Informe de la primera sesión de prácticas

Jose Antonio Lorencio Abril

1 Introducción

En las clases anteriores vimos cómo utilizar el método de paso fijo de Euler para modelar el problema del movimiento parabólico de un objeto bajo un campo gravitatorio.

Ahora, vamos a usar este mismo método para modelar el problema del SHO (Simple Harmonic Oscillator), en el que podemos distinguir tres casos, cuyas ecuaciones se obtienen con base en la Ley de Hooke ($F(x) = -k(x-l)$):

- SHO sin rozamiento:
$$mx''(t) = -k(x(t) - l), \quad k > 0$$
- SHO con rozamiento:
$$mx''(t) = -k(x(t) - l) - b \cdot x'(t), \quad b > 0$$
- SHO forzado:
$$mx''(t) = -k(x(t) - l) - b \cdot x'(t) + f(t)$$
en nuestro caso, usaremos
$$f(t) = A \cdot sen(wt)$$

2 Objetivos

2.1 Aplicar el método de Euler para resolver la ecuación para el SHO forzado en el caso unidimensional

Establecemos los valores

$$m = 1,\ b = 0.3,\ k = 1.5,\ A = 0.4,\ w = \{1.3, 2.4, 3.5\},\ l = 0.5$$

Y, usando los módulos que habíamos definido para resolver el problema del tiro parabólico, definimos la clase 'SHOforzado.java', que implementa la interfaz 'InitialValueProblem'.

```
public class SHOforzado implements InitialValueProblem {
        private static double m=1, k=1.5, l=0.7;
        private static double[] w = {1.3, 2.4, 3.5};
        private double b,A;

        public SHOforzado(double bb, double AA) {
                b=bb;
                A=AA;
        }

        @Override
        public double getInitialTime() {
                return 0;
        }

        @Override
        public double[] getInitialState() {
                return new double[] {0.5,0};
        }

        public double f(double t, int i) {
                return A*Math.sin(w[i]*t);
        }

        //Definimos la derivada:
        // x[0]'=x[1]
        // x[1]'={lo que da la fórmula}
        @Override
        public double[] getDerivative(double time, double[] state) {
                return new double[] {state[1],
                -k/m*(state[0]-l)-b/m*state[1]+f(time, 2)/m};
        }
```

Definimos los métodos de la forma esperable y definimos nuestra función $f(t)$, a la que añadimos un índice con el que poder variar el valor de w.

Y en el main hacemos:

```
public static void main(String[] args) {
        double b=0.3, A=0.4;
        SHOforzado problem = new SHOforzado(b, A);
        FixedStepMethod method =
                new FixedStepEulerMethod(problem, 0.01);

        if (false) method.solve(15);
    else {
        double time = method.getSolution().getLastPoint().getTime();
        while (time<40) time = method.step().getTime();
    }
```

Vemos que la solución es la misma que en el caso del tiro parabólico, a excepción de la condición de parada, que queda en función del tiempo, pues esto es necesario para el segundo objetivo de la práctica.

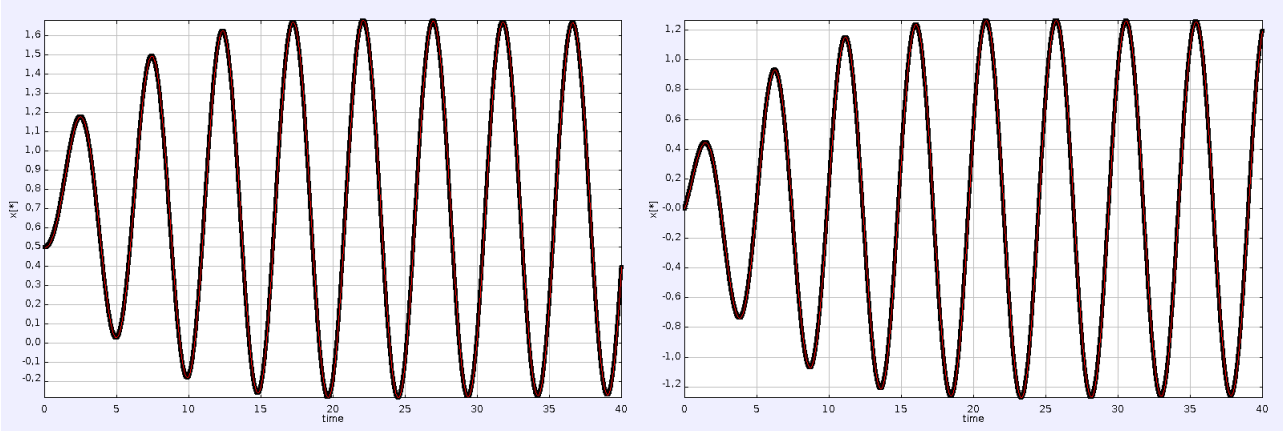
Los comentarios a este respecto los haré en el objetivo 2, pues van de la mano.

2.2 Dibujar la evolución temporal de x, v_x, $t \in [0, 40]$.

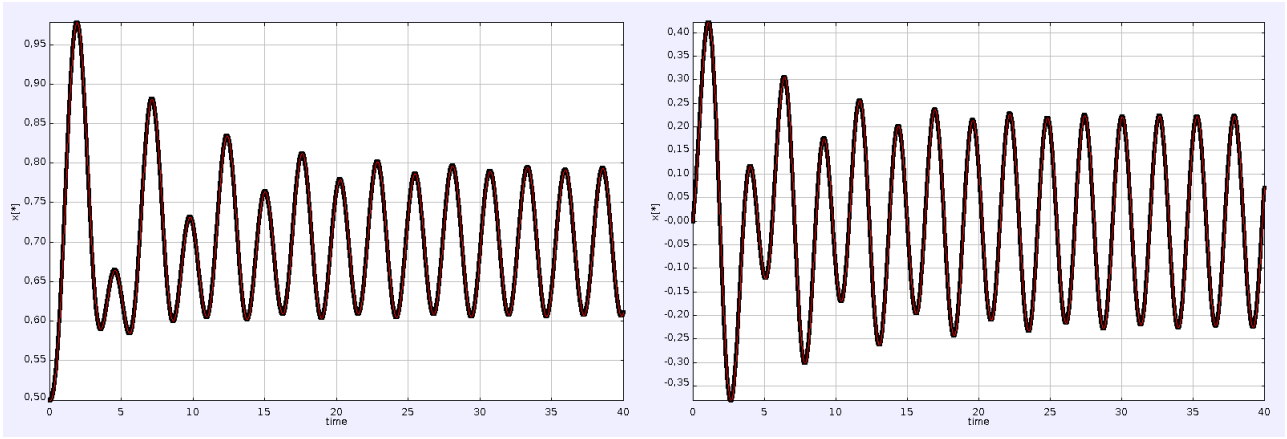
```
//Mostramos la posición respecto el tiempo
DisplaySolution.timePlot(method.getSolution(), new int[] {0});

//Mostramos la velocidad respecto el tiempo
DisplaySolution.timePlot(method.getSolution(), new int[] {1});
```

Y obtenemos las siguientes gráficas.

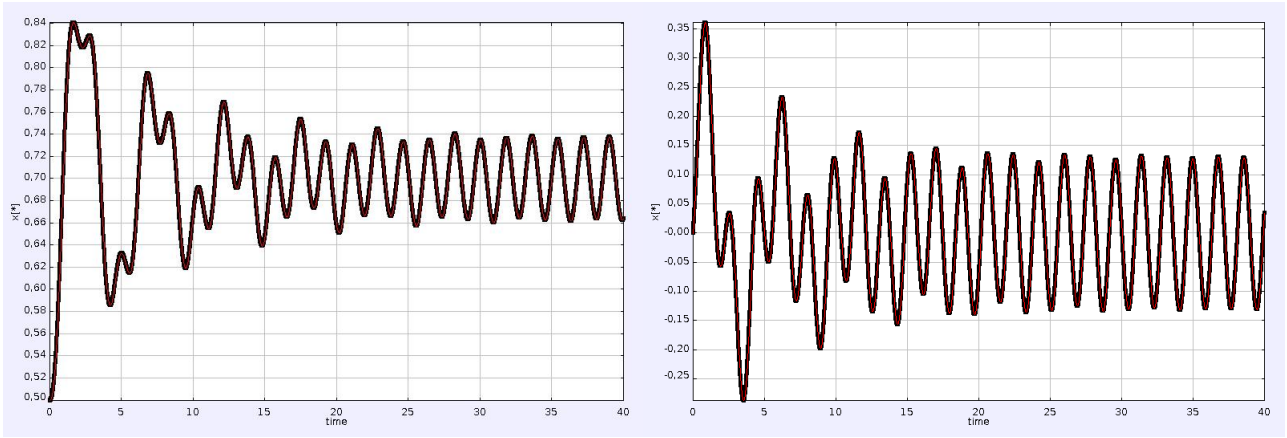
Para $w = 1.3$ (posición y velocidad, respectivamente):



Para $w = 2.5$:



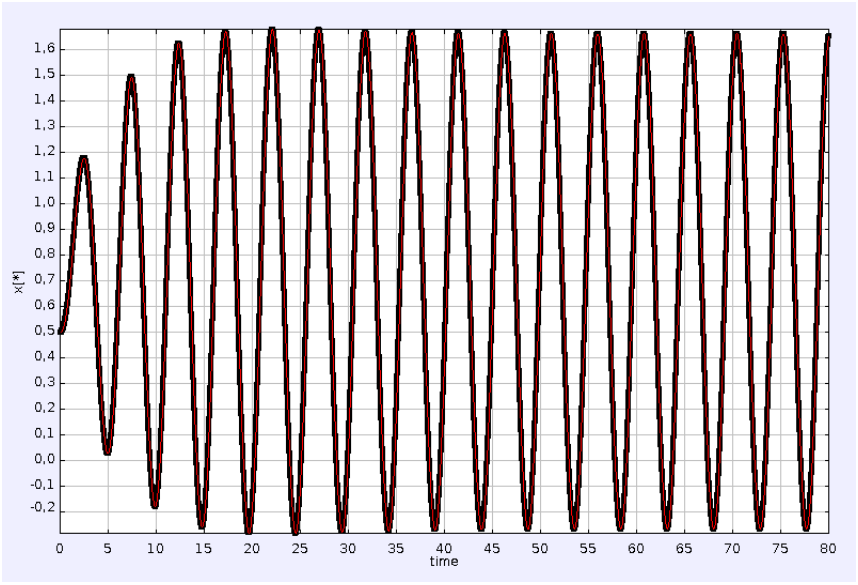
Para $w = 3.5$:



Podemos observar el comportamiento de la masa en el extremo del muelle en el caso forzado. Vemos como comienza con un movimiento un poco caótico, para ir convergiendo a una solución estable (aparentemente), en la que oscila alrededor de $l = 0.7$. Cuanto más aumentamos el valor de w, más cerca del punto de equilibrio oscila.

Esta conclusión no la deduje bien en el primer envío, pues el comportamiento inicial caótico me hacía pensar que en el primer caso, la solución era un oscilamiento creciente.

Para comprobar todo esto, podemos ver que pasa si aumentamos el tiempo de simulación, digamos a $80s$.



Y aquí se ve más claramente que el comportamiento es el descrito anteriormente.

2.3 Validación del método

Para validar el método, seguimos una lógica similar a la que utilizamos para el tiro parabólico: tomamos un problema similar del que sabemos calcular la solución analítica y comparamos nuestro método con esta solución.

Vamos a tomar el problema del SHO sin rozamiento:

$$x''(t) = -\frac{k}{m}(x(t) - l)$$

La solución general para $a(t)\,x'' + b(t)\,x' + c(t)\,x = g(t)$ se puede escribir como

$$x = x_h + x_p$$

donde x_h es solución de la ecuación homogénea y x_p es una solución particular de la no homogénea.

Hacemos, entonces

$$x'' + \frac{k}{m}x = 0$$

De donde obtenemos la ecuación característica

$$r^2 + \frac{k}{m} = 0 \implies r = \sqrt{\frac{-k}{m}} = \pm i\sqrt{\frac{k}{m}}$$

Luego

$$x_d(t) = c_1 cos\sqrt{\frac{k}{m}}t + c_2 sen\sqrt{\frac{k}{m}}t$$

Y para obtener $x_p(t)$, hacemos variación de las constantes:

$$x' = c_1' cos - c_1 r sen + c_2' sen + c_2 r cos$$

y le imponemos que $c_1'cos + c_2'sen = 0$. Entonces

$$x' = r\left(c_2 cos - c_1 sen\right) \implies x'' = r\left(c_2' cos - c_2 r sen - c_1' sen - c_1 r cos\right)$$

Luego, queda

$$-r^2\left(c_1 cos + c_2 sen - l\right) = r\left(c_2' cos - c_2 r sen - c_1' sen - c_1 r cos\right)$$

$$rl = c_2' cosrt - c_1' senrt$$

Tenemos, entonces que

$$c_1' = -c_2'\frac{senrt}{cosrt} \implies rl = c_2'\left(cosrt + \frac{sen^2 rt}{cosrt}\right) = c_2'\frac{cos^2 + sen^2}{cos} = \frac{c_2'}{cos} \implies c_2' = rl \cdot cosrt$$

$$\implies c_2 = lsenrt + k_2$$

Y

$$c_1' = -rlsenrt \implies c_1 = lcosrt + k_1$$

Y queda

$$x\left(t\right) = \left(lcosrt + k_1\right)cosrt + \left(lsenrt + k_2\right)senrt = l + k_1 cosrt + k_2 senrt$$

Queremos que $x\left(0\right) = x_0,\ x'\left(0\right) = 0$, luego

$$x = l + k_1 \implies k_1 = x_0 - l$$

Y,

$$0 = -\left(x_0 - l\right)rsenrt + k_2 rcosrt$$

Por lo que $k_2 = 0$ sirve.

O sea, que nuestra solución es

$$x\left(t\right) = l + \left(x_0 - l\right)cos\left(\sqrt{\frac{k}{m}}t\right)$$

Podemos ahora comprobar nuestra metolodía. Para ello, creamos un nuevo problema de esta clase, diciéndole que el rozamiento es nulo ($b = 0$) y que no vamos a usar la función para forzar ($A = 0$).

En clase no me salió la misma solución de la ED, por lo que voy a modificar un poco el código enviado por la mañana.

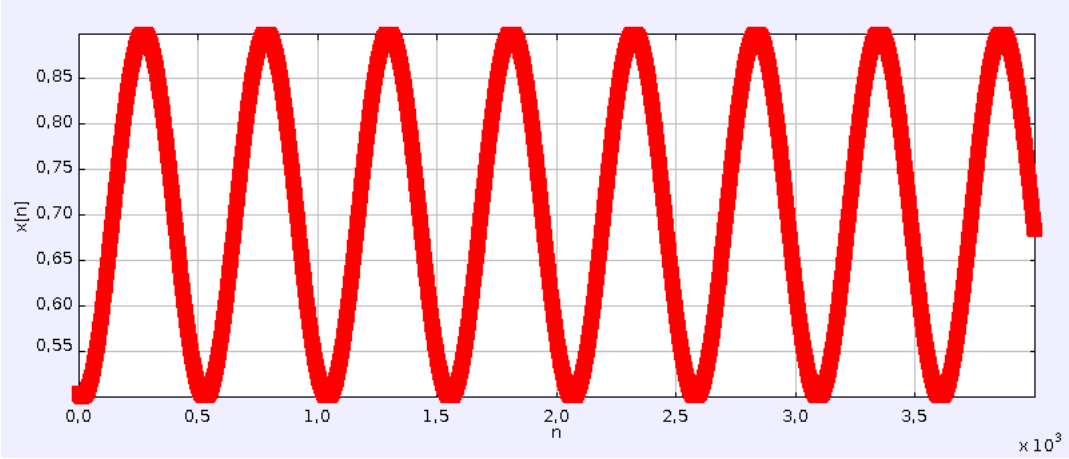
Además, ahora se me ha ocurrido plotear no solo el error, si no las dos gráficas, de forma que veamos cómo se manifiesta ese error.

He añadido, también, el cálculo de una media del error.

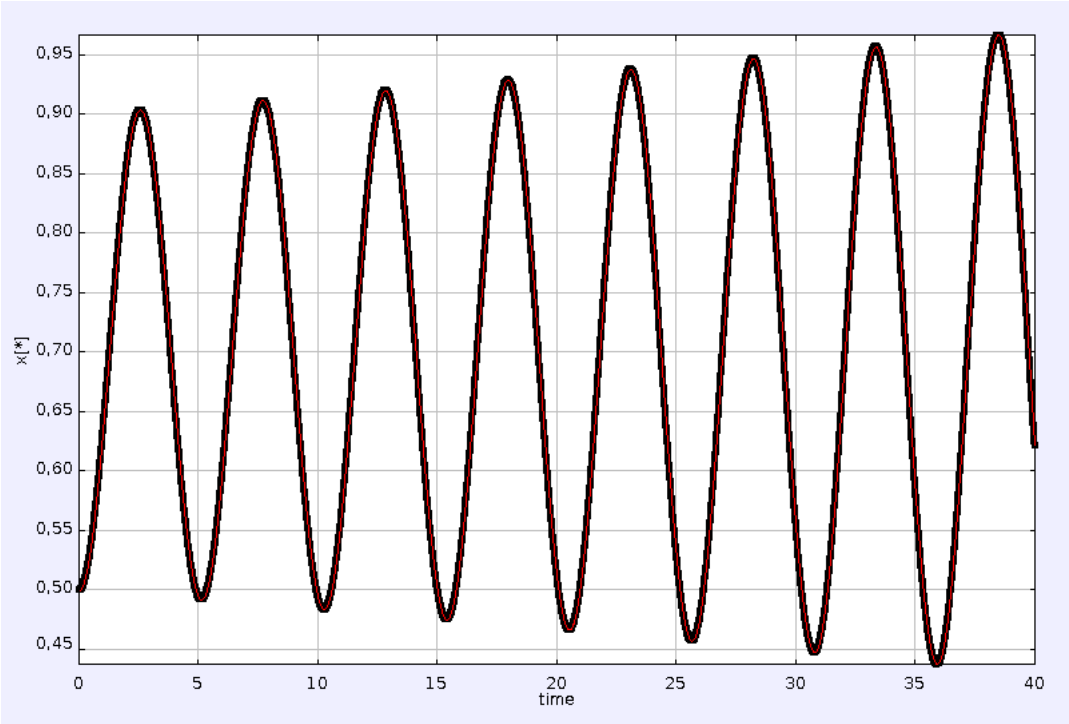
```
b=0;
A=0;

SHOforzado problem2 = new SHOforzado(b,A);

FixedStepMethod method2 =
        new FixedStepEulerMethod(problem2, 0.01);

double paso=0.01;
int n = (int) Math.round(40/paso);
double[] x = new double[n];

if (false) method2.solve(15);
else {

        x[0]=problem2.getInitialState()[0];
        double x0=x[0];

        int i=0;

        double media=0;

        double time = method2.getSolution().getLastPoint().getTime();
    while (time<40) {
        method2.step();
        //Obtenemos la solución analítica de la EDO
                // x=l+(x0-l)*sen(\sqrt(k/m)*t)
        //le restamos lo que obtenemos con el método
        x[i]=Math.abs(l+(x0-l)*Math.cos(Math.sqrt(k/m)*time)-
                    method2.getSolution().getLastPoint().getState(0))
                time += paso;
                media += y[i];
                i++;
    }
}
DisplaySequence.plot(x);
DisplaySolution.timePlot(method2.getSolution(), new int[] {0});

System.out.println("A lo largo de 40 segundos obtenemos
                un error de "+ x[n-1]+
                "de un total de "+
                method2.getSolution().getLastPoint().getState(0));

media=media/n;
System.out.println("La media del error es:"+media);
```

Obtenemos las siguientes gráficas.

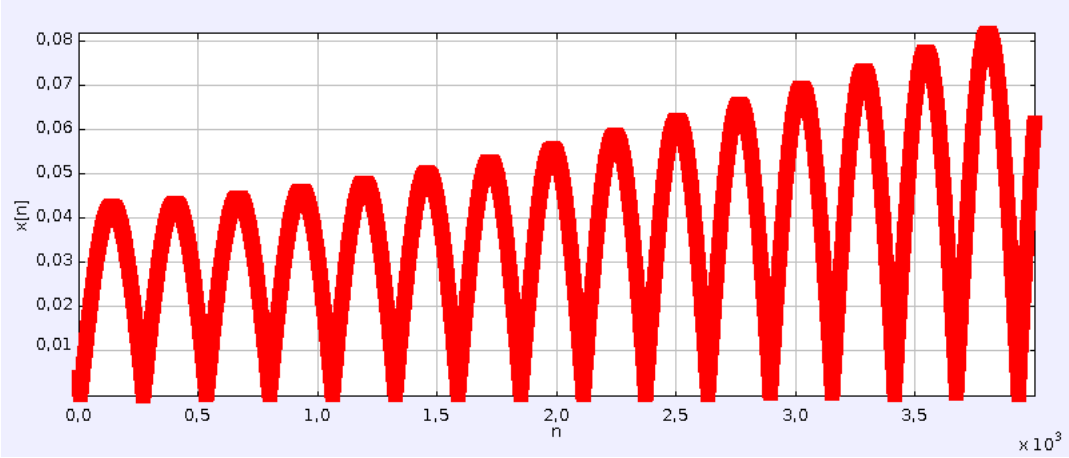
Para la solución exacta:



La aproximada:



Y el error en valor absoluto:



En las dos primeras gráficas podemos observar que la solución aproximada va abriendo las oscilaciones, mientras que la solución real permanece estable en una amplitud determinada oscilando alrededor del punto de equilibrio.

En la tercera gráfica observamos el error en valor absoluto, que vemos que va oscilando. Es pequeño cuando estamos cerca del punto de equilibrio y crece tanto al alejarnos del punto de equilibrio, como con el paso del tiempo.

Esto nos dice varias cosas sobre nuestro método:

- Es probable que sea bueno para buscar momentos temporales como cuándo hay un pico o un valle, o cuando pasamos por el punto de equilibrio.
- Es probable que no sea tan bueno para calcular la posición exacta del muelle en un momento concreto, pues va a depender de si en ese momento nos encontramos cerca del punto de equilibrio (buena aproximación) o no (no tan buena).
- El error podremos acotarlo en un intervalo, pero, dado que es creciente, es improbable que seamos capaces de acotarlo en un intervalo arbitrario.
- Conclusión: la aproximación es decente, pero debemos tener cuidado.

2.4 Elegir el paso h para que el error final sea del orden de 10^{-2}.

En el apartado anterior, la media del error obtenida fue 0.0368.

Como el método de Euler es de orden lineal, es esperable que dividiendo el paso por 10, obtengamos una media de error del orden buscado.

Un problema que surge respecto a la forma en que hice el apartado anterior, es que parece ser que Java no acepta arrays de más de 4000 posiciones, por lo que voy a hacerlo por partes. (Esto realmente me parece muy raro, 4000 no es un tamaño muy grande, es probable que sea un error tonto en el que no estoy cayendo. De todas formas, la solución por partes funciona).

```
b=0;
A=0;

SHOforzado problem3 = new SHOforzado(b,A);

FixedStepMethod method3 = new FixedStepEulerMethod(problem3, 0.001);

paso=0.001;
// 40/0.001=40000, necesitamos 10 arrays
n = 4000;

double media=0;
double time = method3.getSolution().getLastPoint().getTime();
double x0=problem3.getInitialState()[0];

for(int i=0; i<10; i++) {
        x[0]=method3.getSolution().getLastPoint().getState(0);
        int j=0;
        while(j<4000) {
                method3.step();
                x[j]=l+(x0-l)*Math.cos(Math.sqrt(k/m)*time);
                y[j]=Math.abs(x[j]-
                        method3.getSolution().getLastPoint().getState(0))
                time += paso;
                media += y[j];
                j++;
        }
}
media=media/40000;
System.out.println("La media del error es:"+media);
```

Esto, en clase, no me dio tiempo a hacerlo bien. En parte porque me quedaba ya poco tiempo y no estaba pensando con claridad.

Ahora vemos que me arroja un error medio de 0.00192, del orden que buscábamos. Tal y como habíamos predicho.

2.5 Calcular la frecuencia del movimiento no forzado

Para hacer esto, debemos ver el período, y si obtenemos un valor t_0, nuestra frecuencia será $w_0 = \frac{2\pi}{t}$.

Entonces, lo que voy a hacer es buscar la diferencia temporal entre dos puntos con la misma posición, y multiplicar ese tiempo por dos, para obtener el período. Esto podemos hacerlo así porque no hay rozamiento y el movimiento debe ser simétrico.

Obtenemos un período de

$$t_0 = 5.06s$$

El teórico es

$$T = 2\pi\sqrt{\frac{m}{k}} \overset{m=1,k=1.5}{\approx} 5.13s$$

Por lo que está bastante bien.

Entonces, la frecuencia es

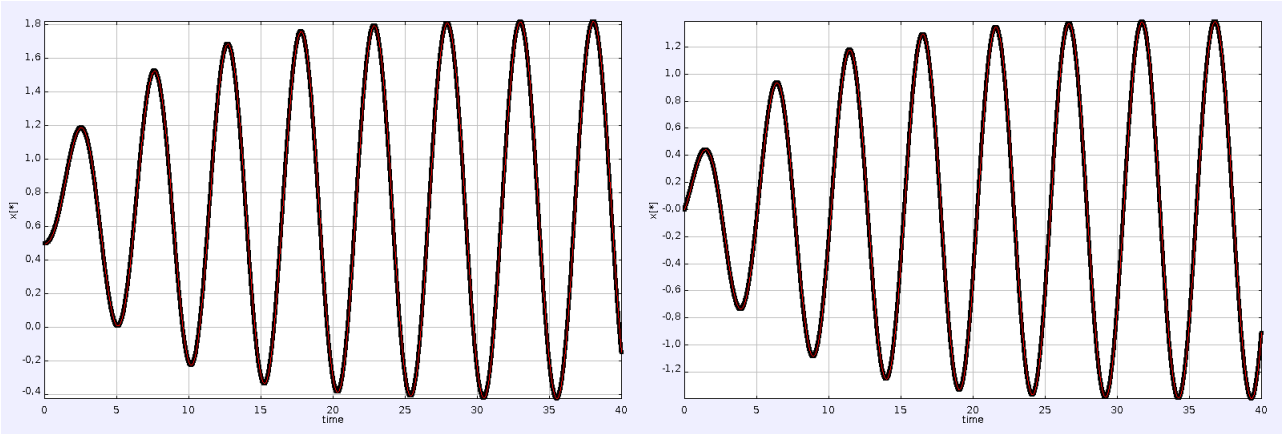
$$w_0 = \frac{2\pi}{t_0} \approx 1.2417$$

2.6 Usar la frecuencia de (5) como w en el caso forzado

Ahora, simplemente repetimos el primer objetivo, cambiando w. Y vemos qué sucede. También voy a plotearlo, como en el segundo objetivo.

El código es el mismo, pero al vector w he añadido el nuevo valor, y en al llamar a la función f, la llamo para que use ese valor.

Obtenemos las siguientes gráficas:



Vemos un comportamiento similar al obtenido con los otros valores de w, la oscilación va aumentando hasta que se estabiliza. Como vimos al principio, la amplitud es mayor que para valores mayores de w.