0; i<N+1; i++)
      psi[i] = expV[i] * psi[i];


    // condiciones de frontera
    psi[0] = 0.0;
```

```
    psi[N] = 0.0;


    tiempo += dt;

  }

  return 0;
}



/*************************************************************************/



//double f( double x )
//{
  //return sqrt(2.0)*sin(5*x*M_PI);
  //return cexp(-50*pow(x-0.5,2) -I*100*x);
//}


void output( complex<double> *psi, double *x, double tiempo, int N, ostream &of )
{
  for(int i=0; i<N+1; i++)
    of << tiempo << "\t" << x[i] << "\t"  << real(psi[i]) << "\t" << imag(psi[i])
<< endl;

  of << endl << endl;
}



void fourier( complex<double> *ftrans, complex<double> *f, int n )
{
  for( int i=0; i<n+1; i++ ){
    ftrans[i] = 0.0;
    for( int j=0; j<n+1; j++ )
      ftrans[i] += f[j] * exp(-2*M_PI*I * (double)j * (double)i / (double)n);

    ftrans[i] /= sqrt(n);
  }
}


void fourierInversa( complex<double> *f, complex<double> *ftrans, int n )
{
  for( int i=0; i<n+1; i++ ){
    f[i] = 0.0;
    for( int j=0; j<n+1; j++ )
```

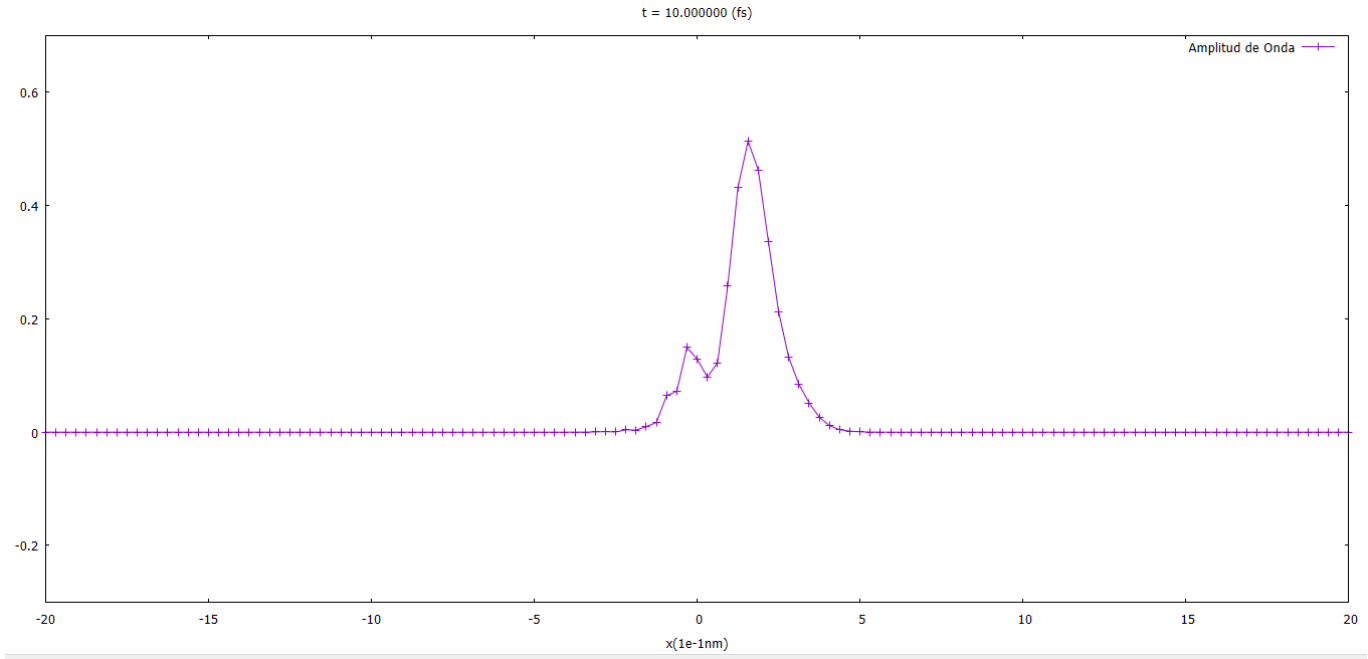
```
      f[i] += ftrans[j] * exp(2*M_PI*I*(double)j*(double)i/(double)n);

    f[i] /= sqrt(n);
  }
}
```

Código gp:

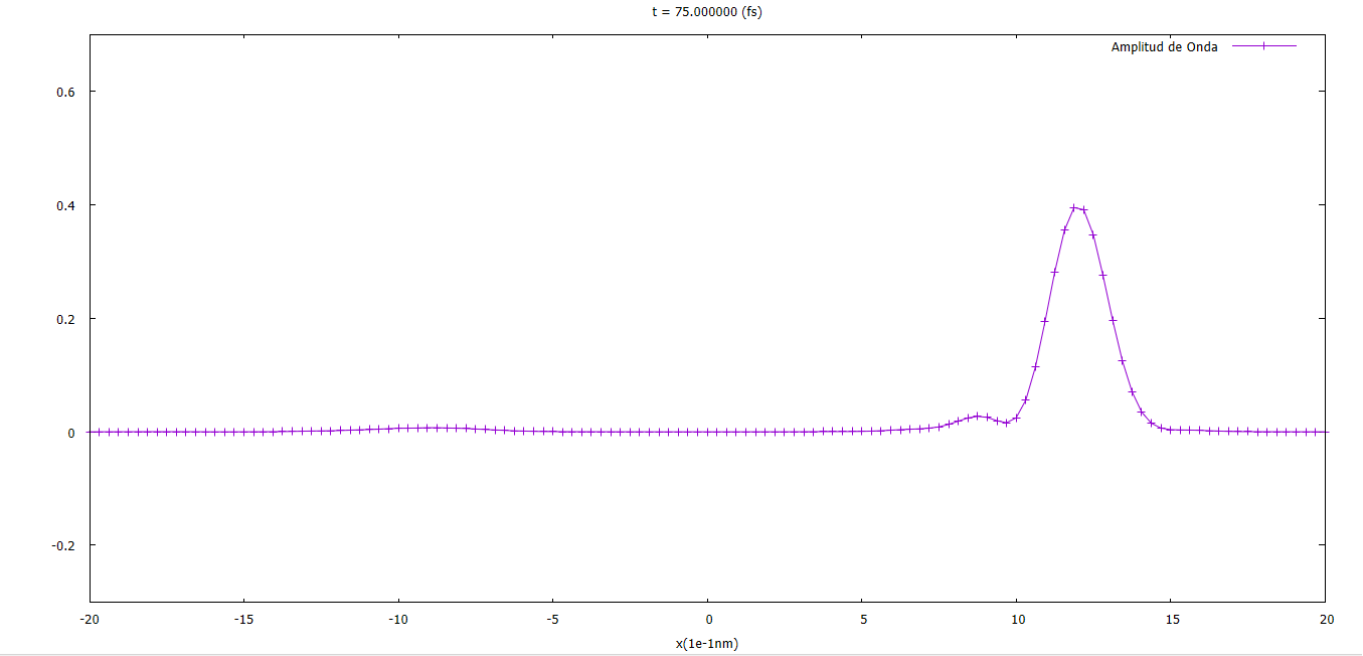
```
set yrange [-0.5:0.5]
set xrange [-20:20]
dt = 5
set xlabel "x(1e-1nm)"
do for [i=0:80] {
    set title sprintf( "t = %f (fs)", i*dt )
  #  plot 'solucion.dat' index i u 2:3 w lp, '' index i u 2:4 w lp
    plot 'solucion.dat' index i u 2:($3**2+$4**2) w lp title "Amplitud de Onda",
step(x) = x>0 ? .2 : 0

    pause -1}
```

Solución:

El procedimiento de este problema es equivalente al anterior, pero ahora añadimos la barrera de potencial en el intervalo deseado.



La onda choca contra la barrera y los componentes reflejados interactuan con los demás.



Evolución temporal del paquete de onda.

Ejercicio 7.22:

Nos piden resolver la ecuación propuesta del libro para la misma barrera de potencial pero ahora con una energía inicial que supera a esta misma.

Código cpp:

```
//============================================================
//
// Ecuacion de Schrodinger con metodo pseudo espectral
//
//============================================================

#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 = 128;
  int Niter = 100;
  int outCada =1;
  double tiempo = 0.0;
  double hbar = sqrt(7.6199682);
  double L = 20.0;
  double k0 = 2*M_PI/(N+1);

  double masa = 1.0;
  double dx    = 2*L/N;
  double dt    = 5e-1;
  double delta_x = 1; // Ancho del paquete
  double k0momentum = sqrt(2*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;
  k = new double[ N+1 ];
  x = new double[ N+1 ];
  V = new double[ N+1 ];



  // Inicializar coordenada x
  for(int i=0; i<N+1; i++)
    x[i] = -L + i*dx;

  // Inicializar k
  for(int i=0; i<(N+1)/2; i++)
    k[i] = i*k0;

  for(int i=(N+1)/2; i<N+1; i++)
    k[i] = -k[N+1-i];


  // Inicializar Potencial
  for(int i=0; i<N+1; i++){
    if ( 0<=x[i] && x[i]<=20 )
      V[i] = 200.0;
```

```
      else
        V[i] = 0.0;

    }

    // Inicializar expomenciales de T y V
    for(int i=0; i<N+1; i++){
      expV[i] = exp( -I*V[i]*dt/(2*hbar) );
      expT[i] = exp( -I*hbar*k[i]*k[i]*dt/(2*masa) );
    }



    /* condiciones de frontera */
    //psi[0] = 0;
    //psi[N] = 0;


    // condiciones iniciales
    for(int i=0; i<N+1; i++)
      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);


    // ciclo principal
    for(int j=0; j<=Niter; j++){

      if ( j%outCada==0 ){
        cout << "it = " << tiempo << " / " << Niter << endl;;
        output( psi, x, tiempo, N, solucion );
      }


      // Aplicacion de los operadores
      for(int i=0; i<N+1; i++)
        phi[i] = expV[i] * psi[i];

      fourier( trans, phi, N+1 );

      for(int i=0; i<N+1; i++)
        phi[i] = expT[i] * trans[i];

      fourierInversa( psi, phi, N+1 );

      for(int i=0; i<N+1; i++)
        psi[i] = expV[i] * psi[i];


      // condiciones de frontera
      psi[0] = 0.0;
      psi[N] = 0.0;
```

```
    tiempo += dt;

  }

  return 0;
}



/*************************************************************************/



//double f( double x )
//{
  //return sqrt(2.0)*sin(5*x*M_PI);
  //return cexp(-50*pow(x-0.5,2) -I*100*x);
//}


void output( complex<double> *psi, double *x, double tiempo, int N, ostream &of )
{
  for(int i=0; i<N+1; i++)
    of << tiempo << "\t" << x[i] << "\t"  << real(psi[i]) << "\t" << imag(psi[i])
<< endl;

  of << endl << endl;
}



void fourier( complex<double> *ftrans, complex<double> *f, int n )
{
  for( int i=0; i<n+1; i++ ){
    ftrans[i] = 0.0;
    for( int j=0; j<n+1; j++ )
      ftrans[i] += f[j] * exp(-2*M_PI*I * (double)j * (double)i / (double)n);

    ftrans[i] /= sqrt(n);
  }
}


void fourierInversa( complex<double> *f, complex<double> *ftrans, int n )
{
  for( int i=0; i<n+1; i++ ){
    f[i] = 0.0;
    for( int j=0; j<n+1; j++ )
      f[i] += ftrans[j] * exp(2*M_PI*I*(double)j*(double)i/(double)n);
```

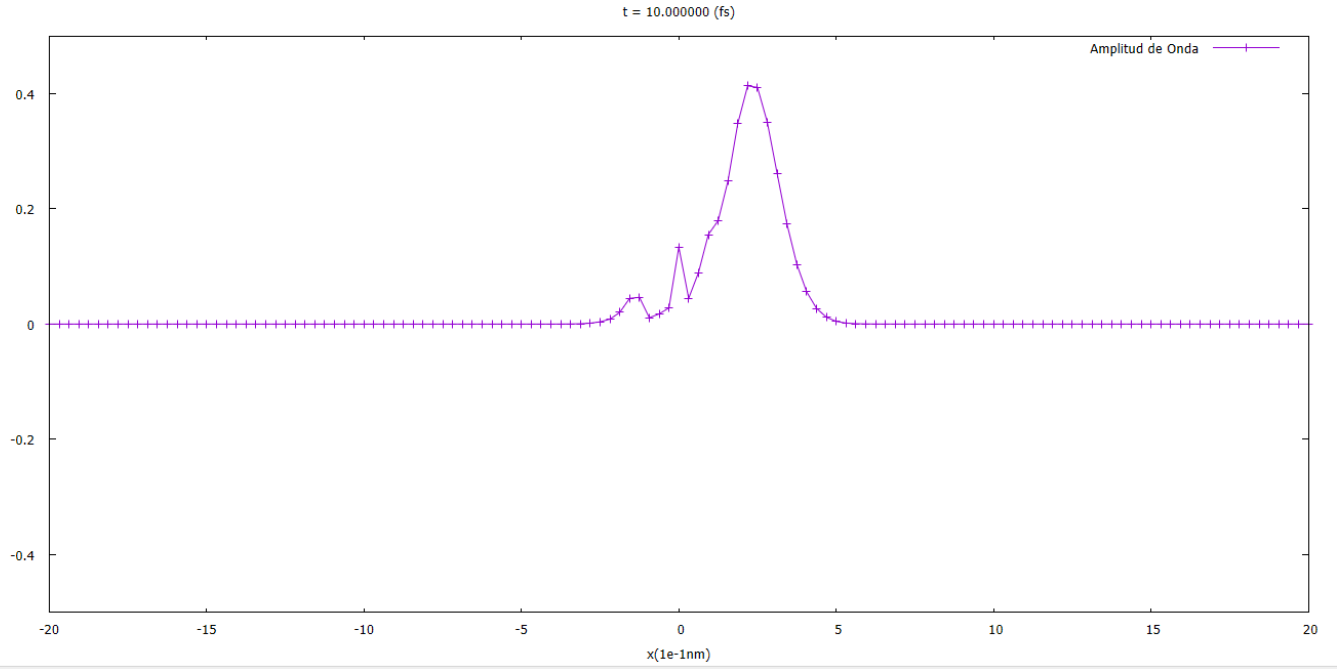
```
    f[i] /= sqrt(n);
  }
}
```

Código gp:

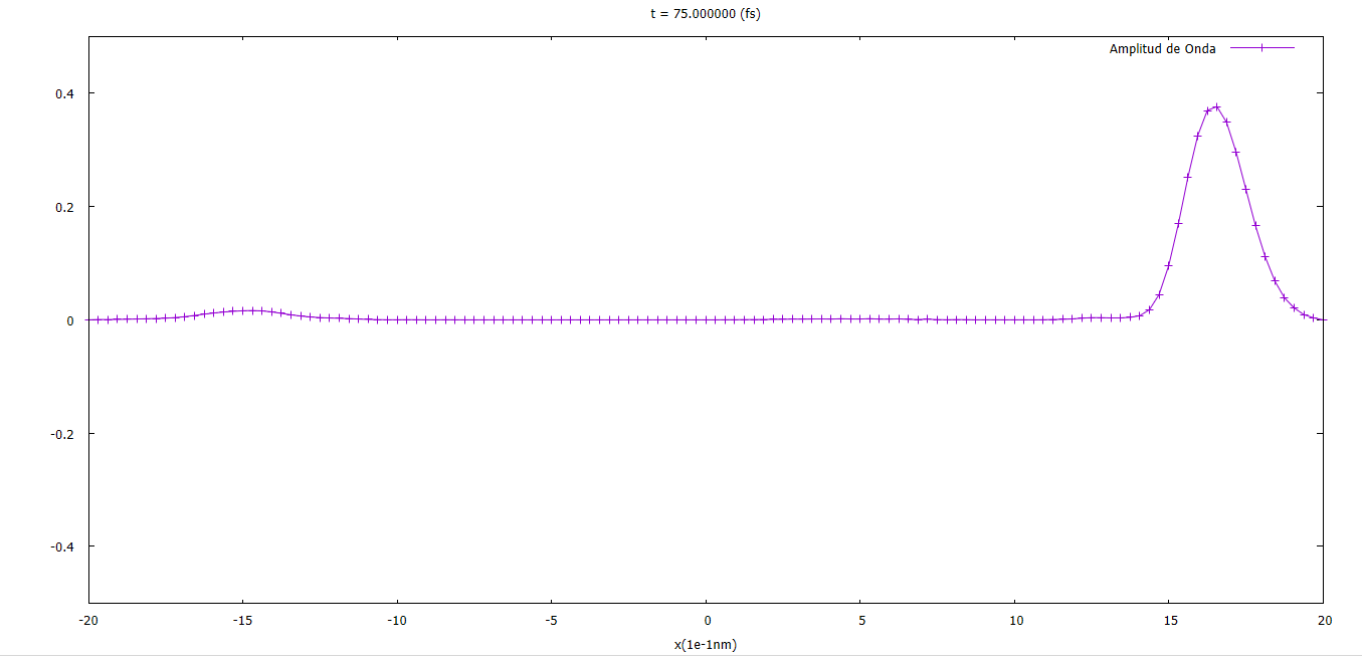
```
set yrange [-0.5:0.5]
set xrange [-20:20]
dt = 5
set xlabel "x(1e-1nm)"
do for [i=0:80] {
    set title sprintf( "t = %f (fs)", i*dt )
  #  plot 'solucion.dat' index i u 2:3 w lp, '' index i u 2:4 w lp
    plot 'solucion.dat' index i u 2:($3**2+$4**2) w lp title "Amplitud de Onda",
step(x) = x>0 ? .2 : 0

    pause -1}
```

Solución:

El procedimiento de este problema es equivalente al anterior, pero ahora añadimos la barrera de potencial en el intervalo deseado. Algo interesante es que observamos que hay interferencia aunque la energía del paquete de onda supera a la de la barrera.



Ocurre interferencia aunque el paquete tenga energía mayor a la barrera.



Evolución temporal del paquete de onda.

Ejercicio 7.25:

Nos piden resolver la ecuación propuesta del libro para el pozo de potencial con ciertos parámetros dados.

Código cpp:

```
//============================================================
//
// Ecuacion de Schrodinger con metodo pseudo espectral
//
//============================================================

#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 hbar = sqrt(7.6199682);
  double L = 30.0;
  double k0 = 2*M_PI/(N+1);
  double a = 1.963;
  double masa = 8.0;
  double dx    = 2*L/N;
  double dt    = 22;
  double delta_x = 1; // Ancho del paquete
  double k0momentum = sqrt(2*100)/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;
  k = new double[ N+1 ];
  x = new double[ N+1 ];
  V = new double[ N+1 ];



  // Inicializar coordenada x
  for(int i=0; i<N+1; i++)
    x[i] = -L + i*dx;

  // Inicializar k
  for(int i=0; i<(N+1)/2; i++)
    k[i] = i*k0;

  for(int i=(N+1)/2; i<N+1; i++)
    k[i] = -k[N+1-i];



  // Inicializar Potencial
  for(int i=0; i<N+1; i++){
    if ( 0<=x[i] && x[i]<=a )
      V[i] = -200.0;
    else
      V[i] = 0.0;
```

```
  }

  // Inicializar expomenciales de T y V
  for(int i=0; i<N+1; i++){
    expV[i] = exp( -I*V[i]*dt/(2*hbar) );
    expT[i] = exp( -I*hbar*k[i]*k[i]*dt/(2*masa) );
  }



  /* condiciones de frontera */
  //psi[0] = 0;
  //psi[N] = 0;


  // condiciones iniciales
  for(int i=0; i<N+1; i++)
    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);


  // ciclo principal
  for(int j=0; j<=Niter; j++){

    if ( j%outCada==0 ){
      cout << "it = " << tiempo << " / " << Niter << endl;;
      output( psi, x, tiempo, N, solucion );
    }


    // Aplicacion de los operadores
    for(int i=0; i<N+1; i++)
      phi[i] = expV[i] * psi[i];

    fourier( trans, phi, N+1 );

    for(int i=0; i<N+1; i++)
      phi[i] = expT[i] * trans[i];

    fourierInversa( psi, phi, N+1 );

    for(int i=0; i<N+1; i++)
      psi[i] = expV[i] * psi[i];


    // condiciones de frontera
    psi[0] = 0.0;
    psi[N] = 0.0;
```

```
    tiempo += dt;

  }

  return 0;
}



/*************************************************************************/



//double f( double x )
//{
  //return sqrt(2.0)*sin(5*x*M_PI);
  //return cexp(-50*pow(x-0.5,2) -I*100*x);
//}


void output( complex<double> *psi, double *x, double tiempo, int N, ostream &of )
{
  for(int i=0; i<N+1; i++)
    of << tiempo << "\t" << x[i] << "\t"  << real(psi[i]) << "\t" << imag(psi[i])
<< endl;

  of << endl << endl;
}



void fourier( complex<double> *ftrans, complex<double> *f, int n )
{
  for( int i=0; i<n+1; i++ ){
    ftrans[i] = 0.0;
    for( int j=0; j<n+1; j++ )
      ftrans[i] += f[j] * exp(-2*M_PI*I * (double)j * (double)i / (double)n);

    ftrans[i] /= sqrt(n);
  }
}


void fourierInversa( complex<double> *f, complex<double> *ftrans, int n )
{
  for( int i=0; i<n+1; i++ ){
    f[i] = 0.0;
    for( int j=0; j<n+1; j++ )
      f[i] += ftrans[j] * exp(2*M_PI*I*(double)j*(double)i/(double)n);

    f[i] /= sqrt(n);
```

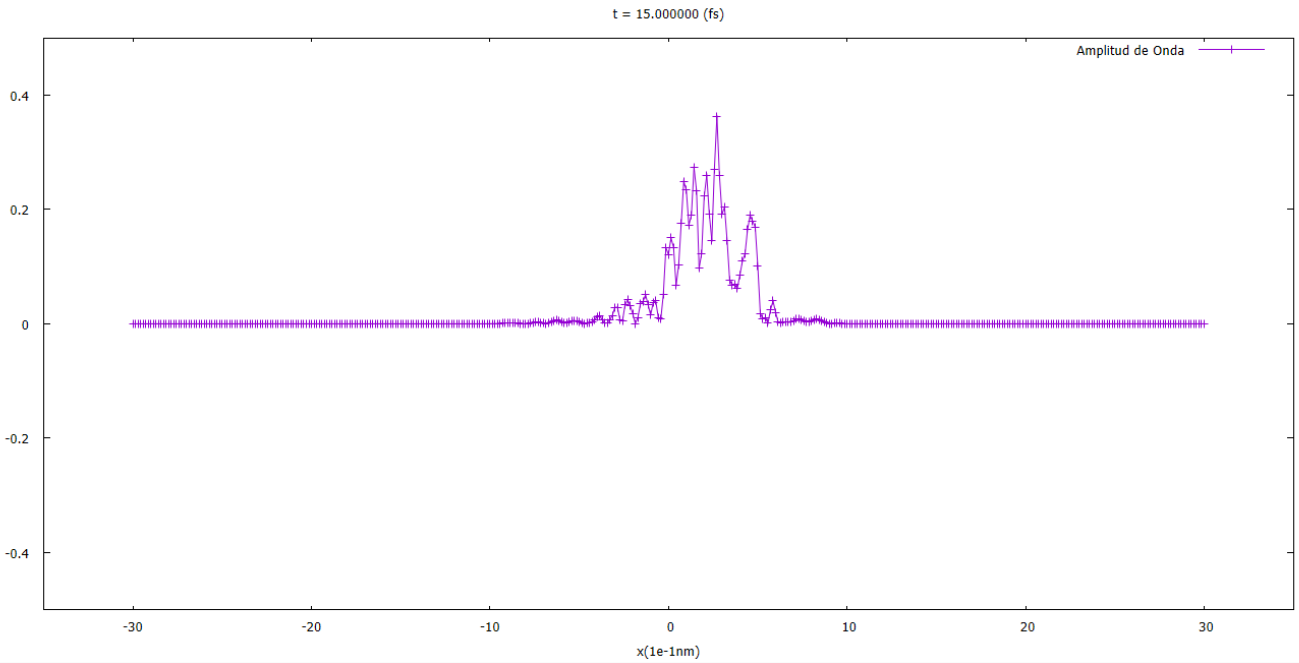
```
    }
  }
```

Código gp:

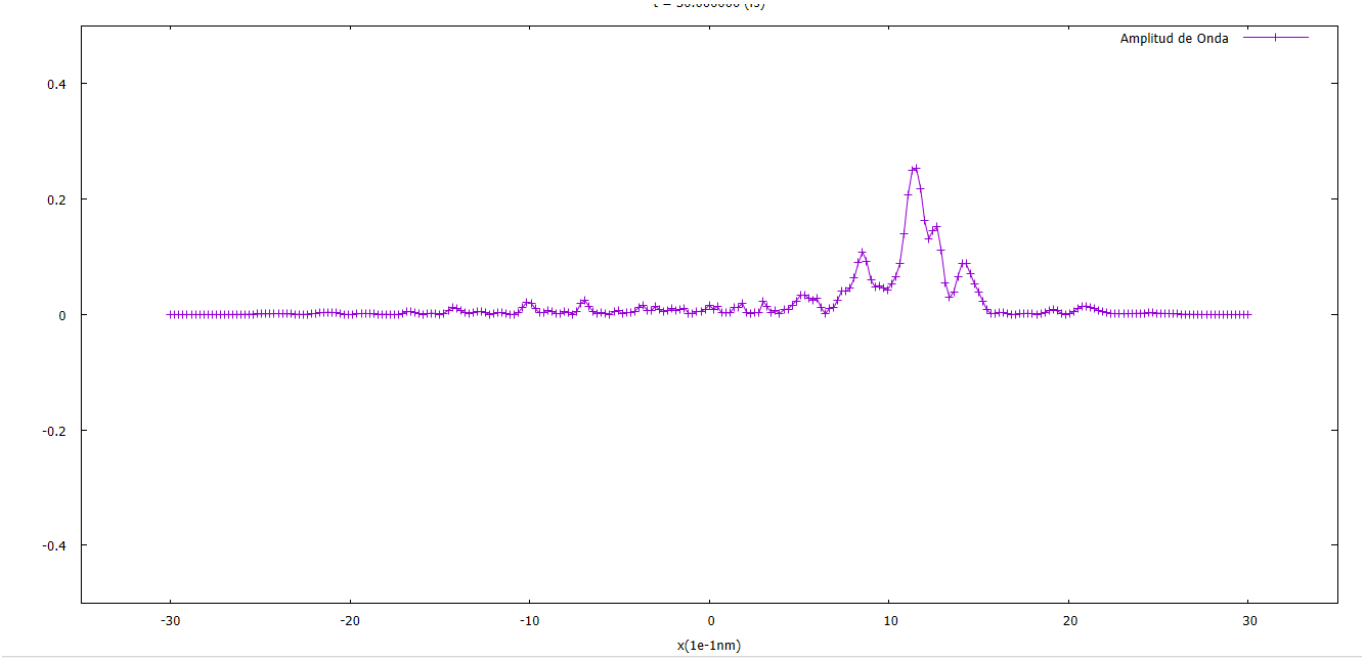
```
set yrange [-0.5:0.5]
set xrange [-35:35]
dt = 5
set xlabel "x(1e-1nm)"
do for [i=0:30] {
    set title sprintf( "t = %f (fs)", i*dt )
  #  plot 'solucion.dat' index i u 2:3 w lp, '' index i u 2:4 w lp
    plot 'solucion.dat' index i u 2:($3**2+$4**2) w lp title "Amplitud de Onda",
step(x) = x>0 ? .2 : 0

    pause -1}
```

Solución:

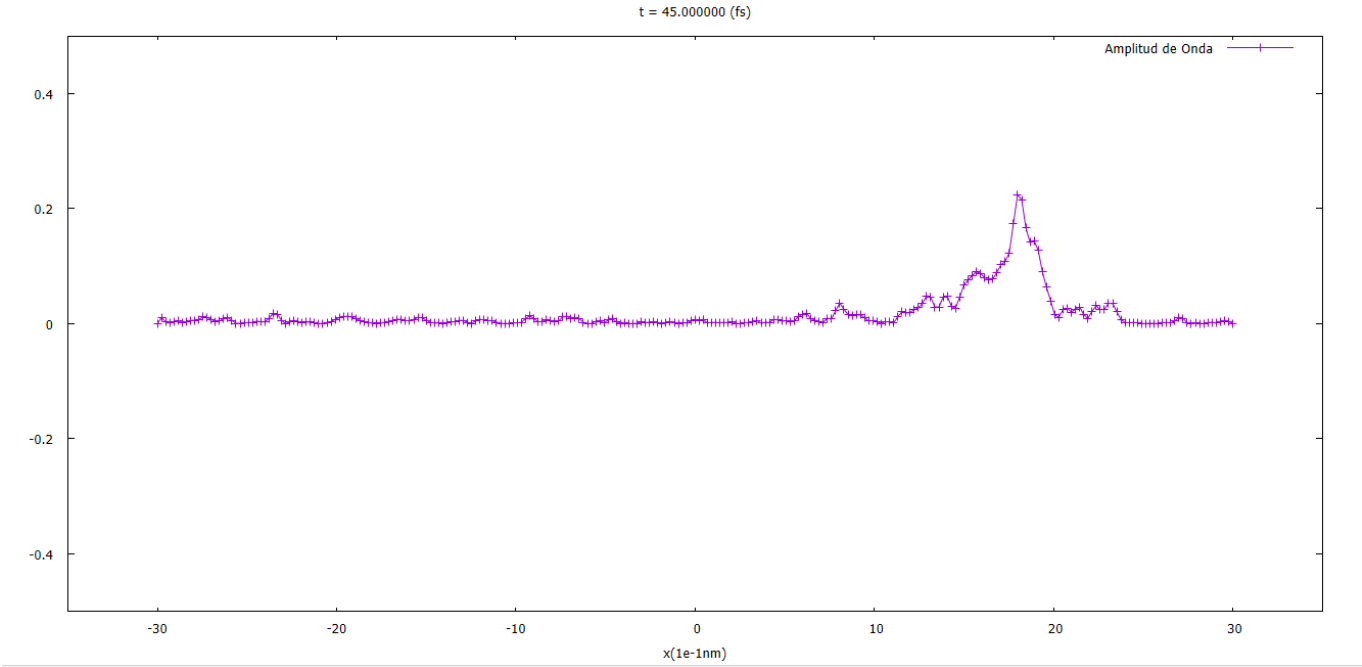
Colocamos los parámetros dados del libro para intentar reproducir el comportamiento encontrado por el artículo referenciado, notamos que si no realizamos el proceso de refinación del pozo con los parámetros pedidos obtenemos un comportamiento distinto al esperado para este fenómeno físico. Pero si realizamos una calibración de las constantes haciendo una red más fina pero no tan fina para que los puntos pasen por las fronteras del pozo, podemos reproducir el comportamiento buscado.



Paquete empezando a cruzar el pozo.



Paquete de onda sobre el pozo.



Paquete de onda finalizando de cruzar el pozo.