Tarea 4

Física Computacional

Diego Sarceño

201900109

28 de octubre de 2022

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

Ejercicio 1

Resolviendo la ecuación de onda en una dimensión para únicamente 20 iteraciones, se obtuvieron datos que no mostraban un movimiento como tal, mi hipótesis es que algo conflictuaba entre α, dt y el número de iteraciones. Entonces por prueba y error se logró generar datos para crear un video, el cual se adjunta a este pdf. Se adjunta un frame de dicho video

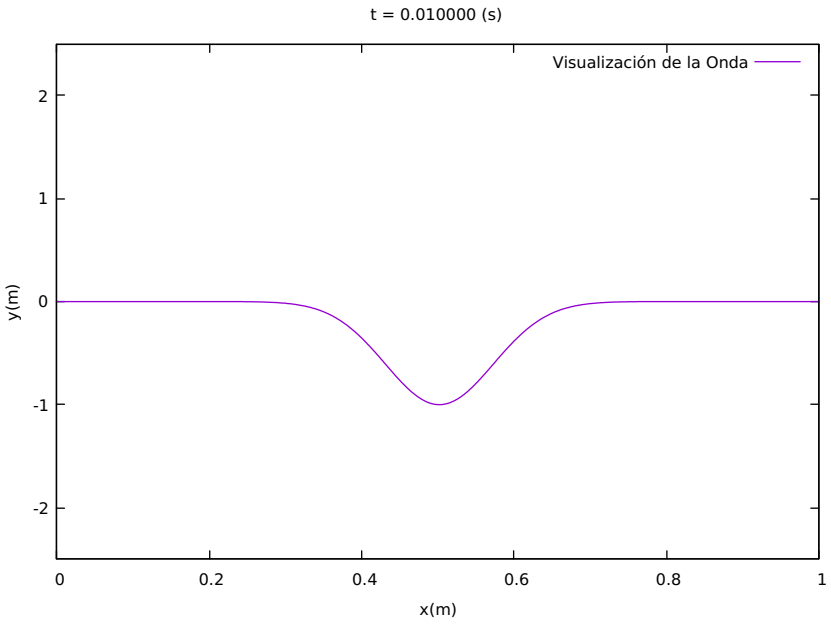

Figura 1: Frame de la solución a la ecuación de onda en una dimensión.

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

using namespace std;

double f_cond_ini( double x );
double g_cond_ini( double x );
```

```
10 double w_cond_frontera( double t, double vel );
11 double z_cond_frontera( double t, double vel );
12 void output( ostream &of, double *u, double *x, double t, int N );
13
14
15 int main()
16 {
17   int N = 500; //numero de puntos en x
18   int out_cada = 20; //output cada no. de iteraciones
19   double L = 1.0; //longitud del dominio en x
20   double dx = L/N;
21   double alfa = 0.5;
22   double vel = 2.0; // velocidad de la onda
23   double dt = alfa*dx/vel;
24   int Niter = 2000; // numero de iteraciones en el tiempo
25   double tiempo = 0.0; // lleva la cuenta del tiempo
26   ofstream outfile;
27   outfile.open( "solucion.dat", ios::out );
28
29   // variables para u
30   double *u_nueva = new double[N+1]; // u_{i,j+1}
31   double *u       = new double[N+1]; // u_{i,j}
32   double *u_vieja = new double[N+1]; // u_{i,j-1}
33   double *x       = new double[N+1]; // coordenada x
34
35
36   // coordenada x
37   for( int i=0; i<N+1; i++ )
38     x[i] = i*dx;
39
40   // condiciones iniciales u_{i0}
41   for( int i=0; i<N+1; i++ )
42     u_vieja[i] = f_cond_ini( x[i] );
43
44   // condiciones iniciales u_{i1}
45   for( int i=0; i<N+1; i++ )
46     u[i] = u_vieja[i] + g_cond_ini( x[i] ) * dt;
47
48
49   // condicion de frontera
50   u[0] = w_cond_frontera( 0.0, vel );
51   u[N] = z_cond_frontera( 0.0, vel );
52
53
54   tiempo += dt;
55
56   // ciclo principal
57   for( int j=0; j<=Niter; j++ ){
```

```
58      for( int i=1; i<N; i++ )
59        u_nueva[i] = 2.*(1.-alfa*alfa) * u[i] + alfa*alfa*(u[i-1] + u[i
              +1]) - u_vieja[i];
60
61      // condicion de frontera
62      u_nueva[0] = w_cond_frontera( tiempo + dt, vel );
63      u_nueva[N] = z_cond_frontera( tiempo + dt, vel );
64
65      // cambiar instantes de tiempo
66      for(int i=0; i<N+1; i++ ){
67        u_vieja[i] = u[i];
68        u[i]       = u_nueva[i];
69      }
70
71      tiempo += dt;
72
73      // output
74      if ( j % out_cada == 0 )
75        output( outfile, u, x, tiempo, N );
76
77    }
78
79
80
81    return 0;
82  }
83
84
85
86  void output( ostream &of, double *u, double *x, double t, int N )
87  {
88    for( int i=0; i<N+1; i++ )
89      of << t << "\t" << x[i] << "\t" << u[i] << endl;
90
91    of << endl << endl;
92  }
93
94
95
96  double f_cond_ini( double x )
97  {
98    double L = 1.0; // longitud de la cuerda
99    //return sin(4*2.*M_PI*x);
100   //return exp(-100*pow(x-L/2,2));
101   return 0.0;
102 }
103
104
```

```
105 double g_cond_ini( double x )
106 {
107   double L = 1.0; // longitud de la cuerda
108   //return 10*exp(-100*pow(x-L/2,2));
109   return 0.0;
110 }
111
112
113 double w_cond_frontera( double t, double vel )
114 {
115   return exp(-100*(0 - vel*t - 0.5)*(0 - vel*t - 0.5));
116 }
117
118
119 double z_cond_frontera( double t, double vel )
120 {
121   return exp(-100*(1 - vel*t - 0.5)*(1 - vel*t - 0.5));
122 }
```

Ejercicio 2

Al programa ya mostrado, únicamente se le agrega la condición inicial en la velocidad y el número de iteraciones a 20.

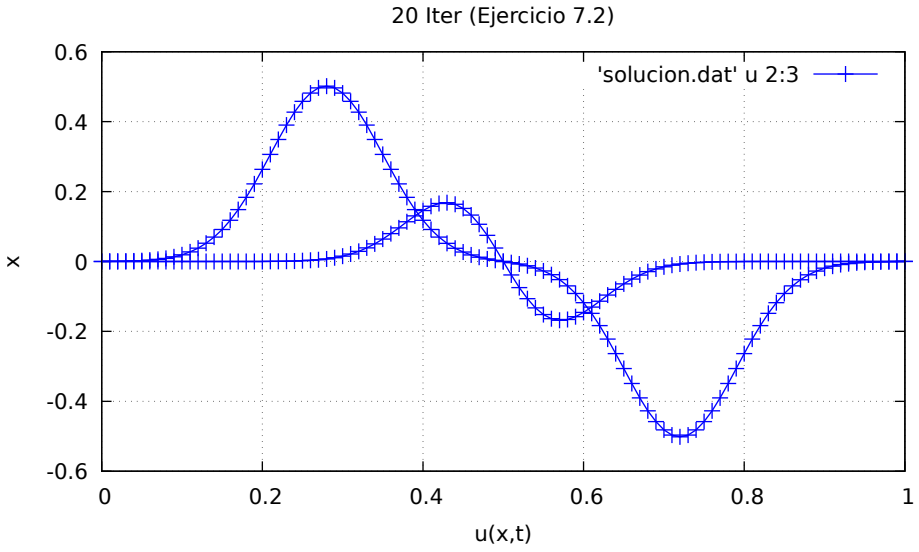

Figura 2: Los "frames" superpuestos en una gráfica.

```
1 double g_cond_ini( double x, double vel )
2 {
3   double L = 1.0; // longitud de la cuerda
4
```

```
5   return -200*vel*(x - 0.5)*exp(-100*(x - 0.5)*(x - 0.5));
6 }
```

Ejercicio 3

Tomando las constantes α y β como ± 1 se manipulan las condiciones iniciales y de frontera manteniendo el signo de la constante en todas. De modo que las funciones de condiciones de frontera e iniciales son

```
1  double f_cond_ini( double x, double vel )
2  {
3    double L = 1.0; // longitud de la cuerda
4
5    return exp(-100*(x-0.75)*(x-0.75))+exp(-100*(x-0.25)*(x-0.25)); //
       interferencia constructiva
6    //return exp(-100*(x-0.75)*(x-0.75))-exp(-100*(x-0.25)*(x-0.25)); //
       interferencia destructiva
7  }
8
9
10 double g_cond_ini( double x, double vel )
11 {
12   double L = 1.0; // longitud de la cuerda
13
14   return -200*vel*(x-0.75)*exp(-100*((x-0.75)*(x-0.75)))+200*vel*(x
       -0.25)*exp(-100*((x-0.25)*(x-0.25))); // interferencia constructiva
15   //return -200*vel*(x-0.75)*exp(-100*((x-0.75)*(x-0.75)))-200*vel*(x
       -0.25)*exp(-100*((x-0.25)*(x-0.25))); // interferencia destructiva
16 }
17
18
19 double w_cond_frontera( double t, double vel )
20 {
21   return  exp(-100*((0+vel*t)-0.75)*((0+vel*t)-0.75))+exp(-100*((0-vel*t
       )-0.25)*((0-vel*t)-0.25)); // interferencia constructiva
22   //return  exp(-100*((0+vel*t)-0.75)*((0+vel*t)-0.75))-exp(-100*((0-vel
       *t)-0.25)*((0-vel*t)-0.25)); // interferencia destructiva
23 }
24
25
26 double z_cond_frontera( double t, double vel )
27 {
28   return  exp(-100*((1+vel*t)-0.75)*((1+vel*t)-0.75))+exp(-100*((1-vel*t
       )-0.25)*((1-vel*t)-0.25)); // interferencia constructiva
29   //return  exp(-100*((1+vel*t)-0.75)*((1+vel*t)-0.75))-exp(-100*((1-vel
       *t)-0.25)*((1-vel*t)-0.25)); // interferencia destructiva
30 }
```

Con estas modificaciones se tienen las siguientes gráficas justo antes de que las ondas se encuentren.

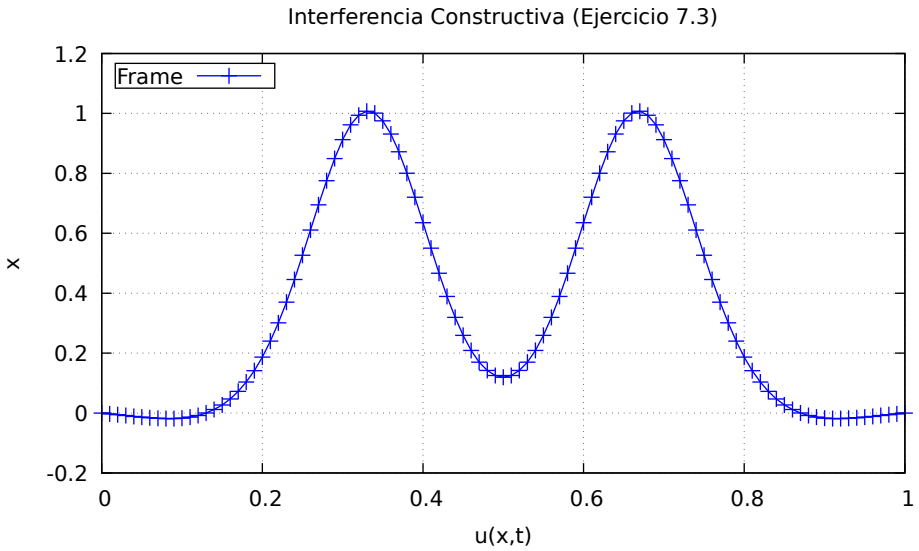

Figura 3: Interferencia constructiva.

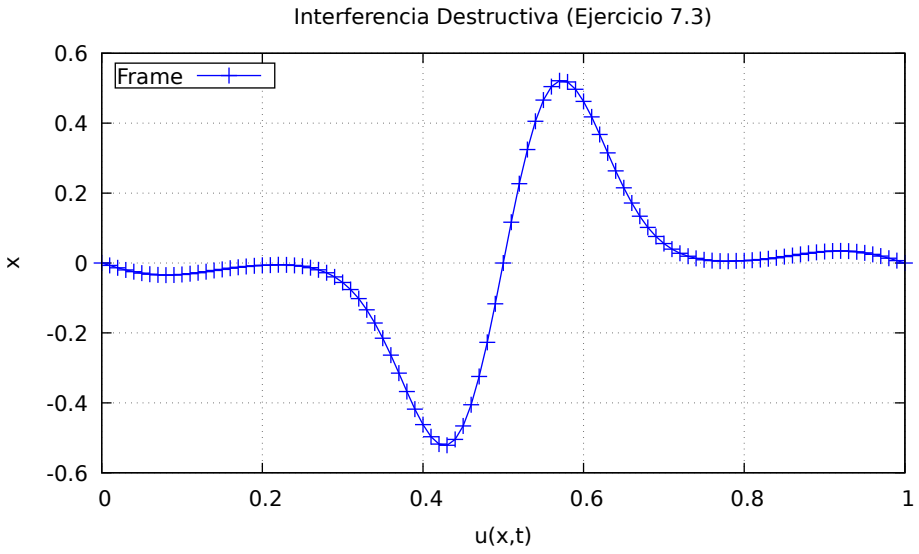

Figura 4: Interferencia destructiva.