Tarea 2

Física Computacional

Diego Sarceño

201900109

11 de octubre de 2022

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

Problema 1

Tomando la ecuación de movimiento para un objeto en caida libre con resistencia cuadrática

$$\frac{\mathrm{d}p}{\mathrm{d}t} = mg - kv^2,$$

para un coeficiente de resistencia $k = 10^{-4}$ y una masa de $10g$, se utilizó el método de runge kutta para resolver la ecuación. Se tomó un paso de $h = 0.1$ y se comparó con la solución analítica para un movimiento sin resistencia del aire.

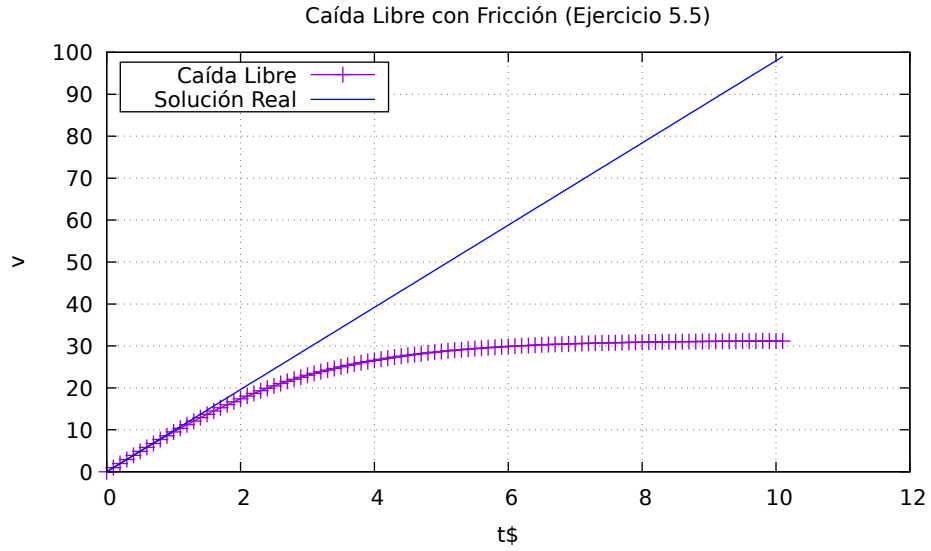

Figura 1: Solución utilizando el método de Runge Kutta comparada con la solución real ignorando la resistencia del aire.

En la figura 1 es clara la implicación de la resistencia del aire, puesto que se tiene una asíntota en aproximadamente $32m/s$, la cual es conocida como la velocidad terminal. Cosa que para el movimiento sin resistencia del aire, no se tiene.

```
1 // Librerias
2 #include <iostream>
3 #include <fstream>
4 #include <cmath>
```

```
5  #include <iomanip>
6
7  using namespace std;
8
9  void RK4(const double *y,
10         const int n_ec,
11         const double t,
12         const double h,
13         double *y_new,
14           void (*derivada)(const double *, const double, double *));
15 void salidaSolucion(const double t, const double *y, const int N);
16 void oneD_ProjectileMotion(const double *y, const double t, double *dydt
       );
17
18
19 int main(){
20   // Datos iniciales
21   const double t0 = 0.0;
22   const double y0 = 0.0;
23   const double h = 0.1;
24   const int N = 10/h; // numero de iteraciones
25   const int out_frec = 1; // frecuencia de output
26   const int n_ec = 1; // numero de ecuaciones
27
28   // reservamos espacio
29   double *y = new double[n_ec];
30   double *y_nueva = new double[n_ec];
31
32   // condiciones iniciales
33   y[0] = y0;
34
35   // puntero a la funcion derivada
36   void (*derivada)(const double *, const double, double *);
37   derivada = oneD_ProjectileMotion;
38
39   // y_nueva
40   y_nueva[0] = 0.0;
41
42   double t = t0;
43
44   cout << t << "\t" << y[0] << endl;
45
46   for(int i = 0; i <= N; i++){
47     RK4(y, n_ec, t, h, y_nueva, derivada);
48
49     y = y_nueva;
50     t += h;
51
```

```
52      if (i % out_frec == 0){
53        cout << t << "\t" << y[0] << endl;
54      } // END IF
55    } // END FOR
56
57
58    return 0;
59  } // END main
60
61
62
63  void salidaSolucion(const double t, const double *y, const int n){
64    cout << fixed << setprecision(3) << t;
65
66    for(int i = 0; i < n; i++)
67      cout << scientific << setprecision(9) << "\t" << y[i];
68
69    cout << endl;
70  } // END salidaSolucion
71
72
73
74  void RK4(const double *y,
75                       const int n_ec,
76                       const double t,
77                       const double h,
78                       double *y_new,
79                       void (*derivada)(const double *, const double,
                            double *)){
80    double *k0 = new double[ n_ec ];
81    double *k1 = new double[ n_ec ];
82    double *k2 = new double[ n_ec ];
83    double *k3 = new double[ n_ec ];
84    double *z  = new double[ n_ec ];
85
86    (*derivada)( y, t, k0 );
87
88    for( int i=0; i<n_ec; i++ )
89      z[i] = y[i] + 0.5*k0[i]*h;
90
91    (*derivada)( z, t+0.5*h, k1 );
92
93    for( int i=0; i<n_ec; i++ )
94      z[i] = y[i] + 0.5*k1[i]*h;
95
96    (*derivada)( z, t+0.5*h, k2 );
97
98    for( int i=0; i<n_ec; i++ )
```

```
99      z[i] = y[i] + k2[i]*h;
100
101   (*derivada)( z, t+h, k3 );
102
103   for( int i=0; i<n_ec; i++ )
104    y_new[i] = y[i] + h/6.0 * ( k0[i] + 2*k1[i] + 2*k2[i] + k3[i] );
105
106   delete[] k0;
107   delete[] k1;
108   delete[] k2;
109   delete[] k3;
110   delete[] z;
111 } // END RK4
112
113 void oneD_ProjectileMotion(const double *y, const double t, double *dydt
      ){
114   const double g = 9.8;
115   const double m = 1e-2;
116   const double k = 1e-4;
117
118   // ecuaciones de movimiento
119   dydt[0] = g - k*y[0]*y[0]/m;
120 } // END oneD_ProjectileMotion
```

Problema 2

En las carpetas mencionadas en *Github* se encuentrar las librerías externas utilizadas "rkqs.hpp" y "odeint.hpp". Se tuvo 379 iteraciones buenas y rechazó 73 iteraciones.

```
1  // Librerias
2  #include <iostream>
3  #include <cmath>
4  #include <iomanip>
5  #include <fstream>
6
7  // Librerias externas
8  #include "rkqs.hpp"
9  #include "odeint.hpp"
10
11 using namespace std;
12
13 void caidaLibreFriccion( double t, double *y, double *dydt );
14
15 int main(){
16   int nvar, nok, nbad;
17   double t1, t2, eps, h, hmin, *ystart;
18
19
```

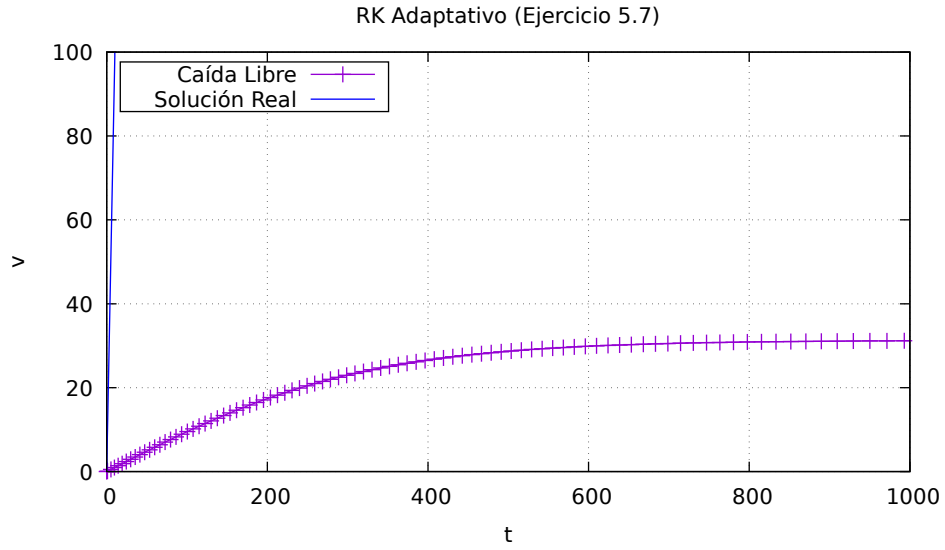

Figura 2: Gráfica para un objeto en caida libre con el método Runge Kutta adaptativo.

```
20  /* memory space for variables */
21  // numero de ecuaciones
22  nvar = 2;
23
24  // valor inicial de cada variable
25  ystart = new double[nvar];
26
27  /* other variables initialization */
28  // tolerancia (error)
29  eps  = 1e-10;
30  h    = 1;
31  hmin = 1e-5;
32  nok  = 0;
33  nbad = 0;
34
35
36
37  /* initial condition */
38  ystart[0]  = 1.0;
39  ystart[1]  = 0.0;
40
41  // tiempo inicial
42  t1 =  0.0;
43
44  // tiempo final
45  t2 =  2000.0;
46
47
```

```
48   odeint(ystart, nvar, t1, t2, eps, h, hmin, &nok, &nbad, &
        caidaLibreFriccion, &rkqs);
49
50   cout << "nok = " << nok <<"\t nbad = " << nbad << endl;
51
52   return 0;
53 } // END main
54
55
56 void caidaLibreFriccion( double t, double *y, double *dydt ){
57   const double g = 9.8;
58   const double m = 0.01;
59   const double k = .0001;
60
61
62   dydt[0] = y[1];
63   dydt[1] = g - (k/m)*y[1]*y[1];
64 } // END oneD_ProjectileMotion
```

Problema 3

Se resuelve el oscilador de Ven der Pol

$$\ddot{x} = -x - \varepsilon(x^2 - 1)\dot{x}.$$

Se utiliza el método de Runge Kutta adaptativo. Tomando un $\varepsilon = 0$ no da una solución; sin embargo, si se utiliza un $h = 10^{-5}$, se tiene

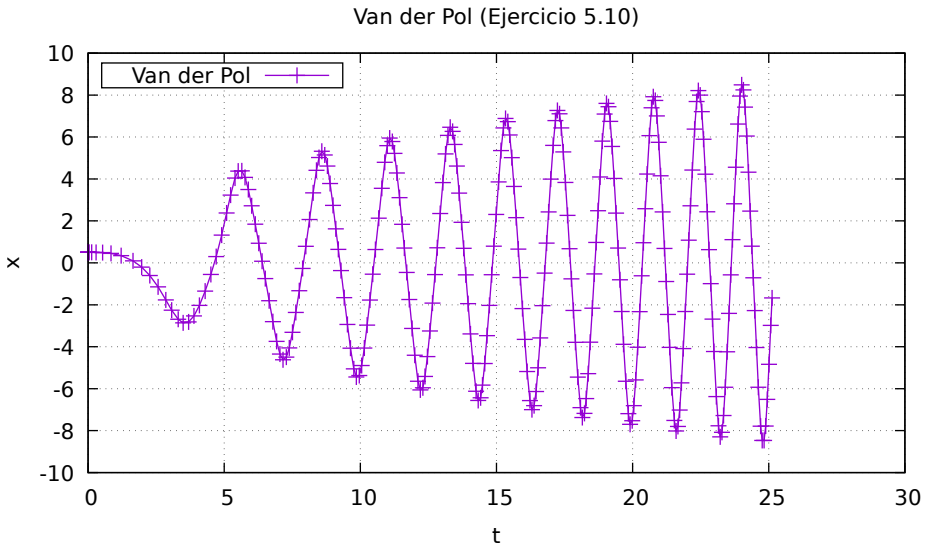

Figura 3: Solución para x del oscilador Van der Pol.

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

#include "rkqs.hpp"
#include "odeint.hpp"

using namespace std;


void VanderPol(double t, double *y, double *dydt);




int main(){
  int nvar, nok, nbad;
  double t1, t2, eps, h, hmin, *ystart;


  /* memory space for variables */
  // numero de ecuaciones
  nvar = 3;

  // valor inicial de cada variable
  ystart = new double[nvar];

  /* other variables initialization */
  // tolerancia (error)
  eps  = 1e-5;
  h    = 0.001;
  hmin = 1e-5;
  nok  = 0;
  nbad = 0;



  /* initial condition */
  ystart[0] = 0.5;
  ystart[1] = 0.0;
  ystart[2] = 0.0;

  // tiempo inicial
  t1 =  0.0;

  // tiempo final
```

```
  t2 =  25.12;


  odeint(ystart, nvar, t1, t2, eps, h, hmin, &nok, &nbad, &VanderPol, &
     rkqs);

  cout << "nok = " << nok <<"\t nbad = " << nbad << endl;

  return 0;
} // END main




void VanderPol(double t, double *y, double *dydt){
  dydt[0] = y[1];
  dydt[1] = y[2];
  dydt[2] = -y[0] - y[1]*(y[0]*y[0]-1);
} // END VanderPol
```

Problema 4

Teniendo el sistema del péndulo simple

$$\ddot{\theta} = -\frac{g}{l}\sin\theta,$$

el cual se divide en dos ecuaciones diferenciales

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = -\frac{g}{l}\sin\theta \end{cases}.$$

Se encuentra la solución al mismo para ambas variables dependientes θ y $\dot{\theta}$. Se graficaron las mismas contra el tiempo y se graficó su diagrama de fase para las siguientes energías $E = 0.25$, $E = 0.5$, $E = 0.75$, $E = 1$, $E = 1.25$ y $E = 1.5$.

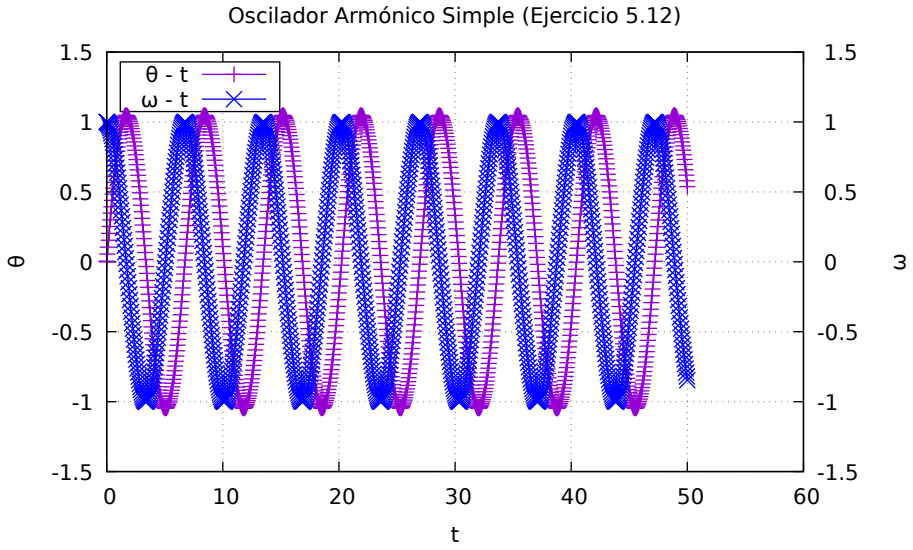

Figura 4: Gráfica de la posición angular y la velocidad angular.

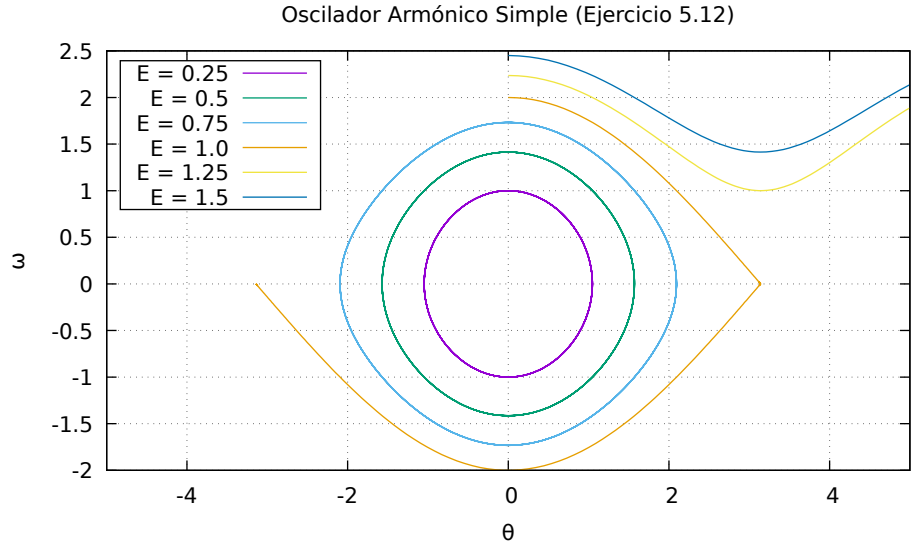

Figura 5: Gráfica de los diagramas de fase para las energías mostradas. Es claro que para la energía de $E = 1$, la trayectoria ya no es cerrada, esto implica que el péndulo no se comporta de la mejor forma y que la energía no se conserva, además de que salimos del rango en el que la aproximación de ángulos pequeños no es válida.

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

```
7  using namespace std;
8
9
10 void RK4(const double *y,
11                      const int n_ec,
12                      const double t,
13                      const double h,
14                      double *y_new,
15                      void (*derivada)(const double *, const double,
                           double *));
16 void simple_oscilator(const double *y, const double t, double *dydt);
17
18
19
20
21 int main(){
22   const double t0 = 0.0;
23   const double y0 = 0.0;
24   const double y1 = sqrt(6);
25   const double h = 0.005;
26   const int N = 10000;
27   const int out_frec = 10;
28   const int n_ec = 2;
29
30   double t = t0;
31
32   // espacio reservado
33   double *y = new double[n_ec];
34   double *y_nueva = new double[n_ec];
35
36   void (*derivada)(const double *, const double, double *);
37   derivada = simple_oscilator;
38
39   for(int i = 0; i < n_ec; i++) y_nueva[i] = 0.0;
40
41   y[0] = y0;
42   y[1] = y1;
43
44   cout << t << "\t" << y[0] << "\t" << y[1] << endl;
45
46   for(int i = 0; i <= N; i++){
47     RK4(y, n_ec, t, h, y_nueva, derivada);
48
49     y = y_nueva;
50     t += h;
51
52     if (i % out_frec == 0){
53       cout << t << "\t" << y[0] << "\t" << y[1] << endl;
```

```
    } // END IF
  } // END for

  return 0;
} // END main




void RK4(const double *y,
                      const int n_ec,
                      const double t,
                      const double h,
                      double *y_new,
                      void (*derivada)(const double *, const double,
                         double *)){
  double *k0 = new double[ n_ec ];
  double *k1 = new double[ n_ec ];
  double *k2 = new double[ n_ec ];
  double *k3 = new double[ n_ec ];
  double *z  = new double[ n_ec ];

  (*derivada)( y, t, k0 );

  for( int i=0; i<n_ec; i++ )
    z[i] = y[i] + 0.5*k0[i]*h;

  (*derivada)( z, t+0.5*h, k1 );

  for( int i=0; i<n_ec; i++ )
    z[i] = y[i] + 0.5*k1[i]*h;

  (*derivada)( z, t+0.5*h, k2 );

  for( int i=0; i<n_ec; i++ )
    z[i] = y[i] + k2[i]*h;

  (*derivada)( z, t+h, k3 );

  for( int i=0; i<n_ec; i++ )
   y_new[i] = y[i] + h/6.0 * ( k0[i] + 2*k1[i] + 2*k2[i] + k3[i] );

  delete[] k0;
  delete[] k1;
  delete[] k2;
  delete[] k3;
```

```
  delete[] z;
} // END RK4


void simple_oscilator(const double *y, const double t, double *dydt){
  const double l = 1;
  const double g = 1;

  dydt[0] = y[1];
  dydt[1] = -(g/l)*sin(y[0]);
} // END simple_oscilator
```

Problema 5

Para generar el video de la animación se utilizó *gnuplot* para crear la todos los frames y se utilizó la librería **ffmpeg** para fusionar todos los frames en un archivo *.mp4*, el resultado esta adjunto a este archivo en el espacio asignado en classroom.

```
# En el directorio 'frames' donde estan las imagenes, ejecutar el siguiente
# comando, esto generara la animacion en un archivo .mp4
# ffmpeg -framerate 30 -pattern_type glob -i '*.png' -c:v libx264 -pix_fmt yuv420p out.mp4

# PROGRAM
set term pngcairo size 800,600 enhanced font "Verdana,12"


set xrange [-1:1]
set yrange [-1:1]
set xlabel 'x (m)'
set ylabel 'y (m)'


set size ratio -1

# longitud del pendulo
lon=1.0;

do for [i=0:1000] {
    set output sprintf( "frames/frame-%04d.png", i )
    set title sprintf( "tiempo = %f", i*0.02 )

    plot 'data2' every ::i::i u (x1=lon*sin($2)):(y1=-lon*cos($2)) w p lc 5 lw 5 t '', \
         '' every ::i::i u (0.0):(0.0):(x1):(y1) w vectors nohead t ''

    print i
```

```
28 }
```