Encriptado caótico: Sincronización de sistemas caóticos con aplicación al encriptado de señales temporales

Martín Famá

09 Octubre 2020

Resumen

A pesar de la alta sensibilidad que tienen los sistemas caóticos a condiciones iniciales, es posible *sincronizar* dos sistemas para que converjan a la misma trayectoria. Se implementó una simulación del atractor de Lorenz que genera una trayectoria a partir de una condición inicial. Luego, un segundo sistema se acopla a una de las variables del primer sistema e, independientemente de las condiciones iniciales, se verificó la sincronización de ambos sistemas. Se utilizó este método para encriptar señales en el tiempo, enmascarando un mensaje $m(t)$ con la señal caótico del primer sistema y extrayendo el mismo mensaje utilizando el segundo sistema. Luego del desarrollo y análisis teórico, se implementó un programa funcional en C++ para encriptar archivos de audio WAV, y se modificó el funcionamiento del método al notar una manera fácil de romper el encriptado.

Contenidos

1 Introducción **2**

1.1 El sistema de Lorenz 2

2 Simulación del sistema de Lorenz mediante el método de Runge-Kutta **3**

2.1 El sistema receptor 4

2.2 Convergencia de la trayectoria 4

2.3 No todas las variables son aptas para sincronizar 5

3 Encriptando un mensaje $m(t)$ **6**

3.1 Encriptando un mensaje de audio 7

3.2 La frecuencia del atractor es demasiado baja 7

3.3 Sampleado distanciado del atractor 8

4 Conclusiones **9**

1 Introducción

La dinámica caótica presenta la oportunidad de su aplicación en el área de la encriptación. En ciertos casos se permite la posibilidad de *sincronizar* dos sistemas. Esto se logra acoplando una o varias de las variables del primer sistema al segundo. De esta manera, indistintamente de las condiciones iniciales del segundo sistema, este mismo converge a la trayectoria del primero. ¿Cómo se puede usar de base para encriptar? Consiste en enmascarar una señal mensaje $m(t)$ sobre una variable caótica (digamos $u_t(t)$) para ser *transmitida* como señal $s(t) = u_t(t)+\epsilon m(t)$, con ϵ una amplitud pequeña respecto a $u(t)$. Luego, un sistema *receptor* acoplado al *transmisor* recibe esta señal como variable, y se simula su evolución, con particular interés en su variable $u_r(t)$, "clon" de $u_t(t)$. Al tender $u_r(t)$ a $u_t(t)$, el receptor puede recuperar el mensaje: $\hat{m}(t) = (s(t) - u_r(r))/\epsilon \approx m(t)$.

1.1 El sistema de Lorenz

El sistema de Lorenz es un sistema de ecuaciones diferenciales ordinarias que presenta, dentro de un cierto rango de sus parámetros (y condiciones iniciales), comportamiento caótico. Para nuestros propósitos, se presenta el sistema con factores de reescaleo para que las variables puedan ser voltajes en un rango razonable para su implementación como circuito eléctrico [1]:

$$\dot{u_t} = \sigma(v_t - u_t) \tag{1}$$
$$\dot{v_t} = ru_t - v_t 20 u_t w_t \tag{2}$$
$$\dot{w_t} = 5u_t v_t - bw_t \tag{3}$$

donde ya vamos insinuando con los subíndices que se trata del sistema transmisor. El receptor tiene el sistema casi idéntico:

$$\dot{u_r} = \sigma(v_r - u_r) \tag{4}$$
$$\dot{v_r} = ru_t - v_r 20 u_t w_r \tag{5}$$
$$\dot{w_r} = 5u_t v_r - bw_r \tag{6}$$

Con la simple acción de reemplazar u_r por u_t en las ecuaciones 5 y 6, el sistema receptor queda acoplado al transmisor, y su trayectoria convergerá a la del mismo, sin importar las condiciones iniciales.

Usando $u_t(t)$ y $u_r(t)$ como en el método explicitado en la primer parte de la introducción, tenemos un sistema listo para encriptar señales temporales. De todas maneras, exploremos la simulación del sistema de Lorenz, y veamos como funciona el sincronizado de un segundo sistema.

2 Simulación del sistema de Lorenz mediante el método de Runge-Kutta

Presentamos el algoritmo utilizado para simular al sistema de Lorenz (tanto el transmisor como el receptor, este último con las modificaciones adecuadas). Esta parte más teórica se implementó en Python.

```
#transmissor solves the Lorenz system numerically using RK4 (fixed step)
#Arguments: initial values (u_0, v_0, w_0)
#           final time (t_f)
#           initial stepsize (h)
def transmissor(u_0, v_0, w_0, t_f, h):
    r, sigma, b = 60, 10, 8/3
    #returns the three differential equations evaluated at u,v,w
    def f(u, v, w):
        return sigma*(v-u), r*u-v-20*u*w, 5*u*v-b*w
    #gives the next RK4 step from (u,v,w), using stepsize h_
    def RK4(h_, u, v, w):
        k_1_u, k_1_v, k_1_w = f(u,            v,            w)
        k_2_u, k_2_v, k_2_w = f(u+h_*k_1_u/2, v+h_*k_1_v/2, w+h_*k_1_w/2)
        k_3_u, k_3_v, k_3_w = f(u+h_*k_2_u/2, v+h_*k_2_v/2, w+h_*k_2_w/2)
        k_4_u, k_4_v, k_4_w = f(u+h_*k_3_u,   v+h_*k_3_v,   w+h_*k_3_w)
        u_ = u + h_*(k_1_u+2*k_2_u+2*k_3_u+k_4_u)/6 # + o(h^5)
        v_ = v + h_*(k_1_v+2*k_2_v+2*k_3_v+k_4_v)/6 # + o(h^5)
        w_ = w + h_*(k_1_w+2*k_2_w+2*k_3_w+k_4_w)/6 # + o(h^5)
        return u_, v_, w_
    #will contain time points
    t = np.arange(0, t_f, h)
    #trajectory of system (we will return these arrays (and t) at the end)
    u = [u_0]
    v = [v_0]
    w = [w_0]
    for j in range(1, len(t)):
        u_, v_, w_ = RK4(h, u[j-1], v[j-1], w[j-1])
        u.append(u_)
        v.append(v_)
        w.append(w_)
    return t, u, v, w
```

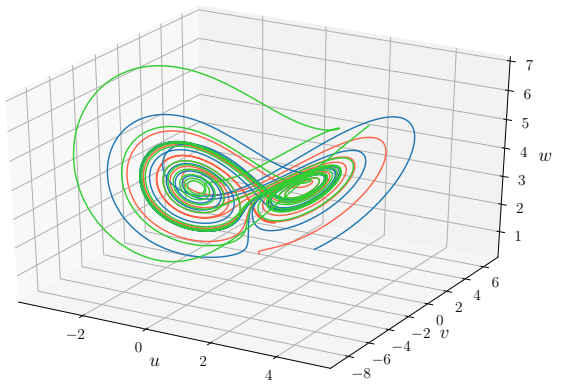


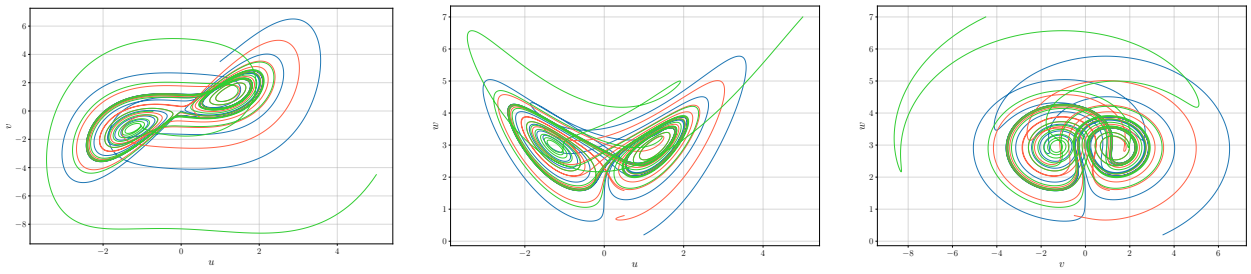
Fig. 1: Tres trayectorias con condiciones iniciales distintas tienden al atractor de Lorenz.

El algoritmo es relativamente simple, y nos devuelve la trayectoria del sistema a lo largo del tiempo. Vemos que $r = 60$, $\sigma = 10$ y $b = 8/3$, valores en los cuáles el sistema está en régimen caótico.

En la figura 1 podemos ver el sistema simulado, con distintas condiciones iniciales. Vemos que en los tres casos las trayectorias convergen a una región del espacio denominada *atractor de Lorenz*. Es justamente el atractor de las trayectorias en el régimen caótico: independientemente de las condiciones iniciales, las trayectorias terminan en el atractor.[1]

En la figura 2 se muestran proyecciones sobre distintos planos. En particular en la figura 2b se puede ver claramente la famosa forma "mariposa" del atractor.

[1]Esto no implica que las trayectorias son las mismas. Justamente, al ser un sistema caótico, la trayectoria es extremadamente sensible a condiciones iniciales. El atractor es simplemente una región en el espacio en donde todas las trayectorias terminan.



(a) Proyección u-v (b) Proyección u-w (c) Proyección v-w

Fig. 2: Proyecciones de la figura 1

2.1 El sistema receptor

El algoritmo del sistema receptor es muy similar al del transmisor, con la salvedad de que acepta como argumento la trayectoria $u_t(t)$, que usa para acoplarse al transmisor (no se incluye el código pues es tan similar al del transmisor que sería redundante). Veamos como evoluciona el receptor al ser alimentado con la señal del transmisor.

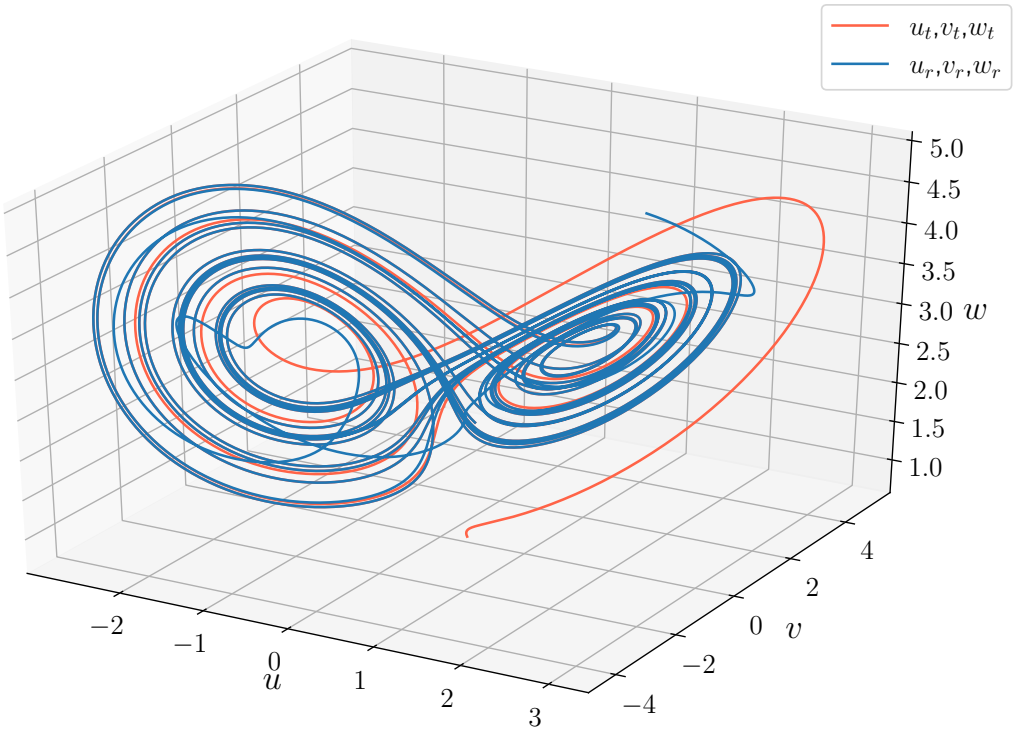


Fig. 3

En la figura 3 se pueden ver las trayectorias del sistema transmisor para ciertas condiciones iniciales (órbita roja), y de un sistema receptor acoplado al primero (órbita azul). El receptor tiene una condición inicial completamente distinta al transmisor, pero se termina sincronizando de manera muy rápida. ¿Qúe tan rápida? Exponencialmente.

2.2 Convergencia de la trayectoria

En la figura 4 se puede ver graficada la distancia punto a punto entre las dos trayectorias a lo largo del tiempo. En particular, se muestran 3 corridas con h (el intervalo del tiempo del RK4) entre 10^{-4} y 10^{-6}. Es interesante notar que el error decae exponencialmente (estamos en escala logarítmica), y llega a un tope que podemos atribuir a la resolución del RK4. No solo eso, el error oscila de manera periódica al rededor de este límite. Muy curiosamente, vemos que para $h = 10^{-4}$ y $h = 10^{-5}$, trayectorias en donde se eligieron las mismas condiciones iniciales, la

evolución del error conserva su forma.

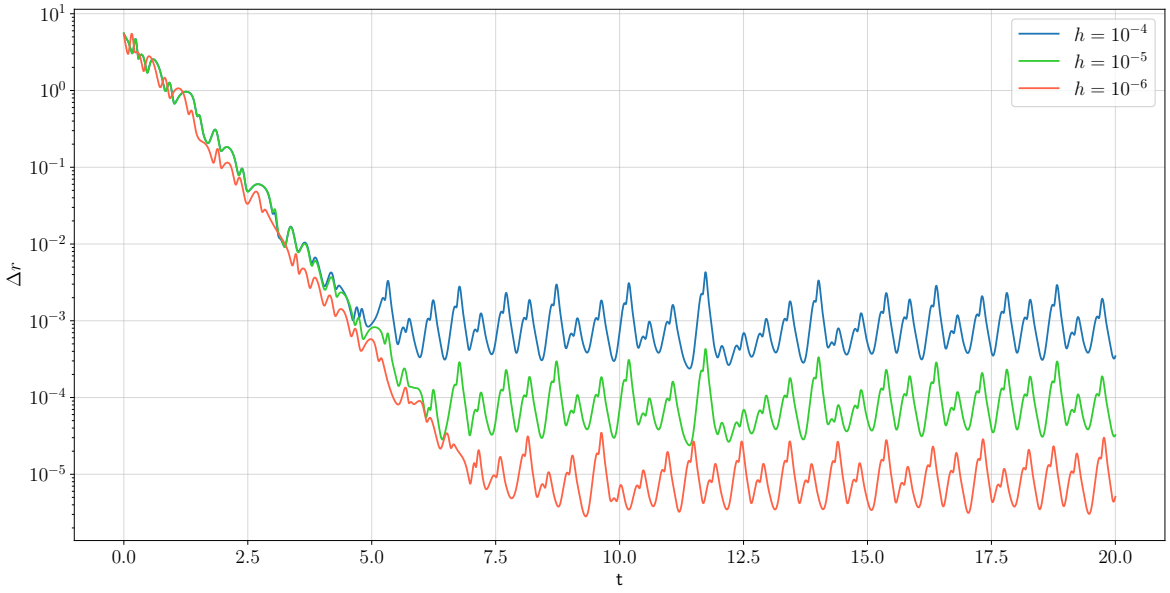


Fig. 4: Distancia punto a punto entre la trayectoria del transmisor y del receptor (h es el intervalo de tiempo usado en el RK4).

2.3 No todas las variables son aptas para sincronizar

La sincronización de los sistemas caóticos no funciona con cualquier variable. Por ejemplo, podemos intentar sincronizar el transmisor y el receptor con las variables $w_t(t)$ y $w_r(t)$, pero como podemos ver en la figura 5, los sistemas no se sincronizan. De hecho, el sistema del receptor diverge de la región del atractor de Lorenz.

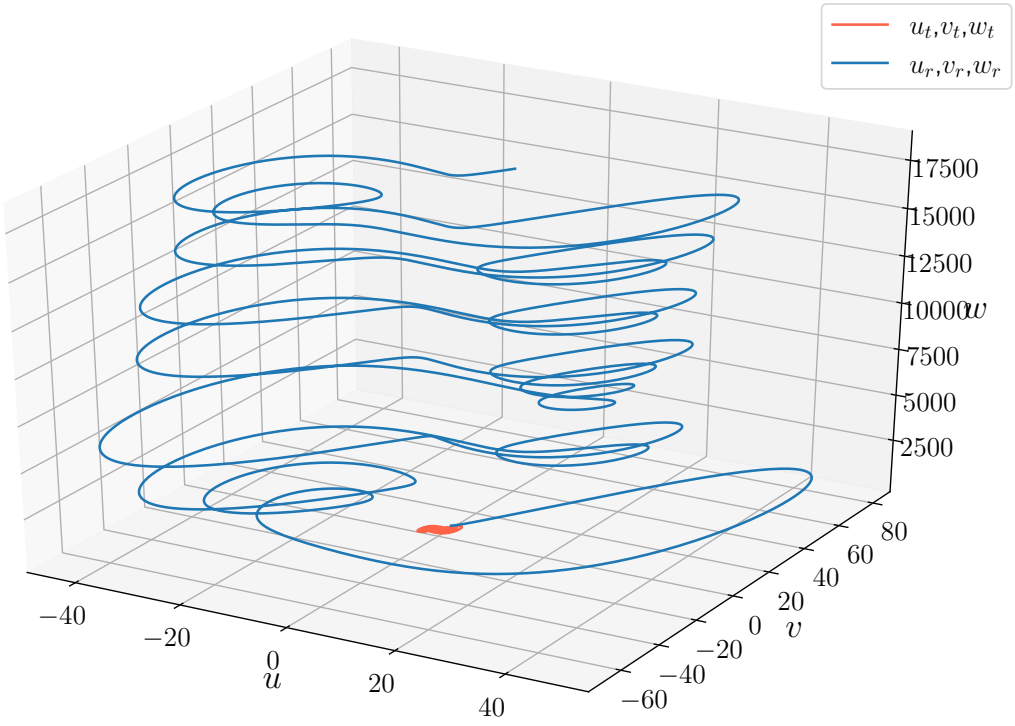


Fig. 5: Acoplando con $w_t(t)$ no sincroniza a los sistemas.

3 Encriptando un mensaje $m(t)$

Esperando que al lector le hayan quedado claro todos los planteos, se puede ver en la figura 6 un mensaje $m(t) = \sin(2\pi f t)$, con $f = 500$ Hz, enviado por un sistema transmisor, y el mensaje decriptado por el receptor: $\hat{m}(t)$. El procedimiento consistió en simular 10 segundos iniciales en donde el mensaje no es montado, como para darle tiempo a los sistemas para que se sincronicen. Luego, se montaron 10 segundos del tono sinusoidal y se llevó a cabo el resto.

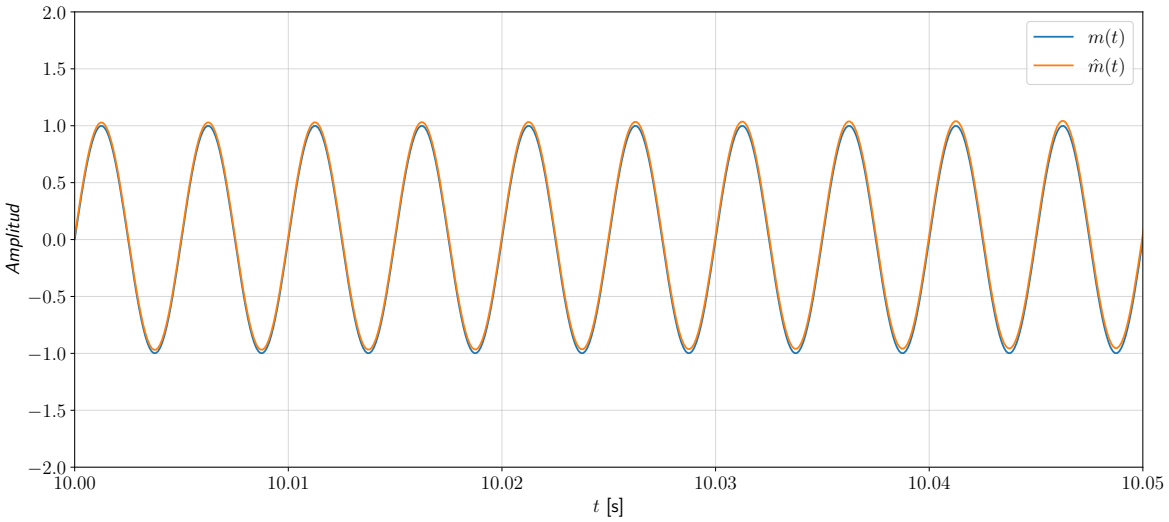


Fig. 6: Segmento del mensaje original sinusoidal de 500 Hz y la versión decriptada.

Vemos inmediatamente que el mensaje decriptado es muy similar al original. Hagamos un análisis que siempre vale la pena hacer: un barrido de frecuencias. En la figura 7 podemos ver la diferencia $\Delta r = |\hat{m}(t) - m(t)|$ punto a punto para mensajes sinusoidales de distintas frecuencias (escala log). Vemos que en general, el error disminuye para frecuencias más altas, con variaciones en el error a lo largo del tiempo que oscilan al rededor de un punto medio. Hay que tener en cuenta también que la precisión estará limitada por el paso temporal utilizado en el RK4 (en este caso se uso $h = 10^{-5}$).

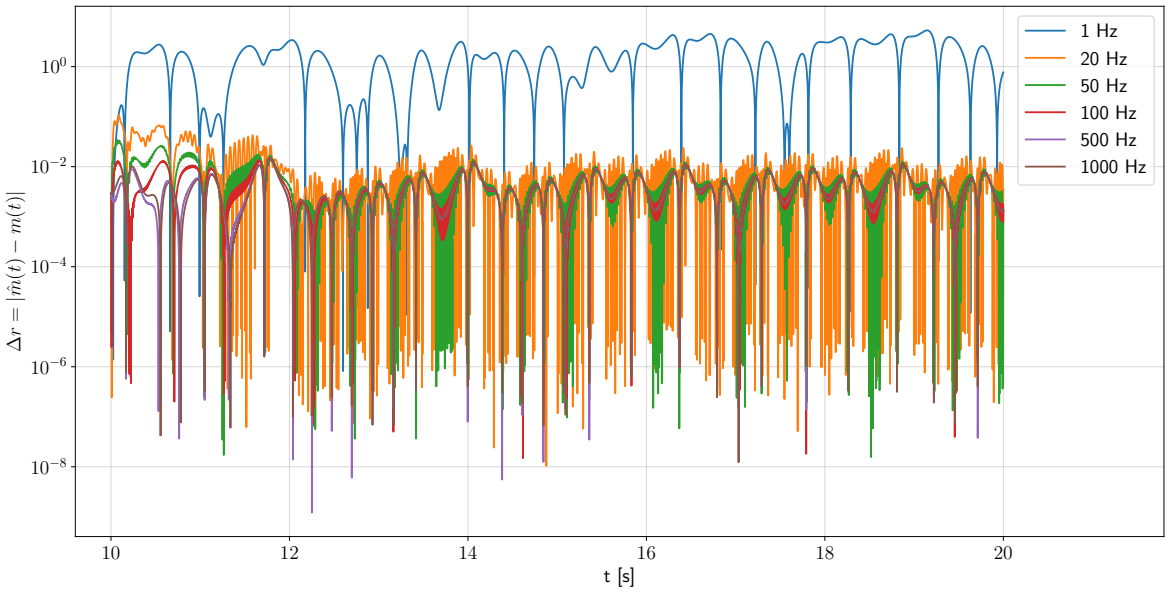


Fig. 7: Errores a lo largo del tiempo en los mensajes decriptados. Tonos sinusoidales de distintas frecuencias.

3.1 Encriptando un mensaje de audio

Teniendo el funcionamiento teórico planteado, hagamos lo que el método promete. Encriptemos un mensaje de audio. Se implementó un programa en C++ que nos permite encriptar y decriptar archivos de audio WAV. En la figura 8 se puede ver un audio original, la señal encriptada enviada por el transmisor, y el audio decriptado por el receptor. Ya nos podemos ir emocionando. Vemos que el audio encriptado es completamente distinto al mensaje original, al menos al ojo. El audio decriptado, que de nuevo, la generó el receptor a partir de solo el audio encriptado, es muy parecido al audio original.

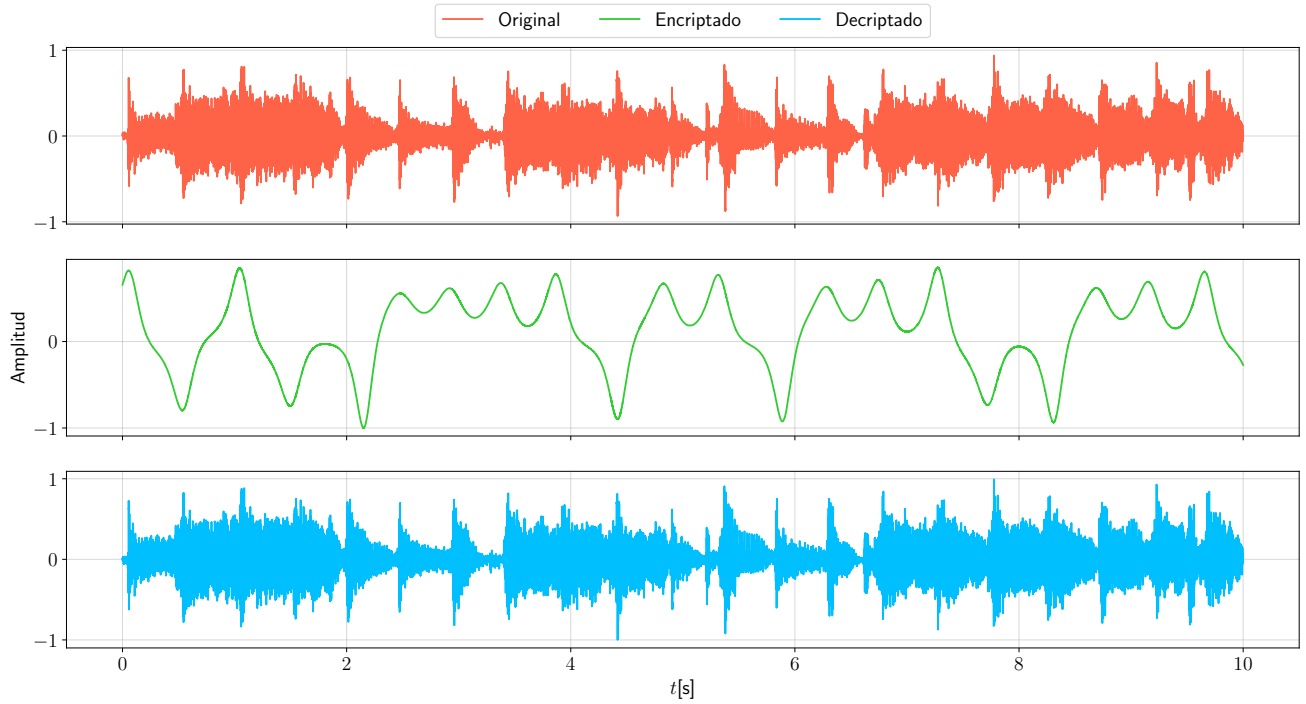


Fig. 8: Mensaje de audio original, luego encriptado y finalmente decriptado por el receptor. Pruebas informales de escuchar indican una reproducción muy exacta del audio original.

3.2 La frecuencia del atractor es demasiado baja

A pesar del "correcto" funcionamiento del método, notamos inmediatamente una debilidad en poder enviar audio: el atractor tiene una frecuencia, así como la sampleamos, muy baja. Para romper el encriptado uno puede pasar la señal por un filtro pasa altos con frecuencia de corte baja y amplificar la salida. Así recuperaríamos la mayoría de las frecuencias del mensaje original, y si es audio con eso alcanza para entender a alguien hablando, sonidos, música, etc. En la figura 9 se puede ver el resultado de este mismo proceso.

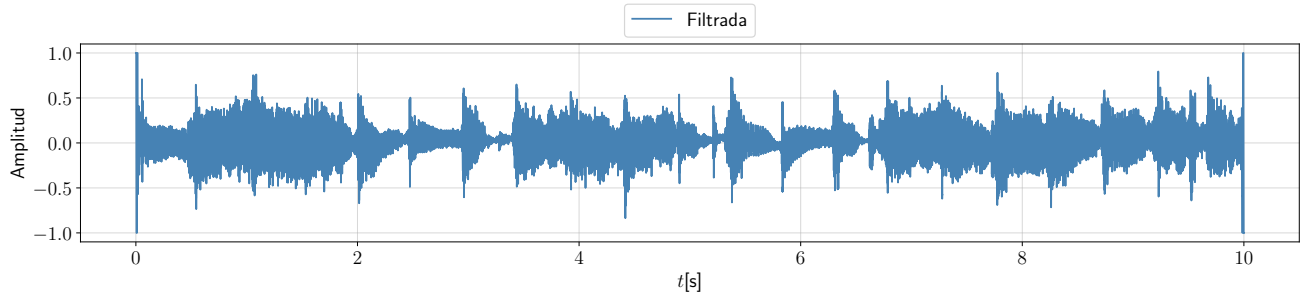


Fig. 9: La señal encriptada pasada por un pasa altos y amplificada. El audio suena muy parecido al original.

Para remediar esta falla catastrófica, se "aumento" la frecuencia del atractor relativamente al mensaje *sampleando* al atractor de manera distanciada. El costo de esto es un incremento en el largo de la trayectoria del atractor, proporcional a la distancia de sampleo.

3.3 Sampleado distanciado del atractor

El sampleado distanciado consiste en que la señal enviada por el transmisor sea compuesta así:

$$s(t_n) = u(t_n) + \begin{cases} \epsilon m(t_n), & \mod (n, s) = 0 \\ 0, & \mod (n, s) \neq 0 \end{cases}$$

en donde s es la distancia de sampleo. Justamente indica que el mensaje se debería superponer en $u_t(t)$ cada s intervalos de tiempo. Esto aumenta la frecuencia relativa entre $u_t(t)$ y $m(t)$. Vemos inmediatamente que esto aumenta la cantidad de puntos que hay que calcular para la trayectoria del atractor. Si $m(t)$ es un mensaje con N puntos, necesitamos calcular sN puntos para $u_t(t)$ y $s(t)$.

En la figura 10 podemos ver el efecto del sampleo distanciado. Se muestran el mensaje encriptado para distintas distancias. La frecuencia del atractor aumenta y entonces nuestro mensaje original no queda expuesto.

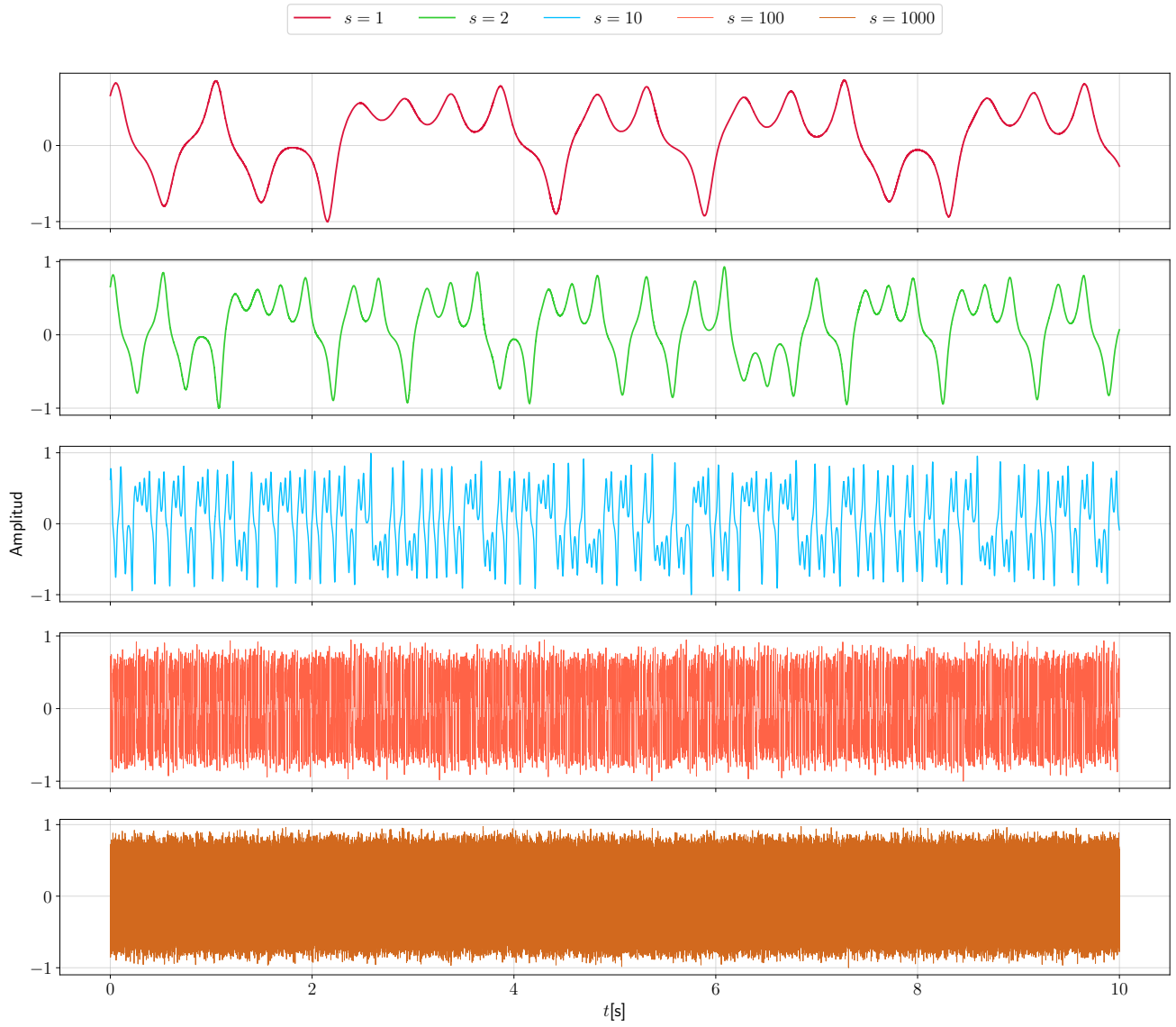


Fig. 10: La frecuencia del atractor aumenta (relativamente al mensaje) al aumentar la distancia de sampleado s.

Podemos analizar con más detalle que rango de frecuencias llega a ocupar el atractor haciendo una transformada de Fourier. En la figura 11 se puede ver el espectro de frecuencias de $u(t)$ para distintas distancias de sampleado s. Además, tenemos el espectro de frecuencias del audio con el que venimos trabajando, con una atenuación de $\epsilon = 0.01$. Vemos que la presencia de frecuencias altas aumenta al aumentar s. Esto es exactamente lo que queremos, así el mensaje $m(t)$ queda opacado y mezclado con $u(t)$, haciendo que su extracción con un filtro sea cada vez menos posible.

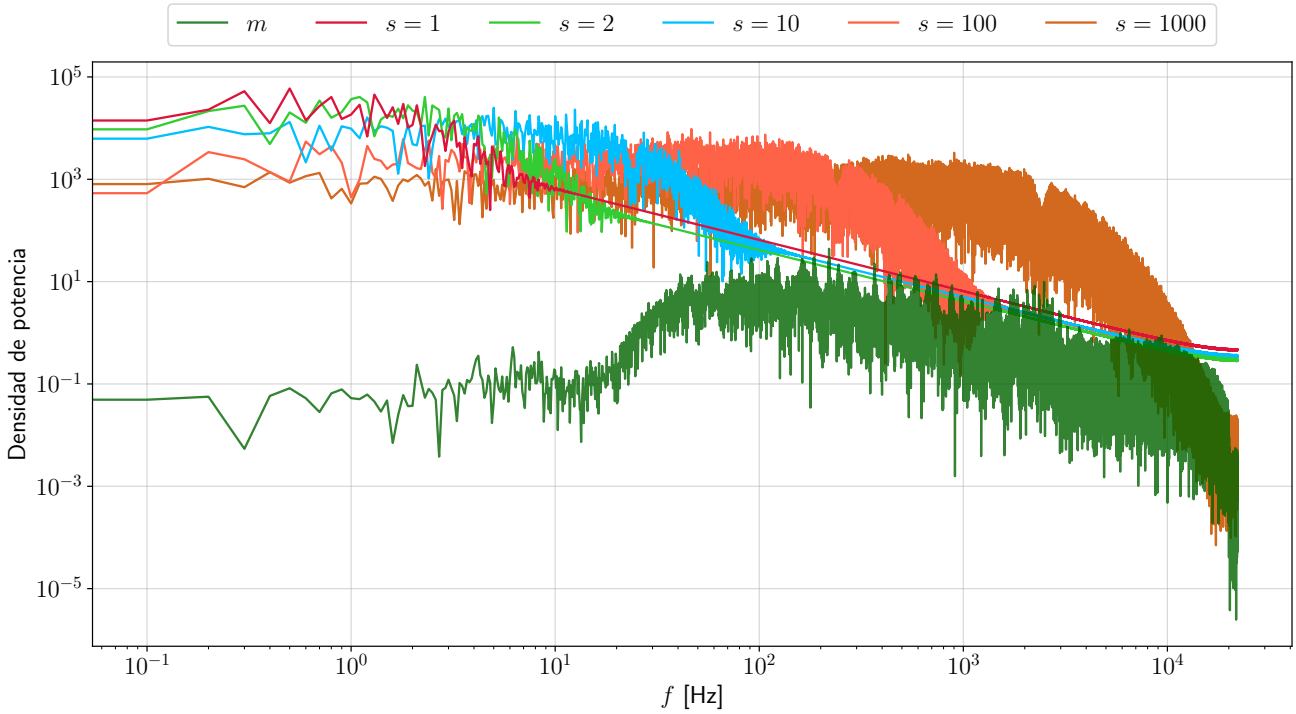


Fig. 11: Espectro de frecuencias de $m(t)$ y de $u(t)$ para distintas distancias de sampleo s.

Claramente el mejor resultado viene dado por el s más grande, en este caso 1000. Sin embargo, a fines prácticos, no es conveniente usar un s tan grande. Nuestro mensaje original de 10 segundos está en formato WAV, canal mono, de 16 bits de profundidad y una frecuencia de sampleo de 44100 Hz. Esto implica 441000 puntos de sampleo, y luego como por cada punto tenemos $s = 1000$ puntos del atractor, terminamos con 4.41e8 puntos. Teniendo en cuenta que cada punto lo estamos guardando como un tipo double de 4 bytes, terminamos con un archivo encriptado de 3.7 GB. Solo para enviar 10 segundos de audio. Es fácil imaginarse lo poco práctico que puede ser esto para mensajes muchos más largos. Con $s = 100$ tenemos un archivo encriptado de 370 MB, que se considera un poco más manejable, al menos en el situación actual de almacenamiento y ancho de banda al que se tiene acceso cotidianamente. Vemos que para $s = 100$, el mensaje queda opacado por $u(t)$ hasta aproximadamente los 1000 Hz. Se intentó extraer el mensaje aplicando filtros a esta señal, y el intentó falló exitosamente (no se descarta que con métodos más elaborados se podría de todas maneras romper el encriptado). Es decir, con $s = 100$ podemos encriptar de manera relativamente segura un mensaje de audio.

4 Conclusiones

Se ha estudiado la sincronización de sistemas caóticos, particularmente el sistema de Lorenz. Verificando la sincronización de dos sistemas, se procedió a implementar un algoritmo para encriptar un mensaje $m(t)$ sobre la señal del primer sistema. Habiendo logrado esto, se desarrolló el programa en C++ que permite encriptar archivos de audio WAV, con la nueva opción de aumentar la distancia de sampleado s. Esta última modificación nos aumenta la calidad del encriptado, con el costo de archivos encriptados de largo proporcional a sN, con N el largo del mensaje $m(t)$ en puntos sampleados. Teniendo en cuenta el acceso cotidiano a almacenamiento y ancho de banda, se determinó que $s \approx 100$ es un distanciado razonable.

Referencias

[1] Cuomo, Oppenheim and Strogatz, *Synchronization of Lorenz-Based Chaotic Circuits with Applications to Communications*. IEEE Trans. Circuits and Systems II - Analog and Digital Proecssing 40:626 (1993).