Transformada de Fourier y la librería de FFT de LIBROW

Transformada de Fourier

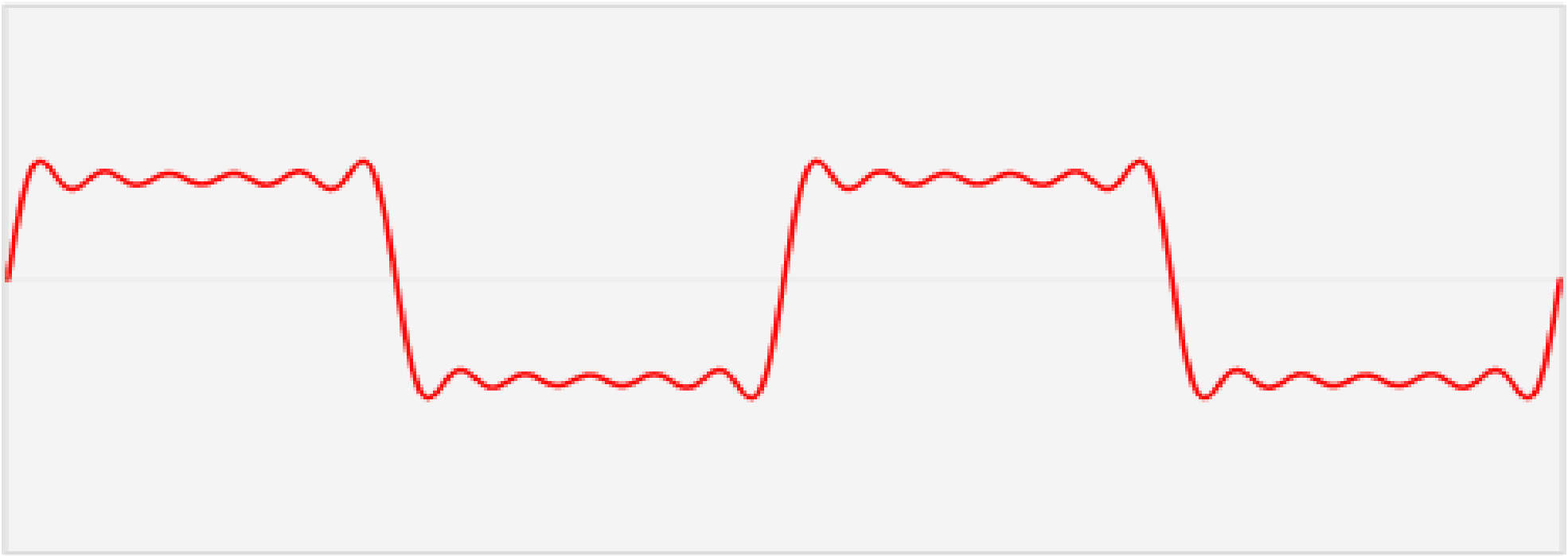
- En 1807, Fourier propuso una solución a una ecuación conocida como “La Ecuación de Calor”.
- Esta ecuación trataba de describir la manera en la que el calor se distribuía en una placa de metal, dada la existencia de fuentes de calor conocidas.
- Era una ecuación diferencial parcial parabólica, que en ese entonces no tenía solución.

Transformada de Fourier

- Antes de Fourier, soluciones particulares para esta ecuación habían sido propuestas.
- Sólo aplicaban si la fuente de calor se comportaba como una *onda*.
  - Dícese, si se comportaba como una señal que oscila a una única frecuencia, como una función de seno o coseno.

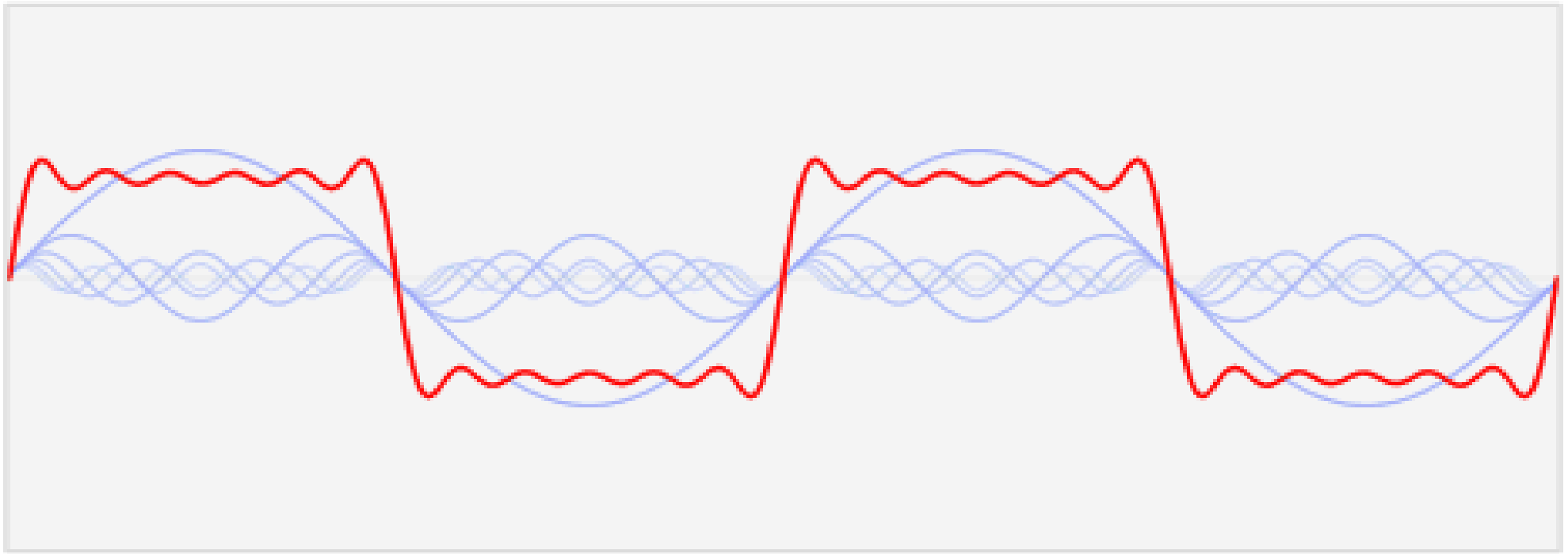
Transformada de Fourier

- Fourier generalizó dichas soluciones de la siguiente manera:
- **Varias ondas sumadas entre si pueden ser utilizadas para representar cualquier señal *periódica***.
- Y así, se inventó las Series de Fourier.

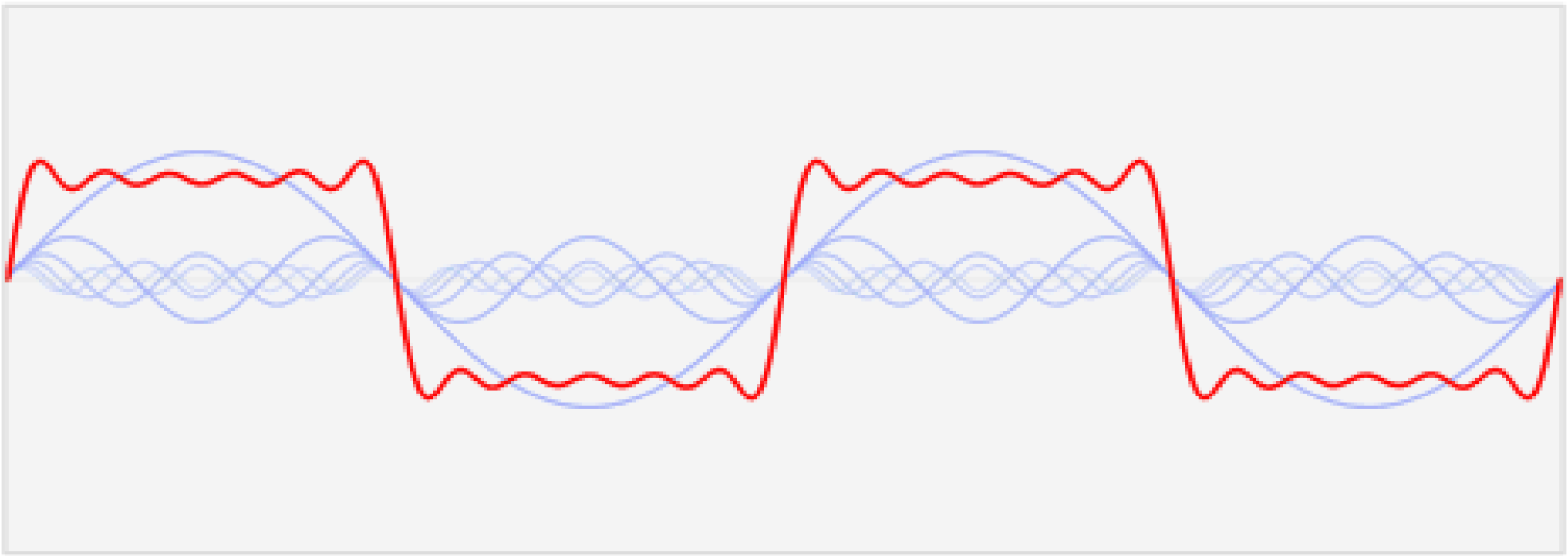


f

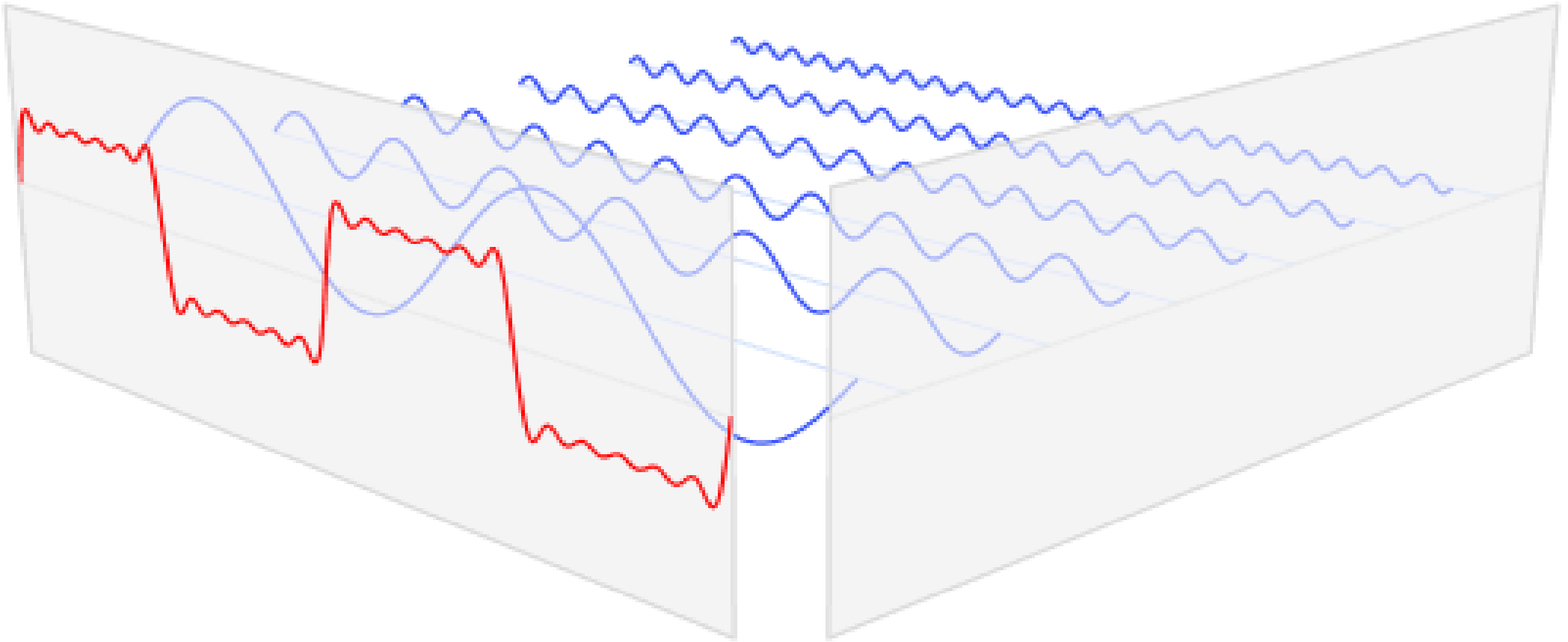
f

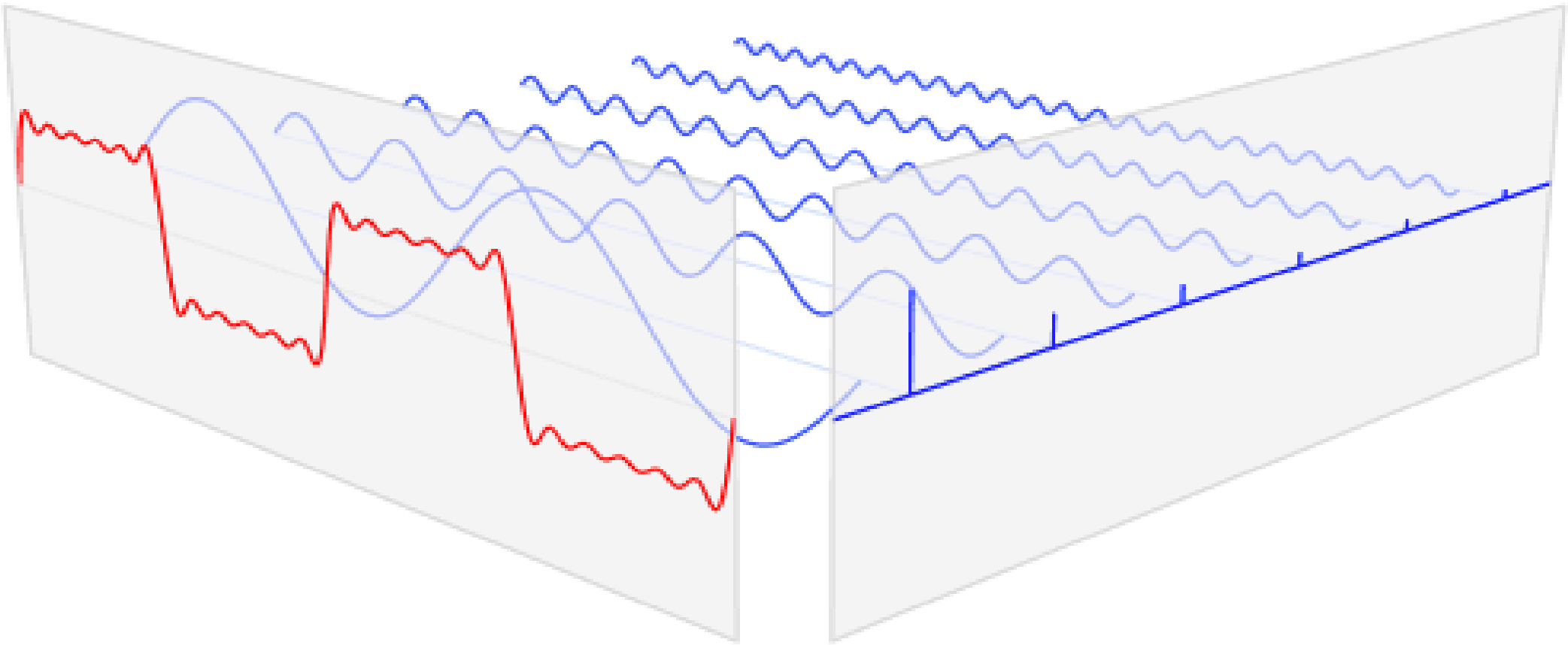


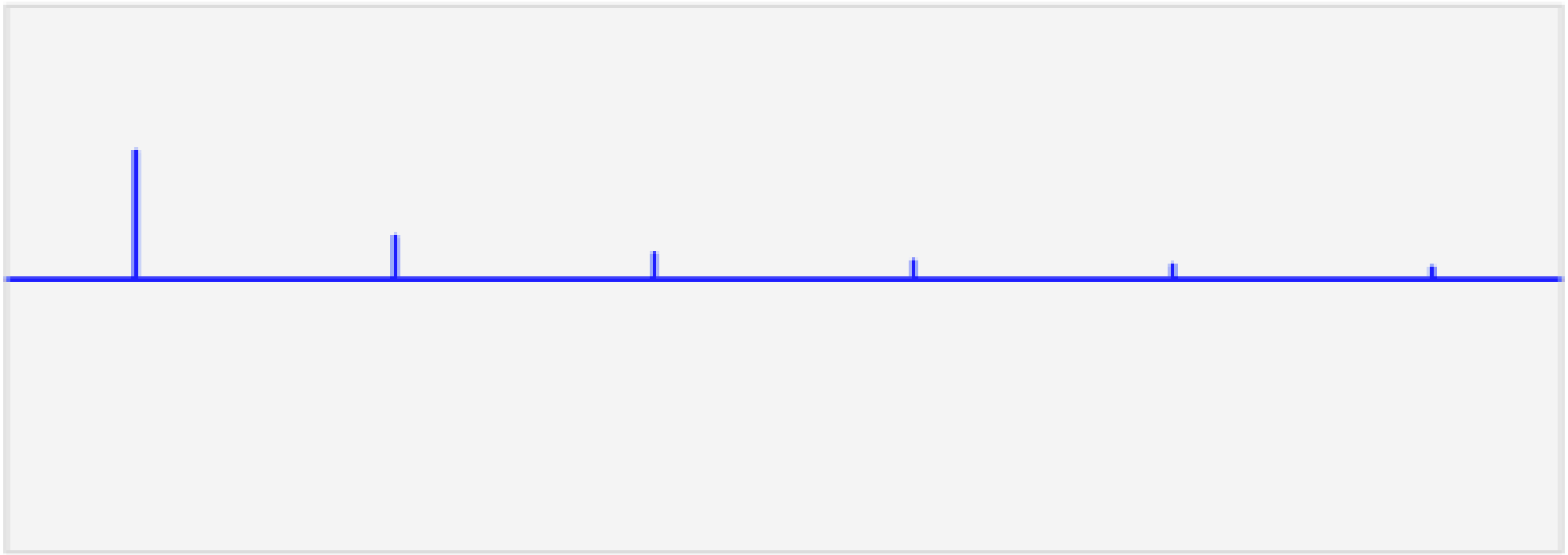
$$f$$

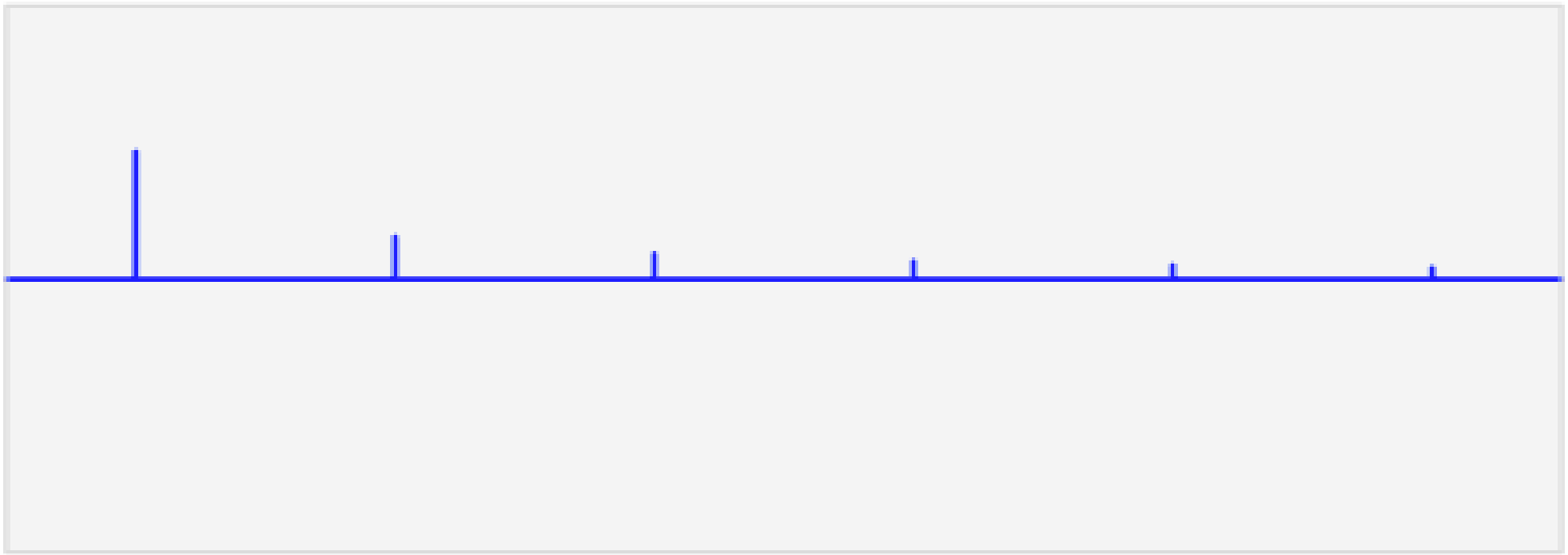


$$a_n \cos(nx) + b_n \sin(nx)$$



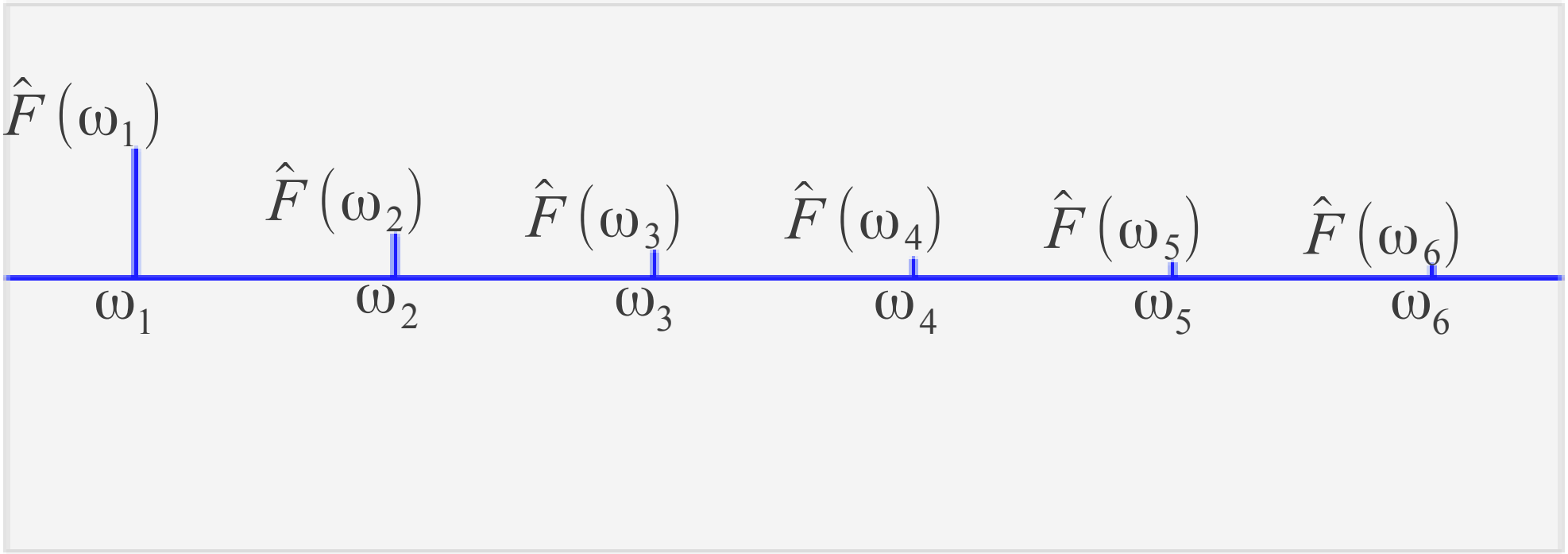






$\hat{f}$

- F es la señal f transformada al dominio de la frecuencia.
- ω_n son las frecuencias de las ondas que, al sumarse, representan a f.
- $F(\omega_n)$ es la magnitud de la onda que oscila con frecuencia ω_n.

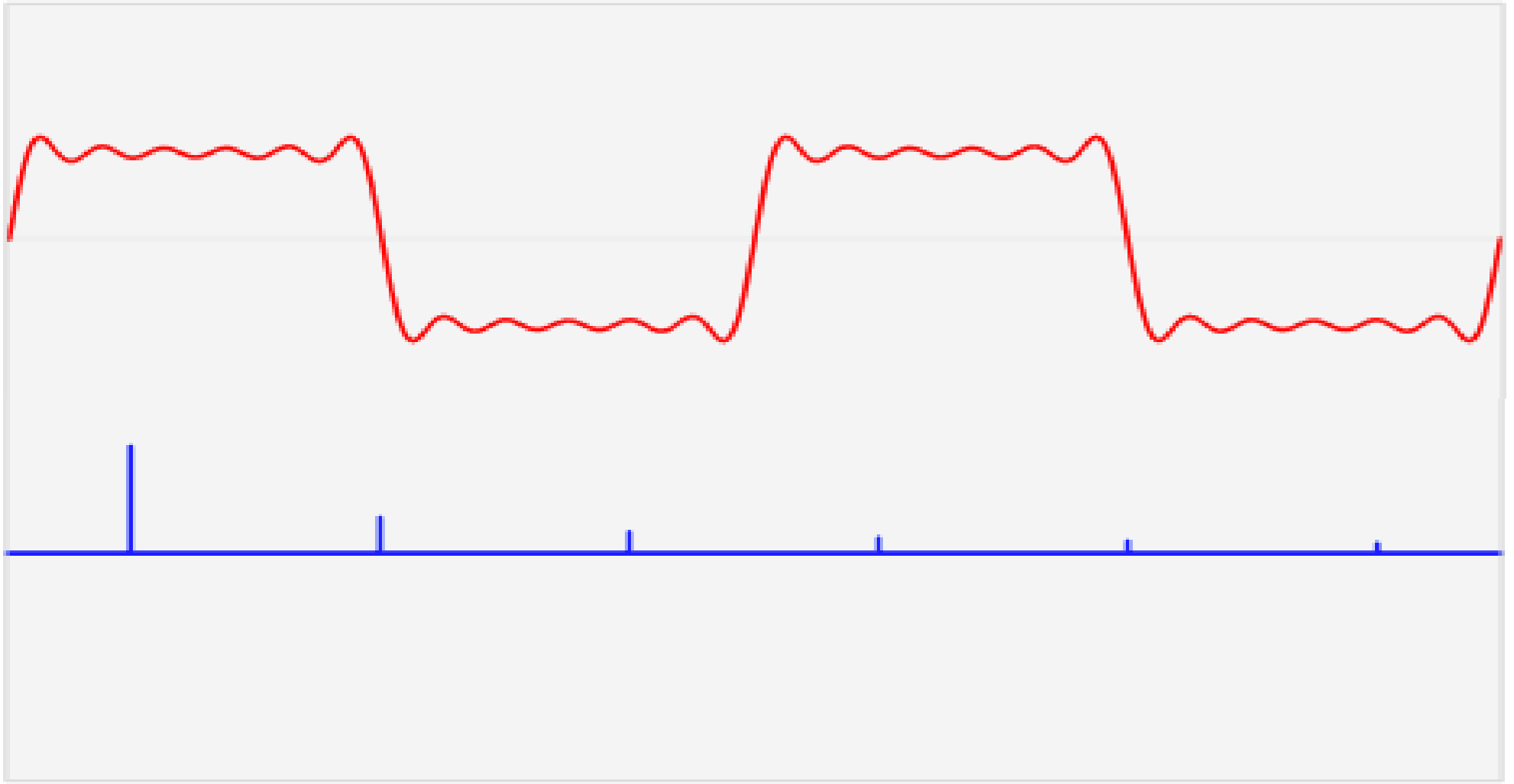


$\hat{f}$

¿Para qué?

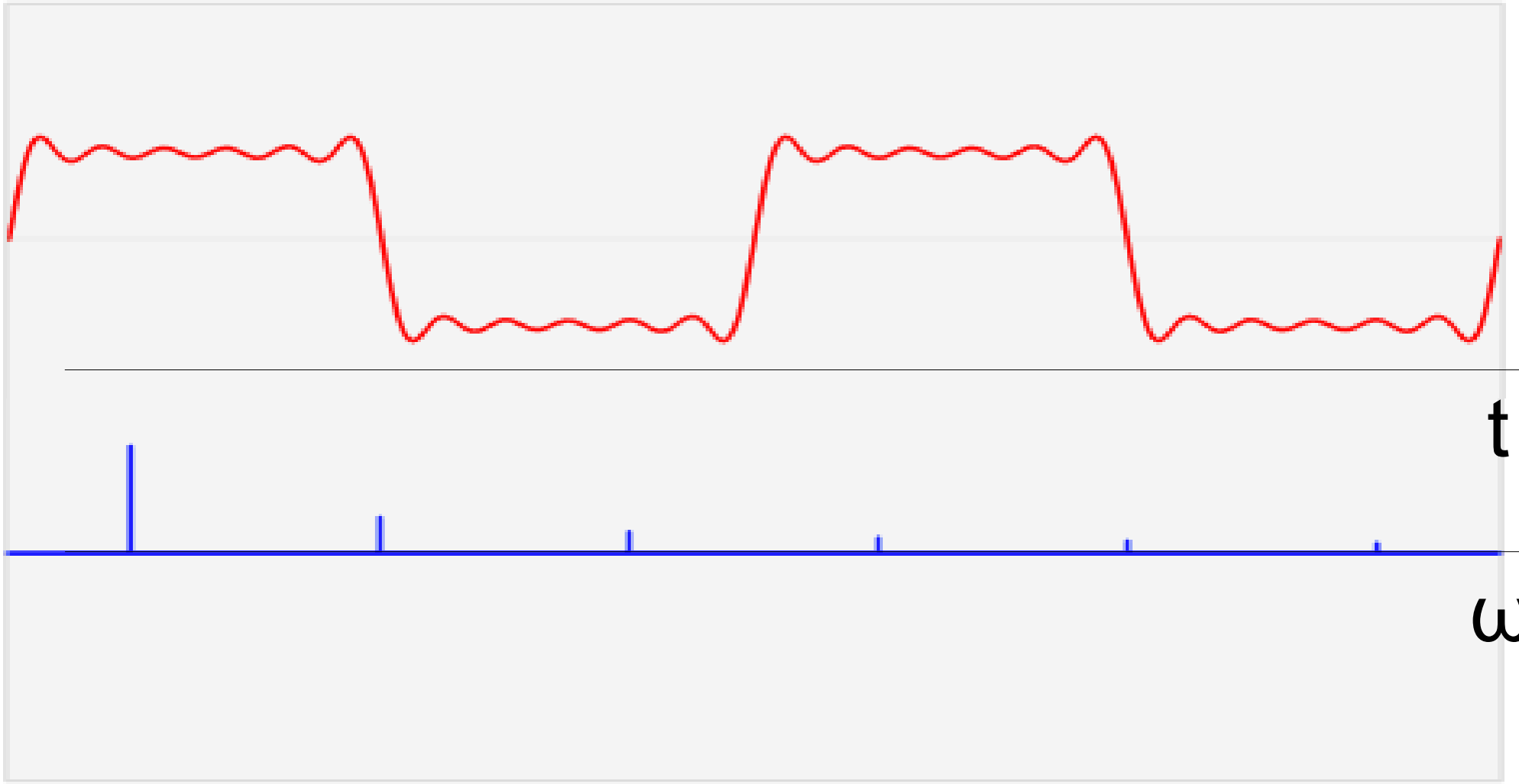
- Permite representar una señal, cualquier señal, en otro *dominio*.
- Es la misma señal, sólo que es vista de dos puntos de vista (o “dominio”) diferente.

f



$\hat{f}$

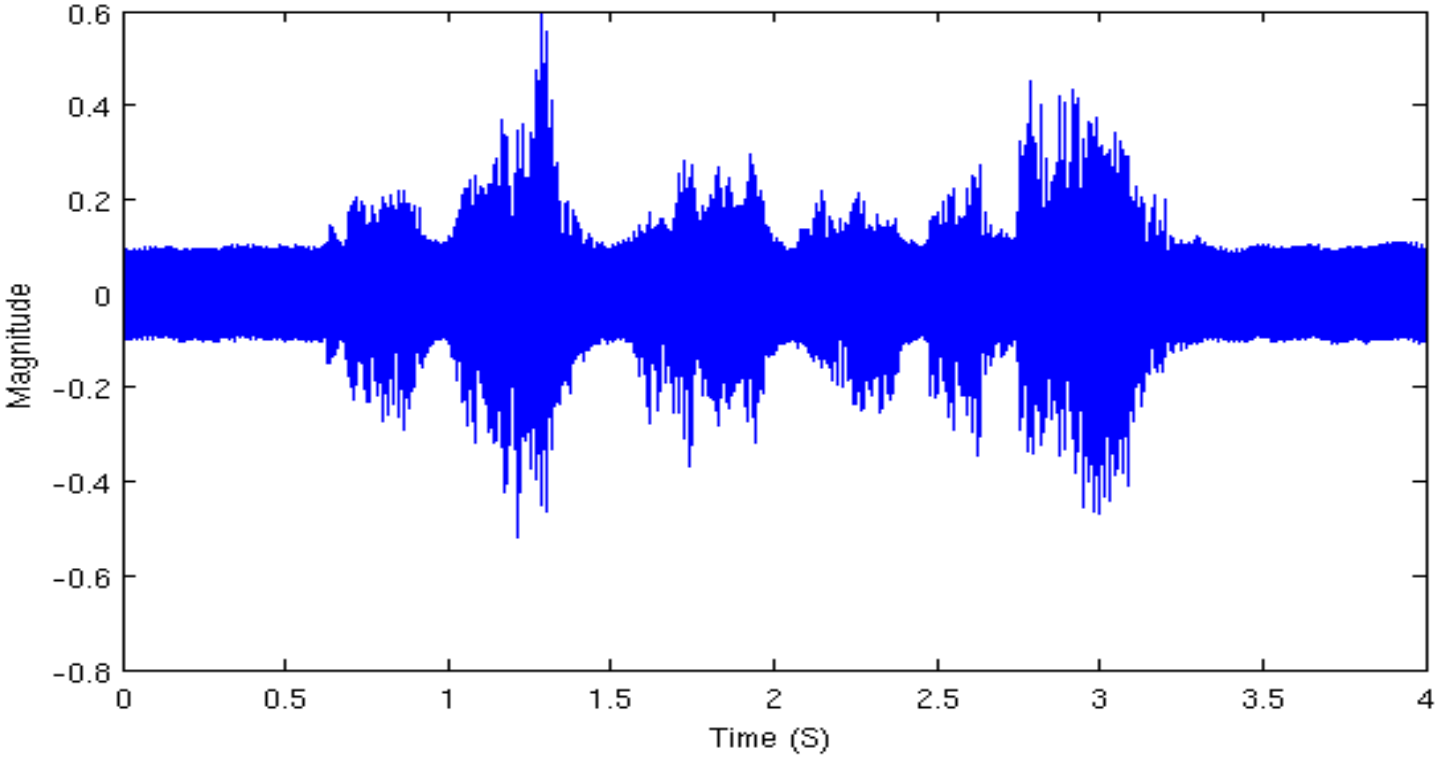
f



$\hat{f}$

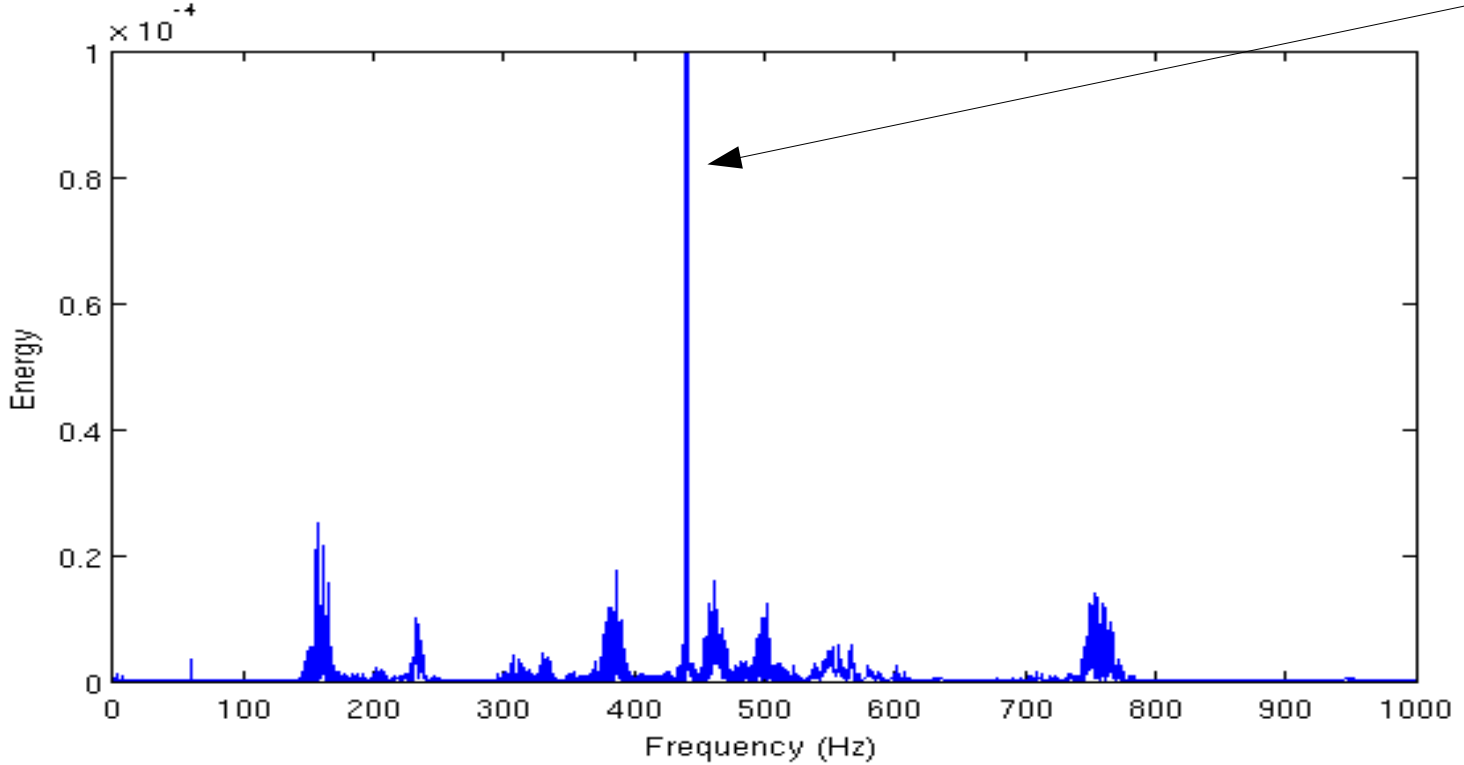
Pero, podemos hacer muchas cosas en el dominio del tiempo... es cómodo y bonito.

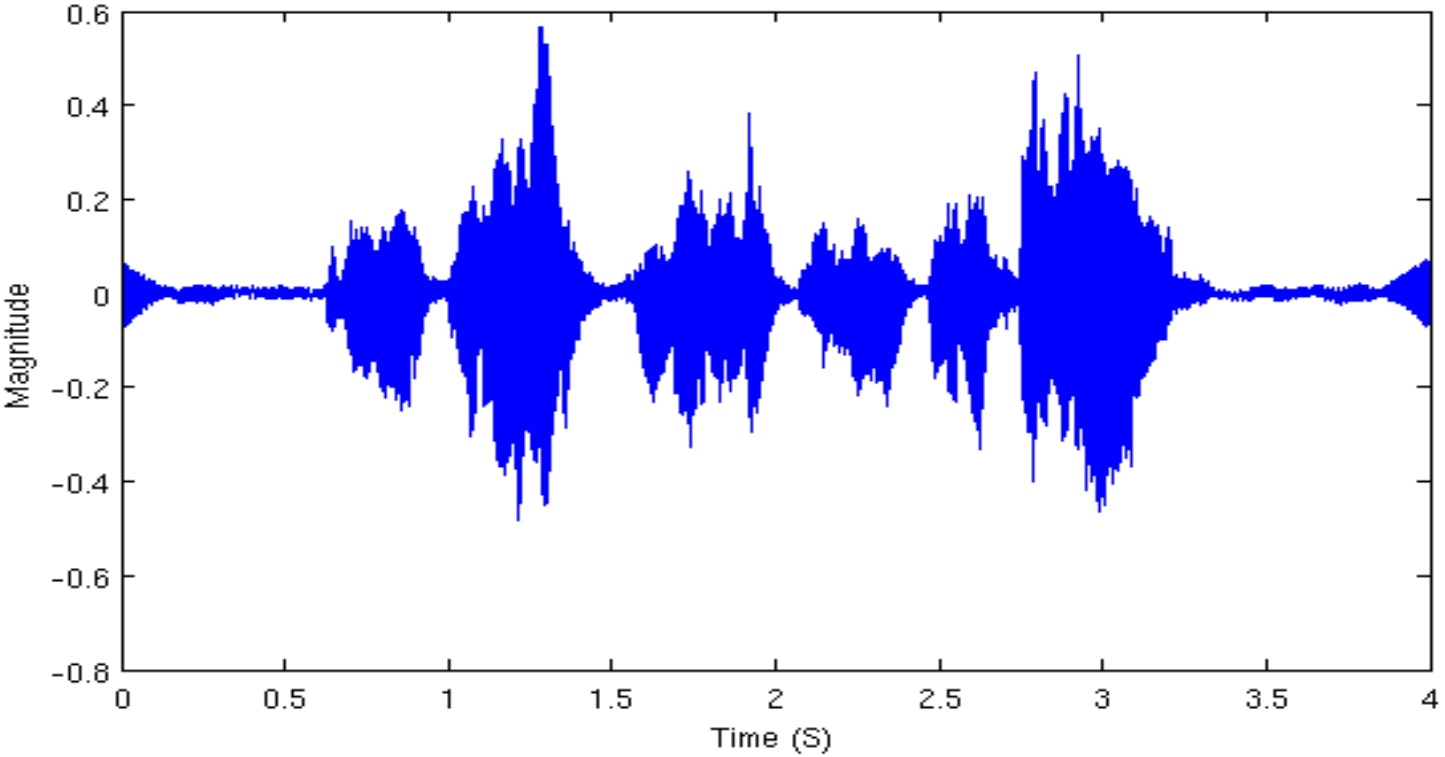
¿Para qué necesitamos irnos al dominio de la frecuencia?

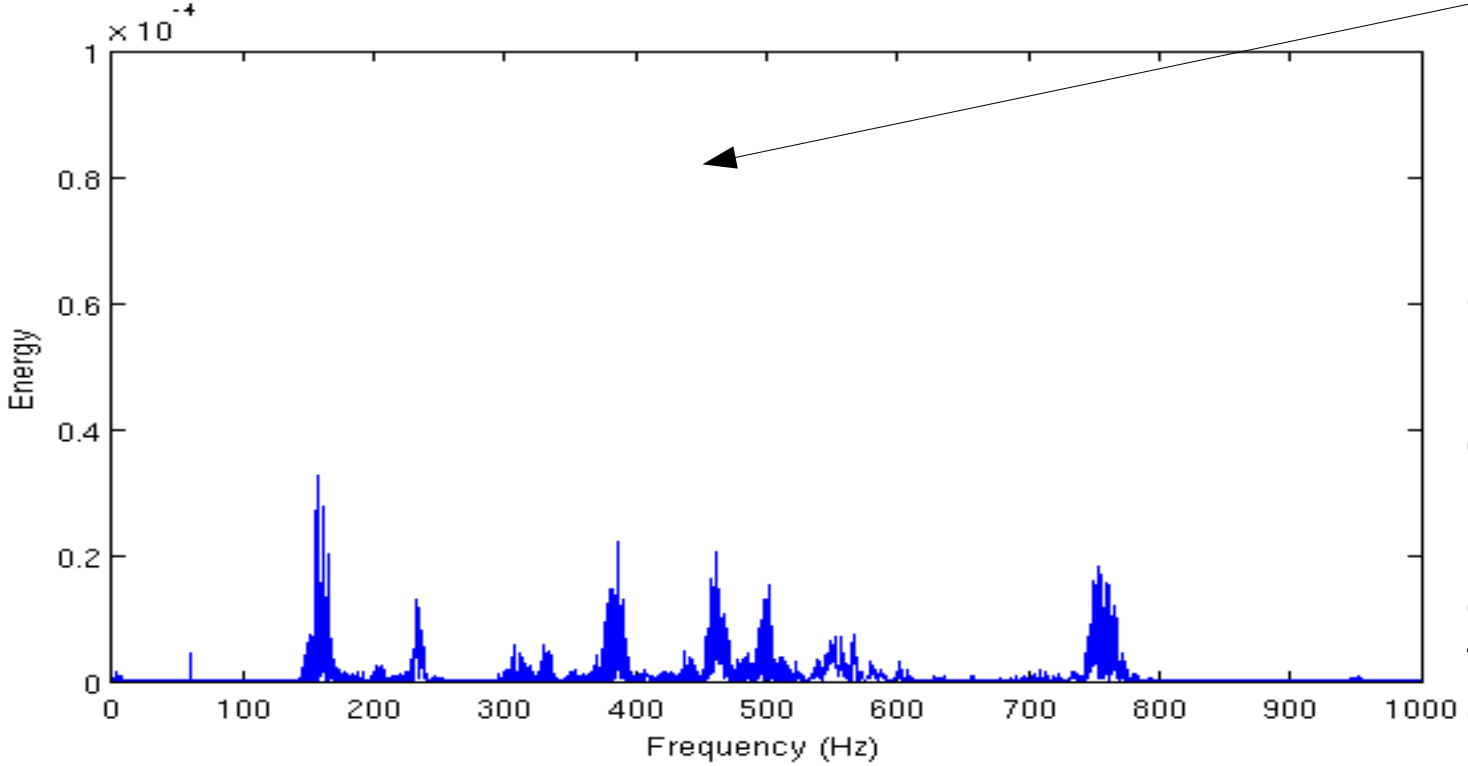


???

Señal de Ruido







Señal de Ruido Removida

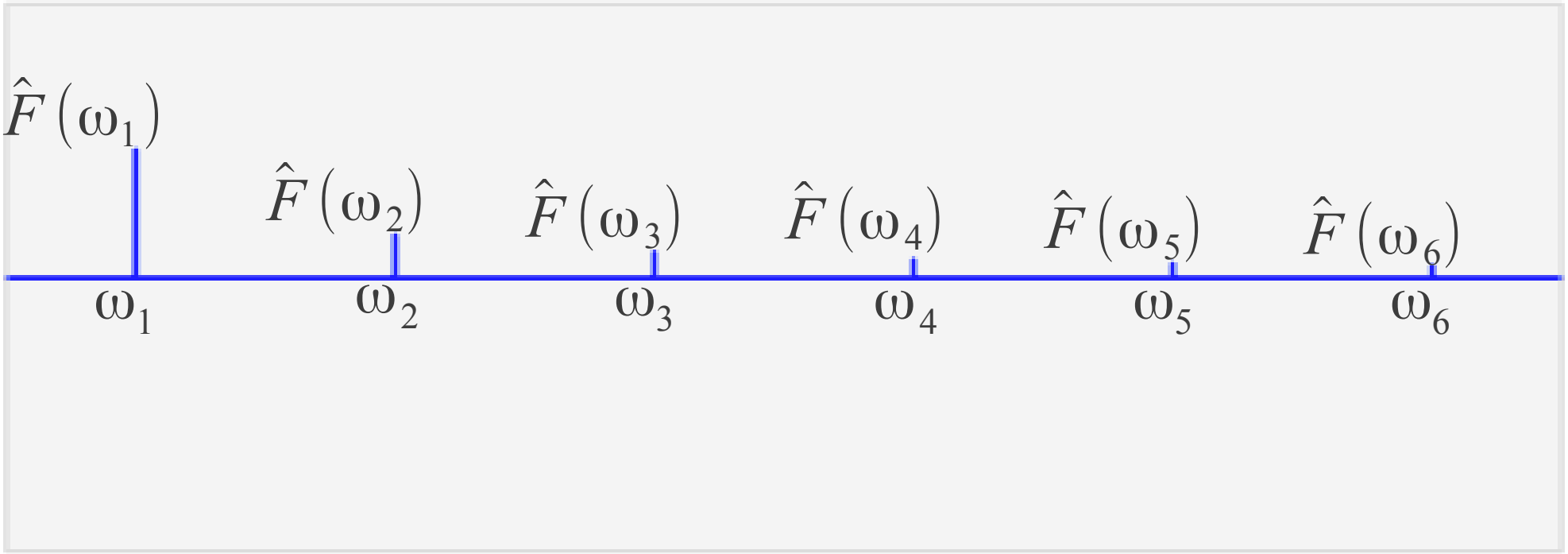
Por medio de hacer 0 a la magnitud de la señal a esa frecuencia.

Filtrado por frecuencia o *filtrado espectral*.

La señal en el dominio de la frecuencia es también conocido como su **espectro**.

Representación de Ondas

- F es la señal f transformada al dominio de la frecuencia.
- ω_n son las frecuencias de las ondas que, al sumarse, representan a f.
- $F(\omega_n)$ es la magnitud de la onda que oscila con frecuencia ω_n.



$\hat{f}$

Representación de Ondas

- Una onda puede ser representada con un coseno, con un seno, o con un exponencial, ya que la fórmula de Euler indica que:

$$e^{ix} = \cos(x) + i\sin(x)$$

- Donde **e** es el número Euler, que es la base del logaritmo natural, y se cálcula con la siguiente serie:

$$e = 1 + \frac{1}{1} + \frac{1}{1*2} + \frac{1}{1*2*3} + \frac{1}{1*2*3*4} \cdots$$

- Se puede redondear a 2.71828.

Representación de Ondas

- Representar una onda con exponenciales nos brinda la posibilidad de simplificar las ecuaciones bastante.
- Por ejemplo, podemos representar una onda s_{f1} que oscila a una frecuencia ω_1 y con una magnitud c_{f1} así:

$$s_{f_1}(t) = c_{f_1} \mathrm{e}^{i\omega_1 t}$$

¿Qué es ω?

- Es la “frecuencia angular”.
- Recordemos que el exponencial es realmente una sumatoria de un coseno y un seno que reciben como entrada un ángulo.
- La relación entre la “frecuencia angular” (ω) y una frecuencia en Hertz (ζ) es:

$$\omega = 2\pi\zeta$$

Cálculo de la Transformada de Fourier

La Transformada es una Sumatoria

- Una señal discreta se puede considerar como un arreglo de muestras de energía.
- Por lo tanto, según Fourier, deberíamos de poder representar cualquier momento en el tiempo de una señal como *una sumatoria de las energías en ese momento de otras señales periódicas*.

Mapeo Formal de Frecuencia a Tiempo

$$f(t) = \sum_{n=1}^{N} \hat{F}(\omega_n) \, e^{i \omega_n t}$$

Donde:

- *t* es un momento en el tiempo en segundos
- *f(t)* es la energía de la señal en el momento *t*
- *N* es el número de señales periódicas con las cuales queremos representar a *f(t)*
- ω_n es la frecuencia de la señal *n*.

¿Mapeo?

- El valor *f(t)* es cálculado con todos los valores del dominio de la frecuencia $F(\omega_n)$. Esto significa dos cosas:
  - El tamaño de la señal en el dominio de la frecuencia **es la misma** que en el dominio del tiempo.
  - Y como todo mapeo, se puede hacer en dirección inversa: calculando un $F(\omega_n)$ con todos los valores del dominio del tiempo *f(t)*.

Mapeo Formal de Tiempo a Frecuencia

$$\hat{F}(\omega_n) = \sum_{t=0}^{T} f(t)\, \mathrm{e}^{-i\,\omega_n t}$$

Donde:

- t es un momento en el tiempo en segundos
- $f(t)$ es la energía de la señla en el momento t
- T es el tiempo final de $f(t)$
- ω_n es la frecuencia de la señal n.

Versión Contínua

- Lo que estamos viendo realmente es la Transformada *Discreta* de Fourier.
  - Ya que es la que nos es más relevante.
- Recordatorio: una sumatoria de los valores de un señal es equivalente a una integral (área bajo la curva).
- Por lo tanto, la versión Contínua de esta transformada (la original) se puede obtener substituyendo las sumatorias por integrales.

Versión Contínua

$$f(t) = \int \hat{F}(\omega) \, \mathrm{e}^{i \omega t} \, d\omega$$

$$\hat{F}(\omega) = \int f(t) \, \mathrm{e}^{-i \omega t} \, dt$$

Problemas que Considerar

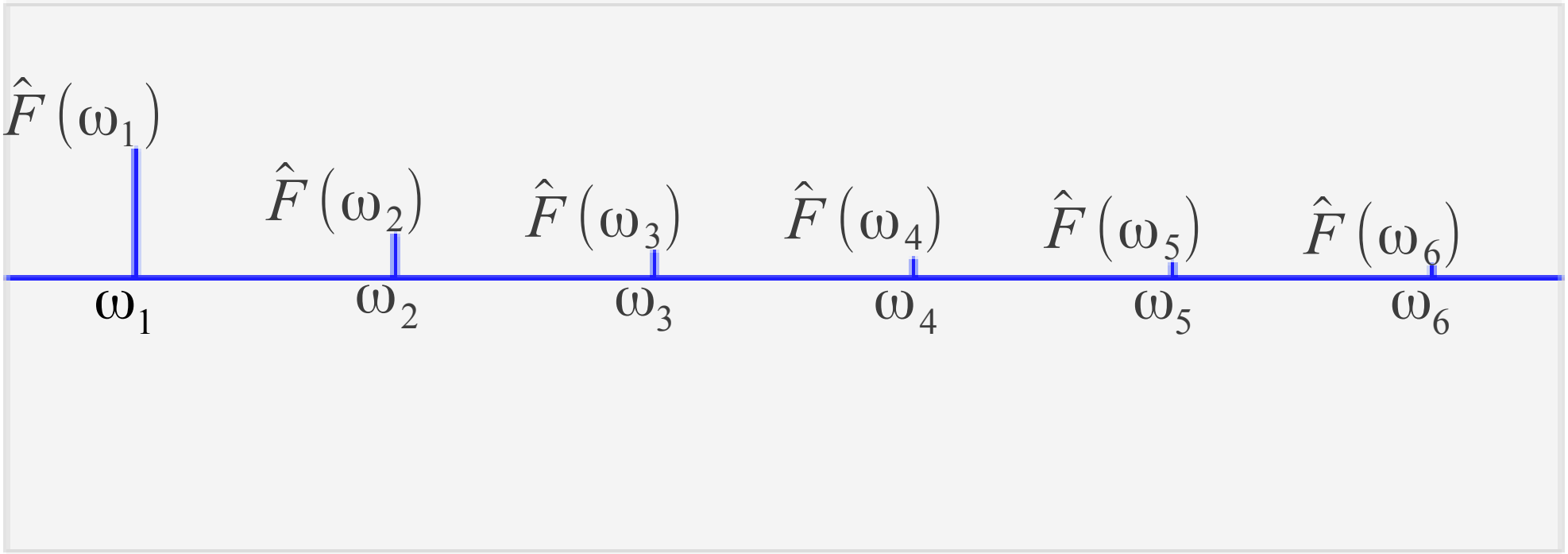
Problema #1

- **¿Con cuáles frecuencias representamos a una señal?**
- Harry Nyquist y Claude Shannon introdujeron un teorema que dice que: “para determinar completamente una señal con un ancho de banda conocido, se debe muestrear con una onda que tenga ***una frecuencia mayor al doble de su frecuencia máxima***”.

Problema #1

- **Mejor dicho:**
  - “Si una señal es muestreada con una frecuencia ω_s, entonces la frecuencia máxima que puede ser representada en el dominio de Fourier es $\omega_s/2$”.

- Tenemos una señal f muestreada de 48000 Hz.
- Si ω_6 es su máxima frecuencia, entonces ésta debe ser 24000*(2π).
- El resto de las ω_n están espaciadas uniformemente de ahí para atrás.



$$\hat{f}$$

Problema #2

- Si la formula de Euler es:

$$e^{ix} = \cos(x) + i \sin(x)$$

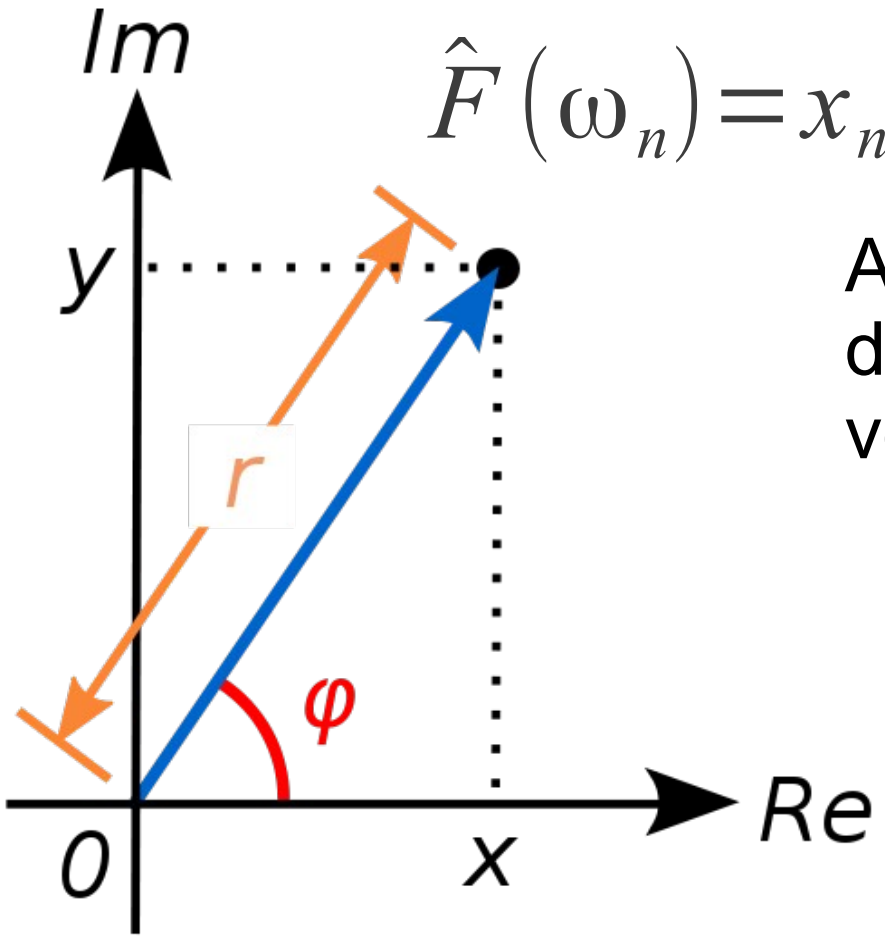
- Y la transformada dice que:

$$\hat{F}(\omega_n) = \sum_{t=0}^{T} f(t) e^{-i \omega_n t}$$

- Entonces $F(\omega_n)$ es un número complejo.

Números Complejos

- Un número complejo es la sumatoria de dos valores: uno real y uno imaginario.



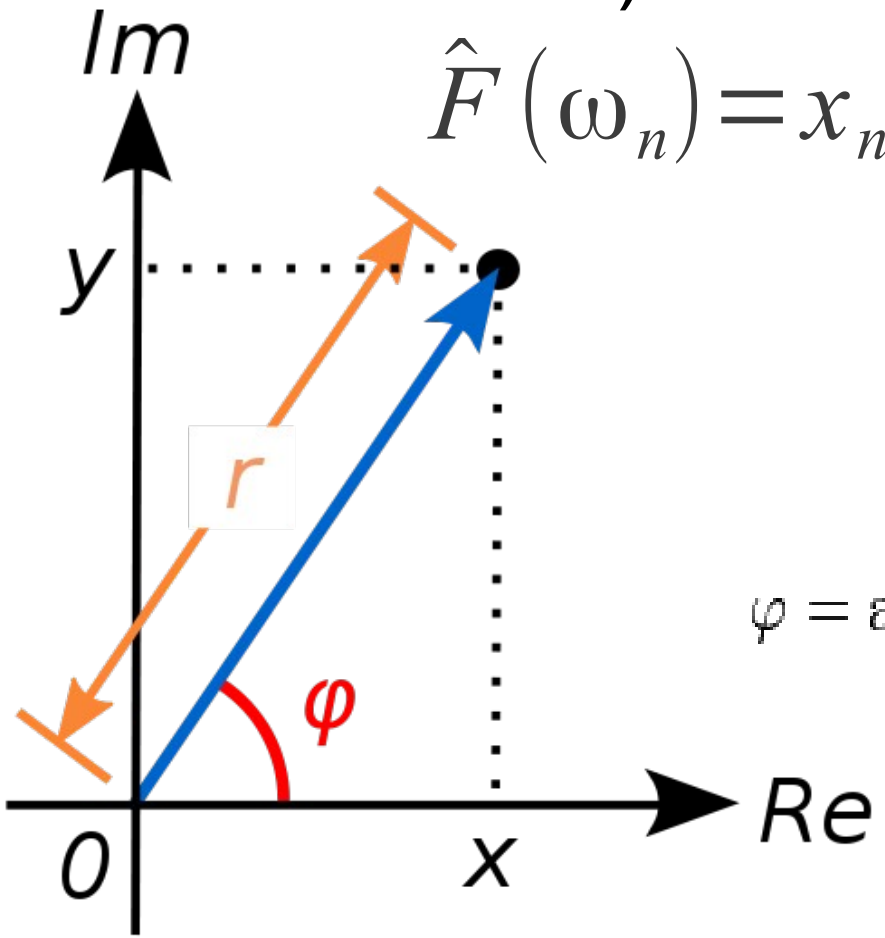
$$\hat{F}(\omega_n) = x_n + iy_n$$

Al visualizar la señal en el dominio de frecuencia, usualmente lo que vemos es la magnitud de $F(\omega_n)$:

$$\left|\hat{F}(\omega_n)\right| = \sqrt{x_n^2 + y_n^2}$$

Números Complejos: Fase

- La “fase” de la frecuencia es el ángulo del número complejo (fácilmente calculado con atan2 en C).



$$\hat{F}(\omega_n) = x_n + iy_n$$

$$\varphi = \arg(z) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ \text{indeterminate} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

Números Complejos

- Es crucial recordar esto al estar manipulando las magnitudes en tiempo real.
- Se pueden hacer restas, multiplicaciones, etc. pero se tienen que hacer considerando que dichas operaciones serán sobre números complejos.

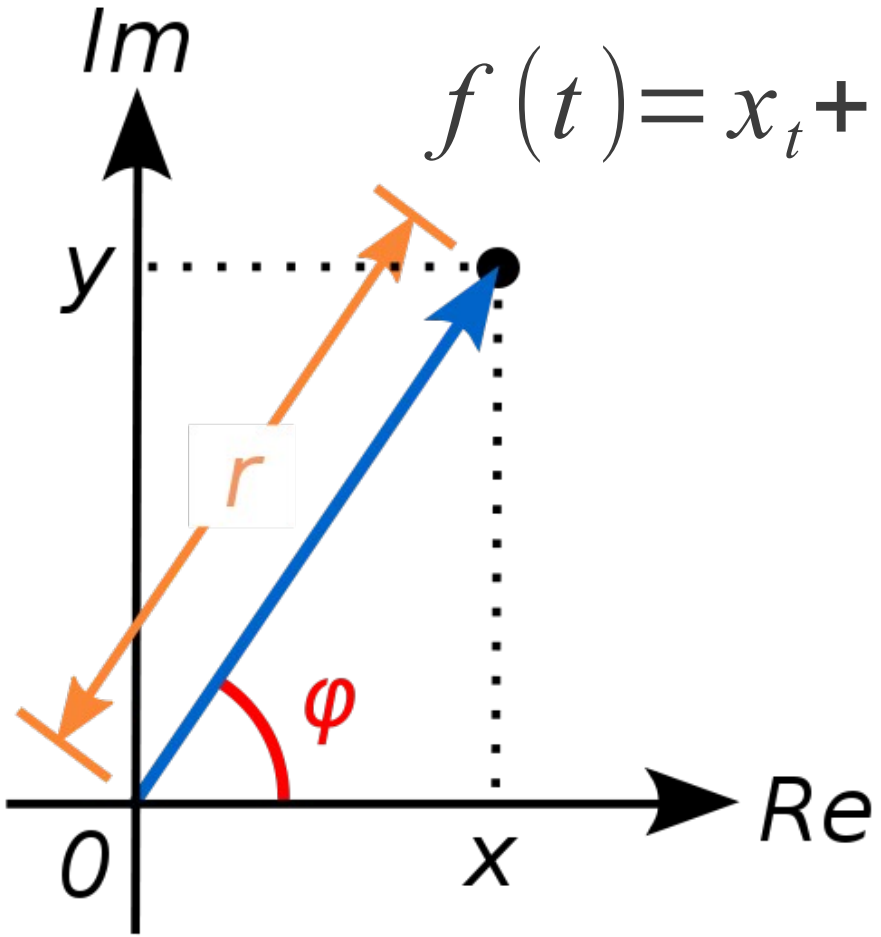
Problema #2.5

- El resultado de la Transformada Inversa también es un número complejo:

$$f(t) = \sum_{n=1}^{N} \hat{F}(\omega_n) \, \mathrm{e}^{i \omega_n t}$$

Problema #2.5

- No es un problema realmente, ya que la energía en el tiempo se presenta en la parte real del número calculado:



$$f(t) = x_t + iy_t$$

$$f(t) \leftarrow \Re(f(t)) = x_t$$

Por lo que el resultado final, realmente es un número real, común y corriente.

Problema #3

- **MITO:** Si el tamaño de la señal en tiempo es la misma que la señal en frecuencia, entonces una señal de 1024 puntos en el tiempo tendrá 1024 puntos de frecuencia.

- Cierto y falso. Sí, serán 1024 puntos de frecuencia, pero no todos representan a una frecuencia única.

La Transformada Inversa

- Habíamos dicho que la transformada inversa es:

$$f(t) = \sum_{n=1}^{N} \hat{F}(\omega_n) \, e^{i \omega_n t}$$

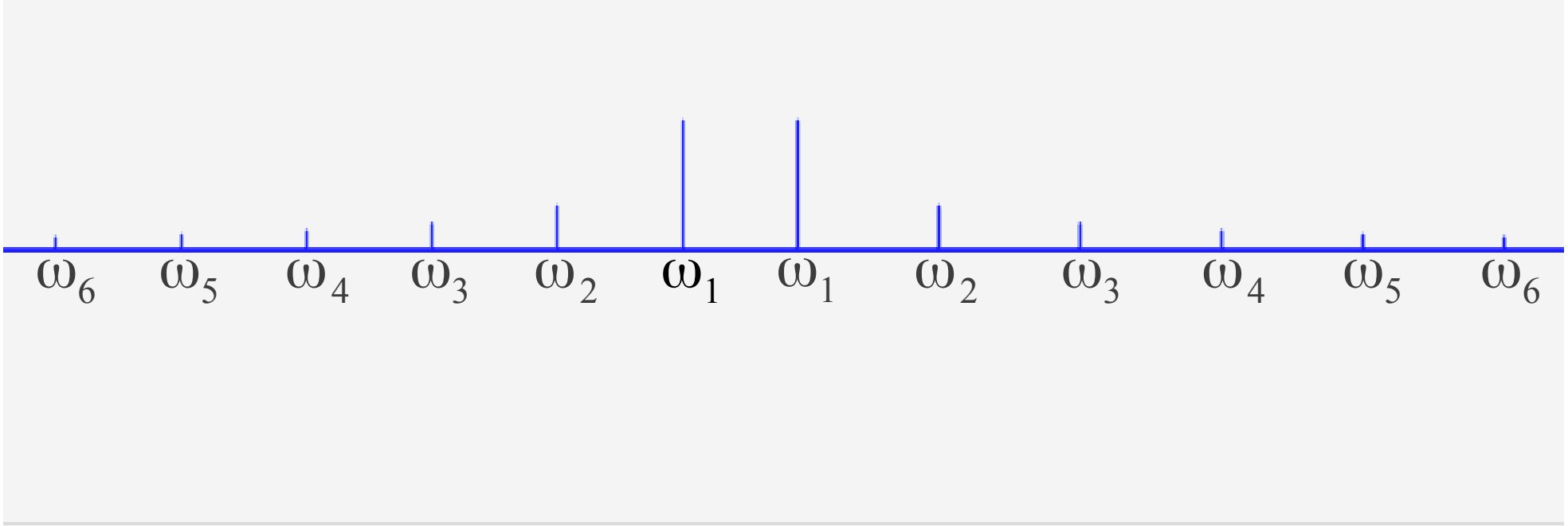
La **Real** Transformada Inversa

- La verdadera transformada inversa es la siguiente:

$$f(t) = \sum_{n=-N}^{N} \hat{F}(\omega_n) \mathrm{e}^{i\omega_n t}$$

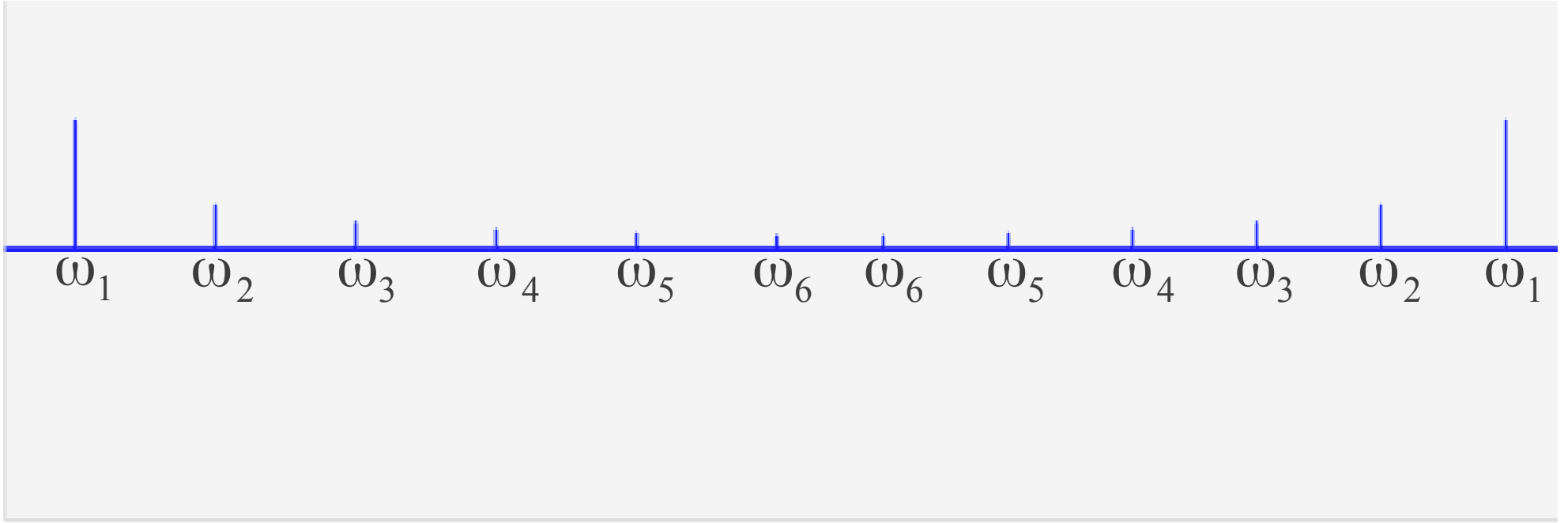
- Sí, hay tal cosa como “frecuencias negativas”, porque Fourier quiso realmente fregar nuestras mentes.

El Resultado de la Transformada de Fourier



Este espejeo tiene que ver con el cálculo de las partes imaginarias y las partes reales de la Transformada de Fourier y los famosos “Círculos de Harmónicos”, ilustrados brevemente en el video “FourierCircles.mp4” de la página del curso.

Lo que normalmente se nos entrega en las implementaciones de Fourier:

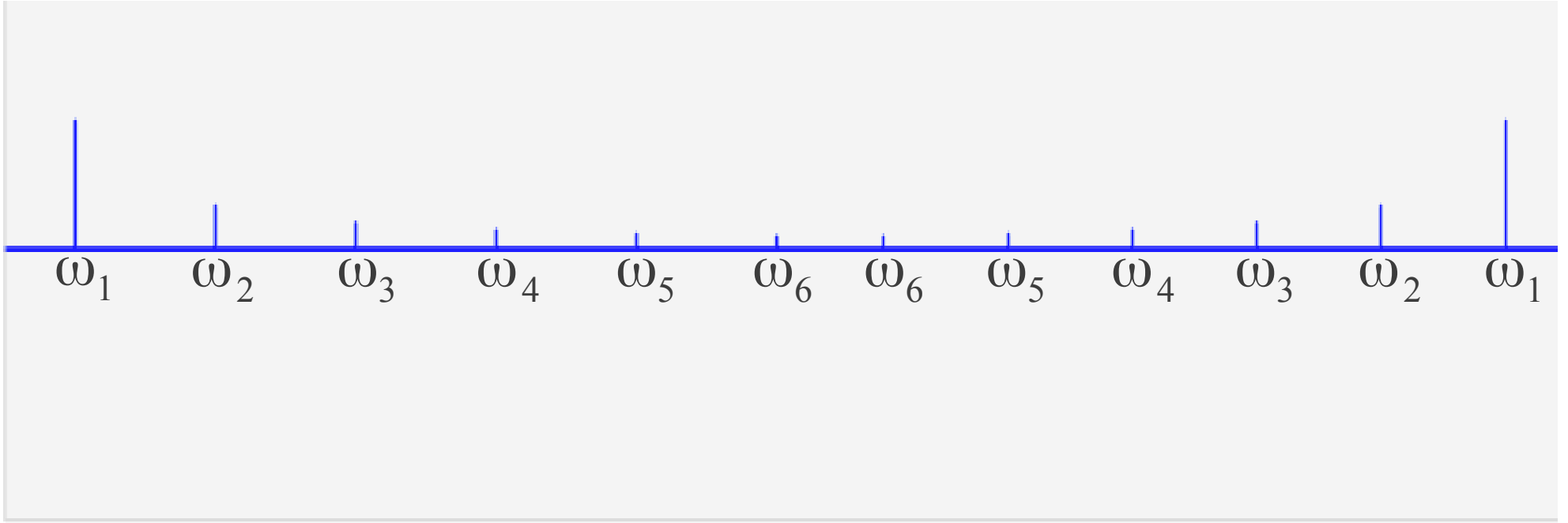


No sé por qué, honestamente.

Supongo que es más sencillo manejar los índices de al principio del vector.

En cualquier caso:

Esto es con lo que vamos a trabajar:



CONCLUSIÓN: El resultado de transformar una señal de 1024 puntos en el tiempo es una señal con 1024 puntos en frecuencia "espejeados", realmente calculando 512 puntos útiles.

POR LO TANTO: al manipular el valor en una frecuencia, se tiene que manipular, de la misma manera, su **contraparte en el "espejo"** de la señal en el dominio de la frecuencia.

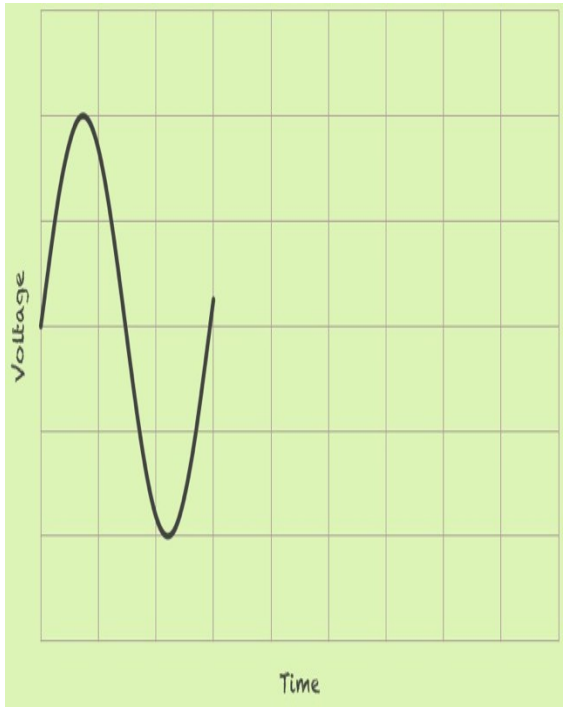
Problema #4

- Fourier propuso que “varias ondas sumadas entre si pueden ser utilizadas para representar cualquier señal *periódica*.”
- **¿Qué sucede si nuestra señal no es periódica?**

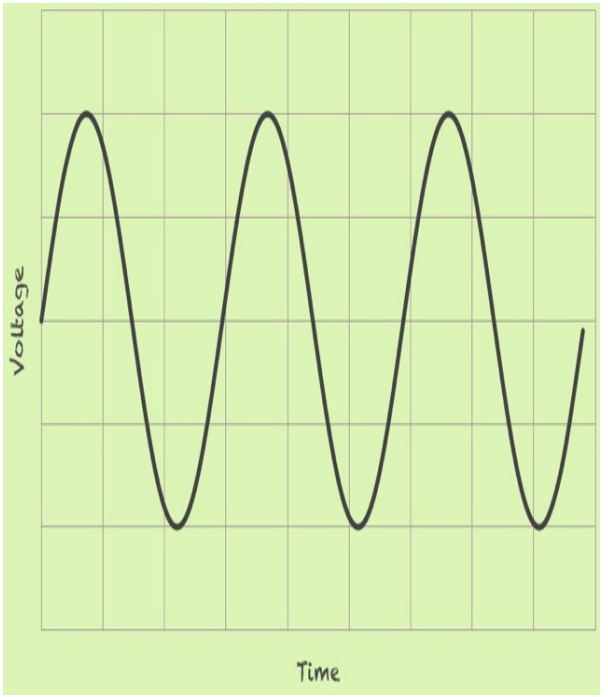
Problema #4

- Hacemos trampa: suponemos que nuestra señal tiene un periodo del *tamaño de la señal*.
- Para que esto funcione adecuadamente, tenemos que cerciorarnos que nuestra señal comience y termine *con valores no discontínuos*.

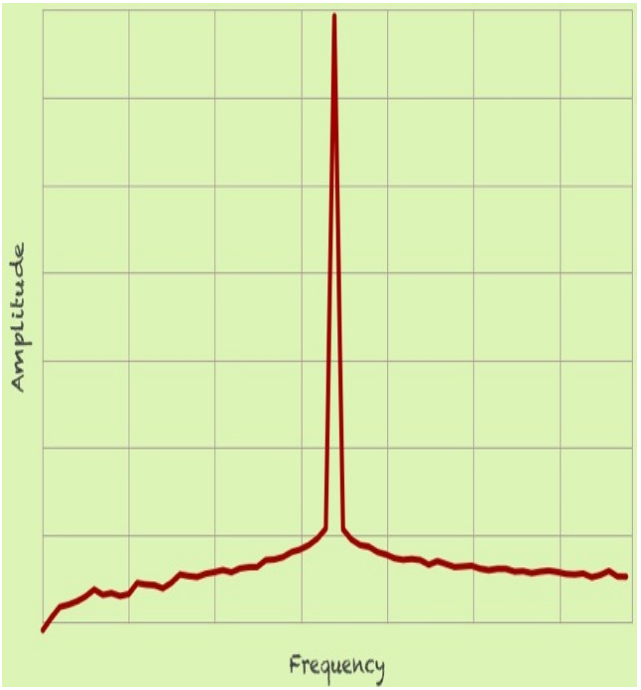
Problema #4



Señal

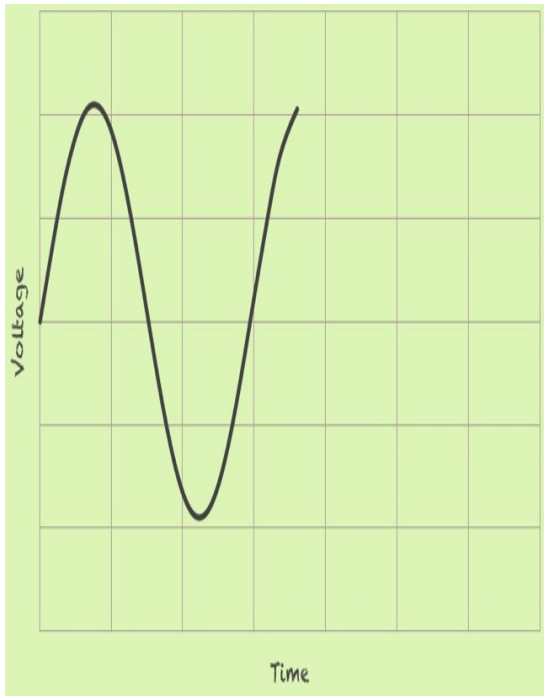


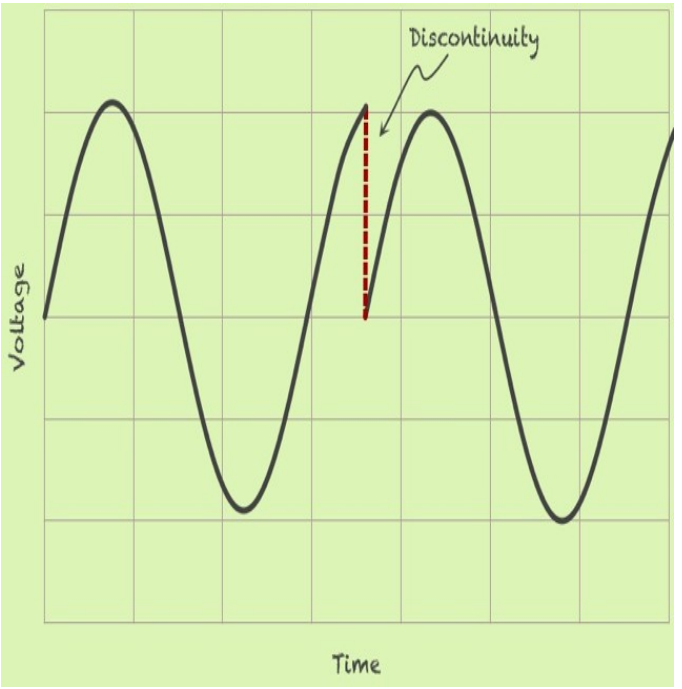
Lo que la Transformada ve

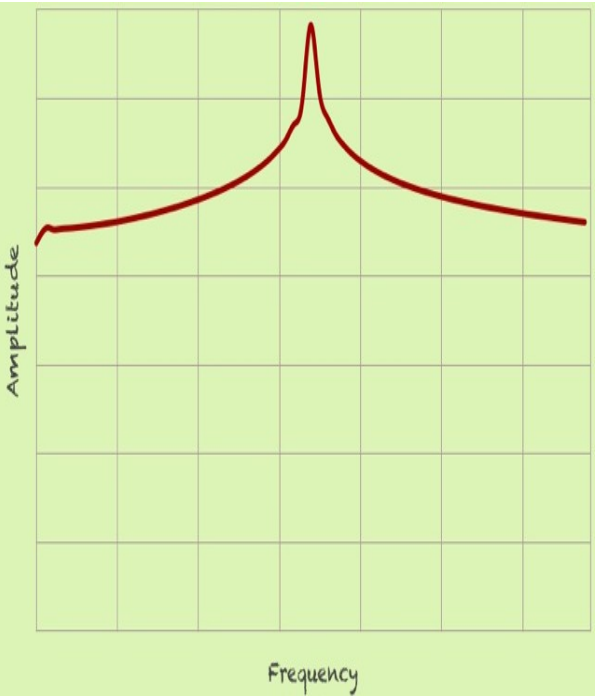


Resultado

Problema #4





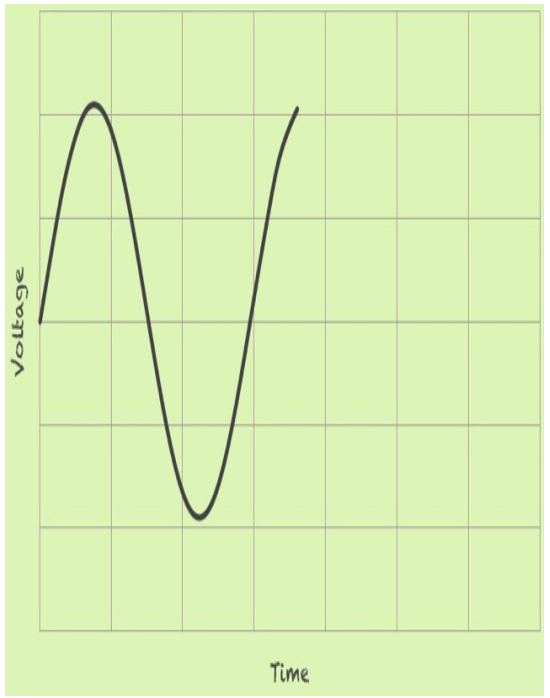


Señal

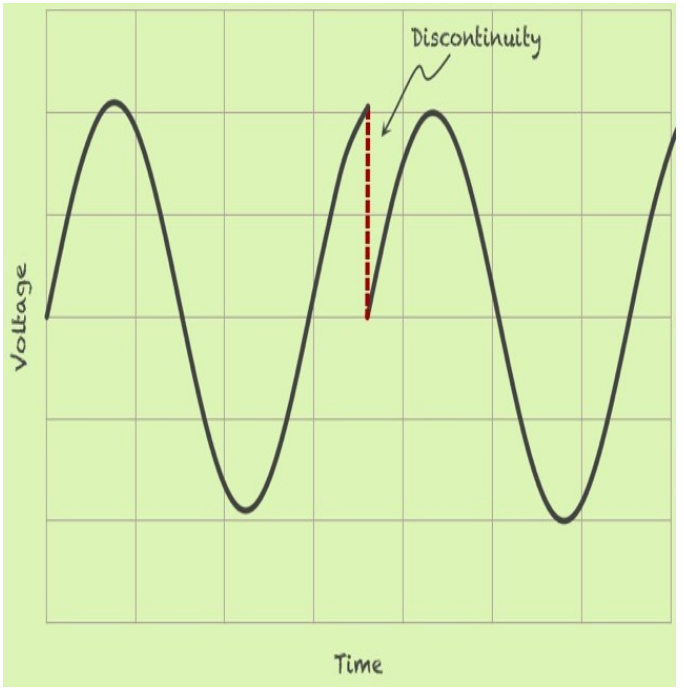
Lo que la Transformada ve

Resultado

Problema #4

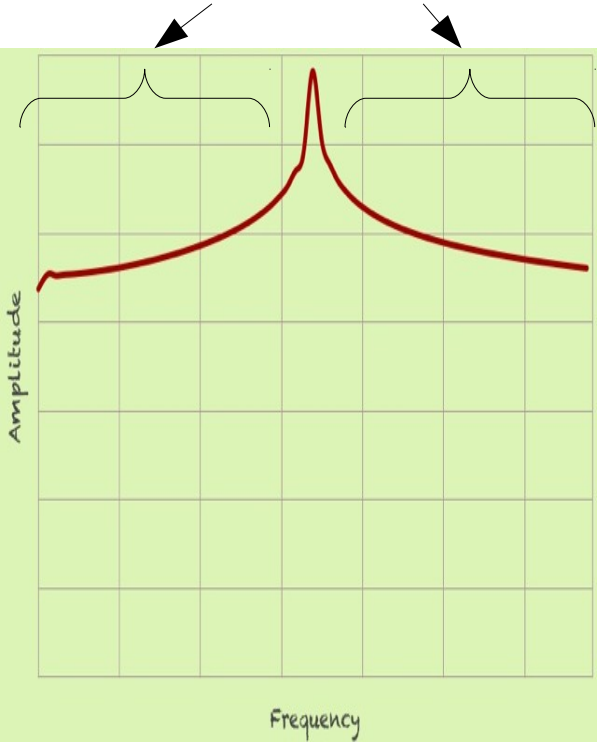


Señal



Lo que la Transformada ve

Sangrado de Frecuencias
(*Frequency Bleed Over*)



Resultado

¿Sangrado de Frecuencias?

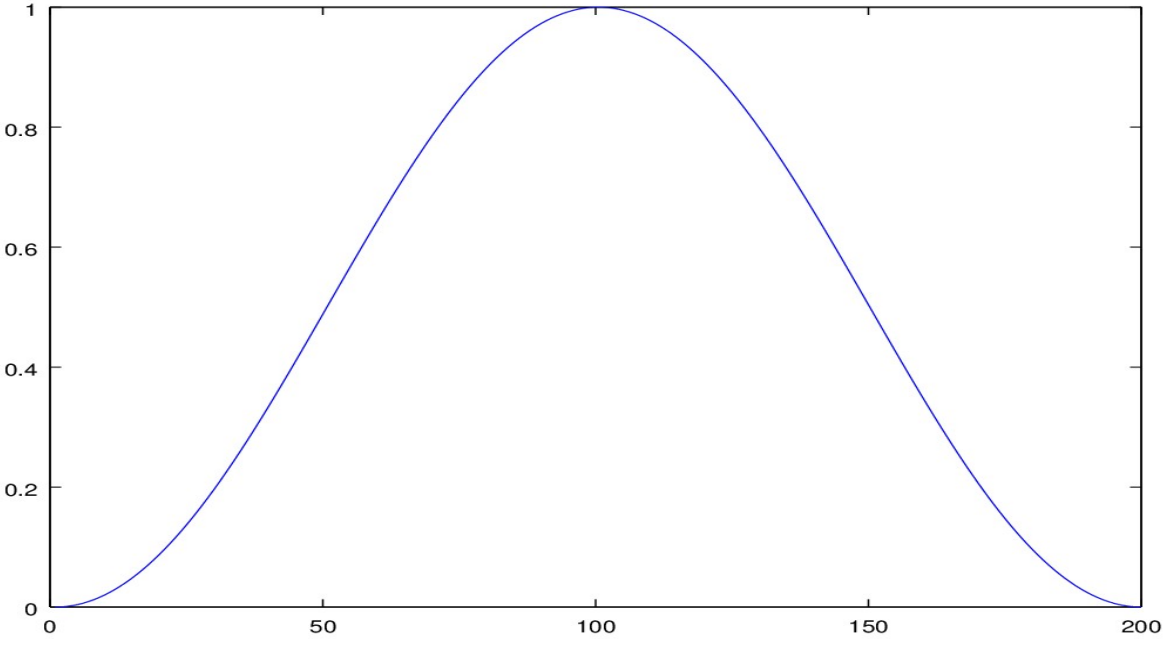
Es la mejor traducción que se me ocurrió a **Frequency Bleed Over**.

- Básicamente, se requiere de la presencia de otras señales periódicas para poder representar discontinuidades.
- Esto no es bueno, porque se insertan frecuencias que no están en nuestra señal original.

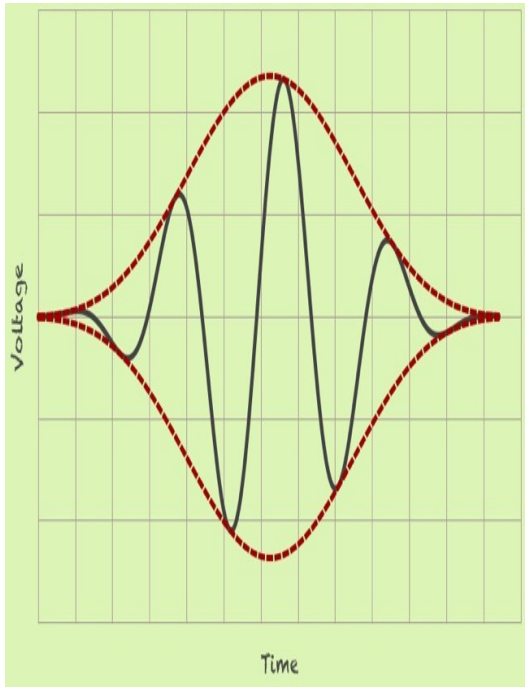
Hann

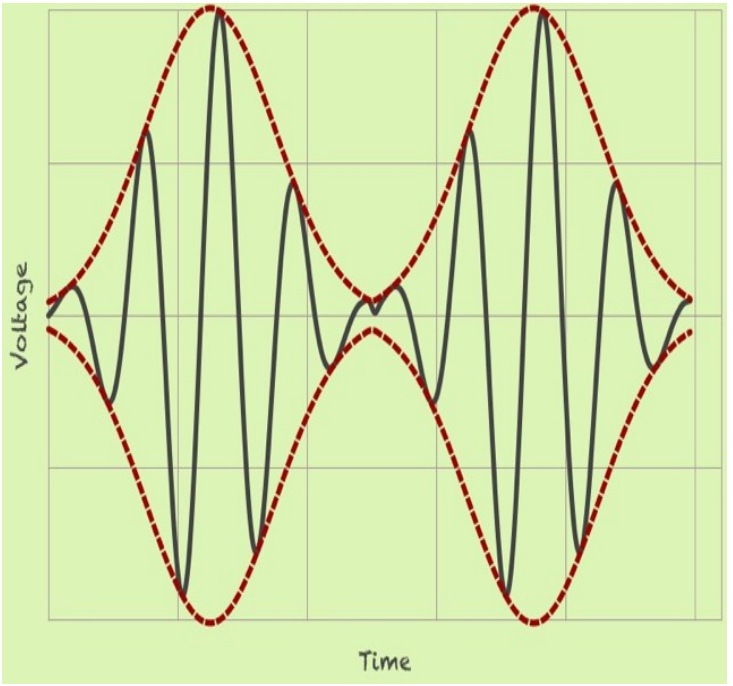
- Hay varias formas de hacer continuos los puntos iniciales y finales de la señal.
- **Forma Popular:** multiplicando, punto a punto, a la señal por la función Hann.
- En este ejemplo, *T* es 200.

$$w(t)=0.5\left(1-\cos\left(\frac{2\pi t}{T-1}\right)\right)$$



Problema #4





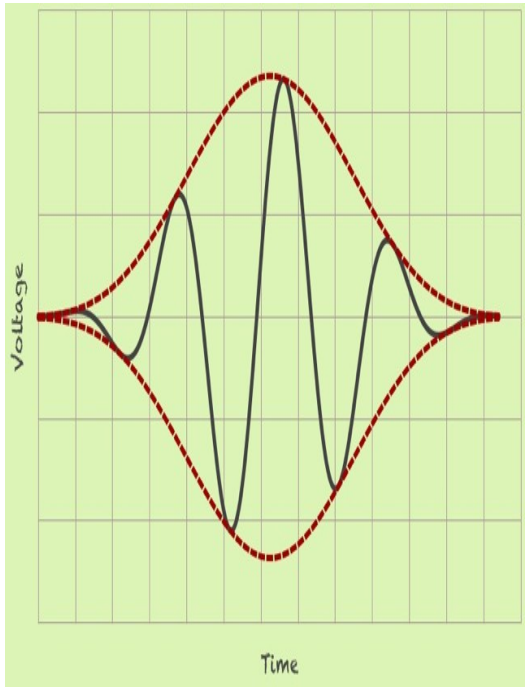
Señal

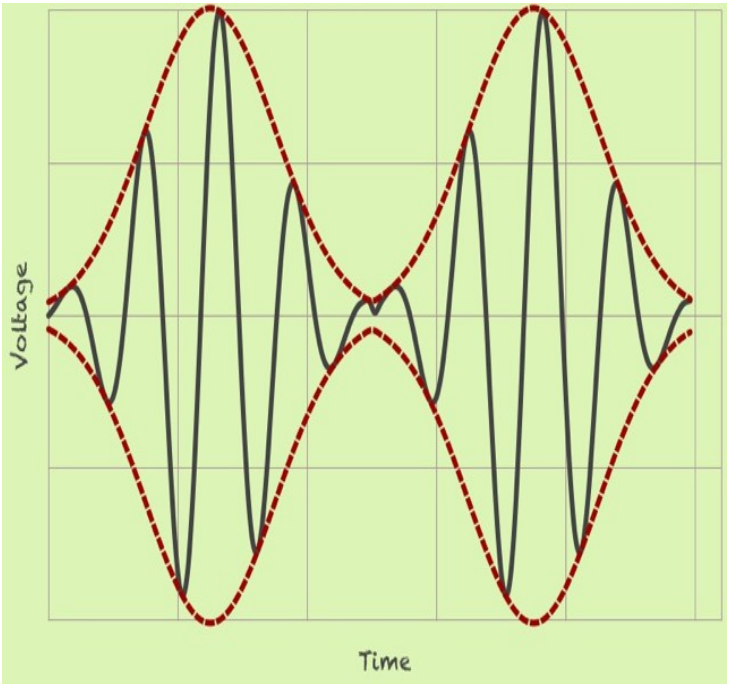
Lo que la Transformada ve

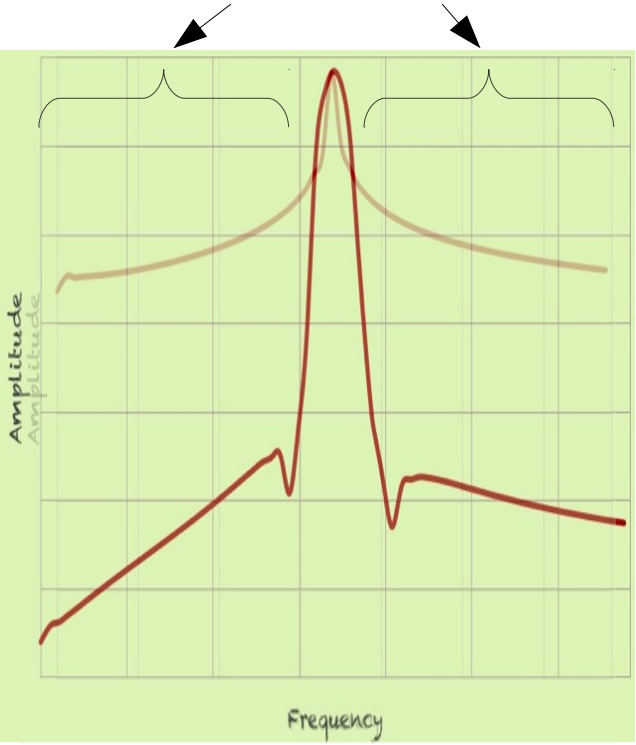
Resultado

Problema #4

El sangrado se minimiza







Señal

Lo que la Transformada ve

Resultado

Problema #5

- Si *T* en la ecuación de la Transformada es el "tiempo final de la señal en el dominio del tiempo":

$$\hat{F}(\omega_n) = \sum_{t=0}^{T} f(t)\, \mathrm{e}^{-i\omega_n t}$$

- Se necesita de toda la señal para llevar a cabo su transformación.
- ¿Y en tiempo real cómo le hacemos?

Transformada de Tiempo Corto

- En vez de transformar toda la señal, transformamos “trozos” de la señal.
- Estos trozos son conocidos como *ventanas.*
- Se transforman una tras otra, como vayan llegando.
- Así *T* se convierte en el “tiempo final de la ventana”.

Transformada de Tiempo Corto

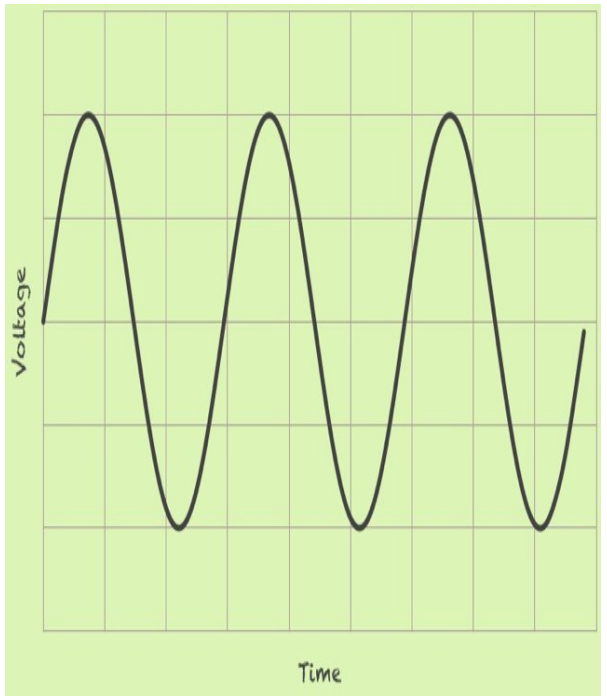
- Uno estaría tentado a solamente partir la señal en dichas ventanas, transformar cada ventana, modificar la ventana en el dominio de la frecuencia, y transforma en inversa.
- Pero, recuerden el Problema #4: tenemos que aparentar como si la señal (o, en este caso, la ventana de la señal) fuera periódica:
  - Hacer que los puntos del tiempo inicial y final no sean discontinuos

Problema #5.5

- Entonces, multiplicamos por Hann cada ventana, la transformamos, la modificamos en frecuencia, y la transformamos de inversa.

  ¿No?

- El resultado de esto es tener varias ventanas multiplicadas por Hann una tras otra.

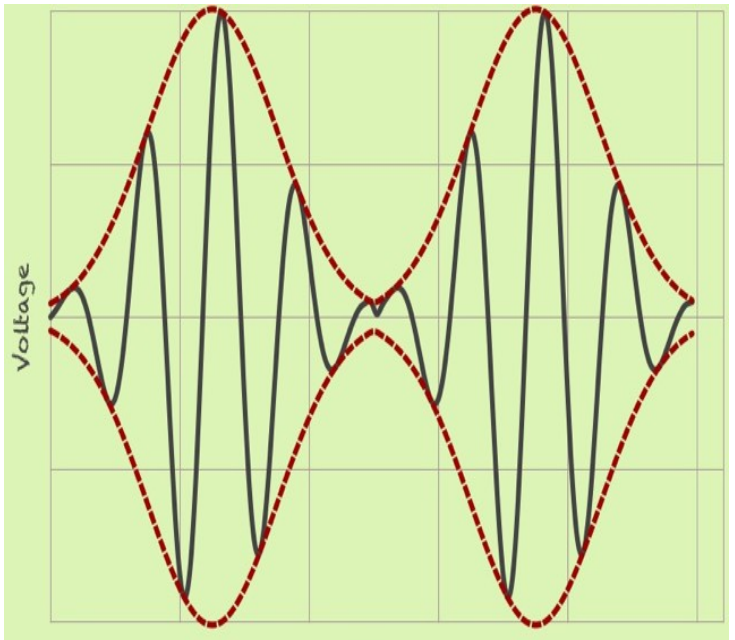


Ventaneo

FFT

Modificación en Frecuencia

IFFT

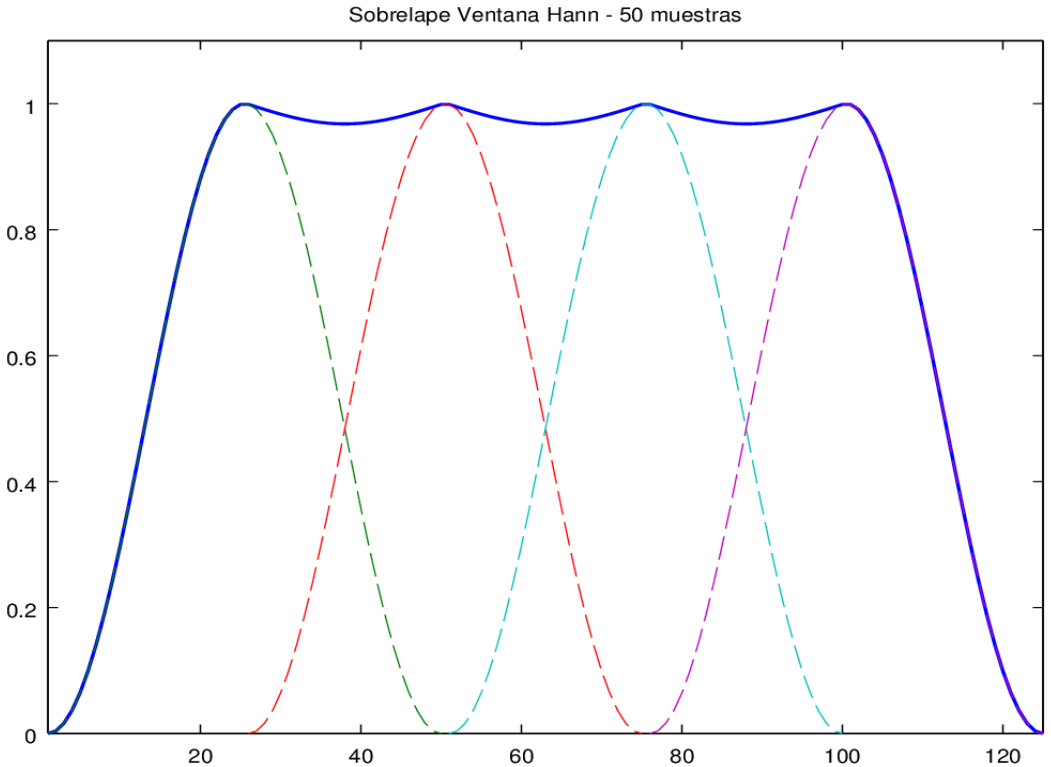


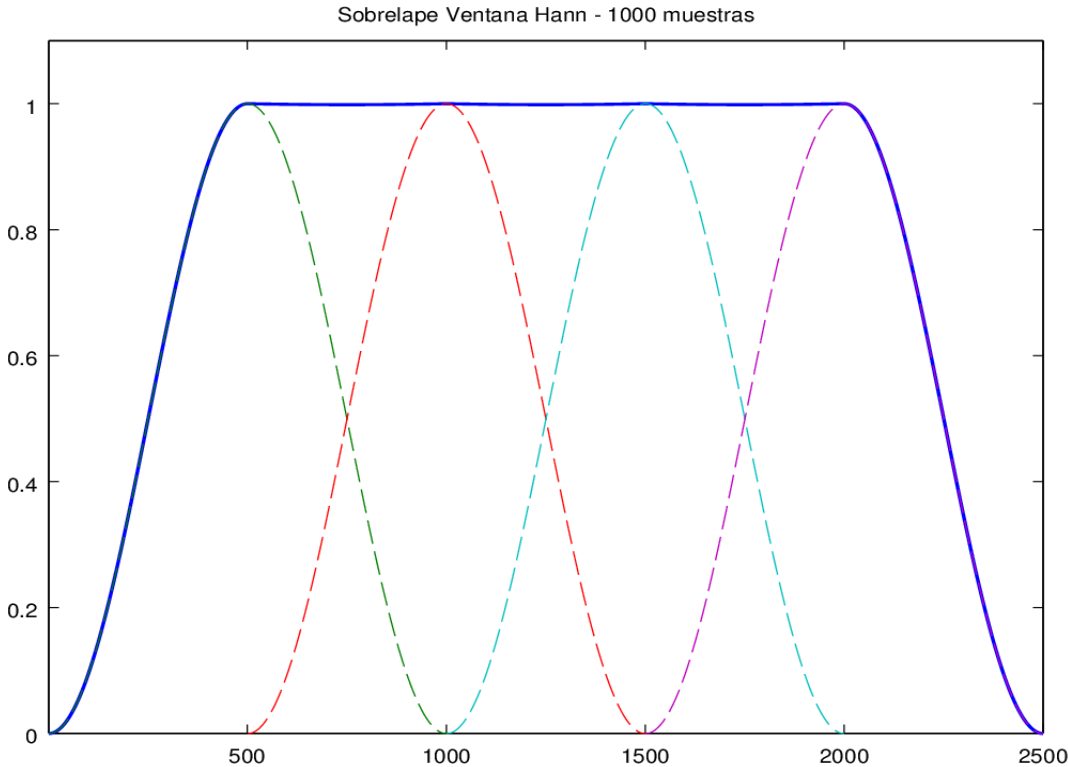
Problema #5.5

- Al hacer esto, tendremos una señal que sube y baja de volumen artificialmente.
- SOLUCIÓN: **sobrelape**.

Sobrelape

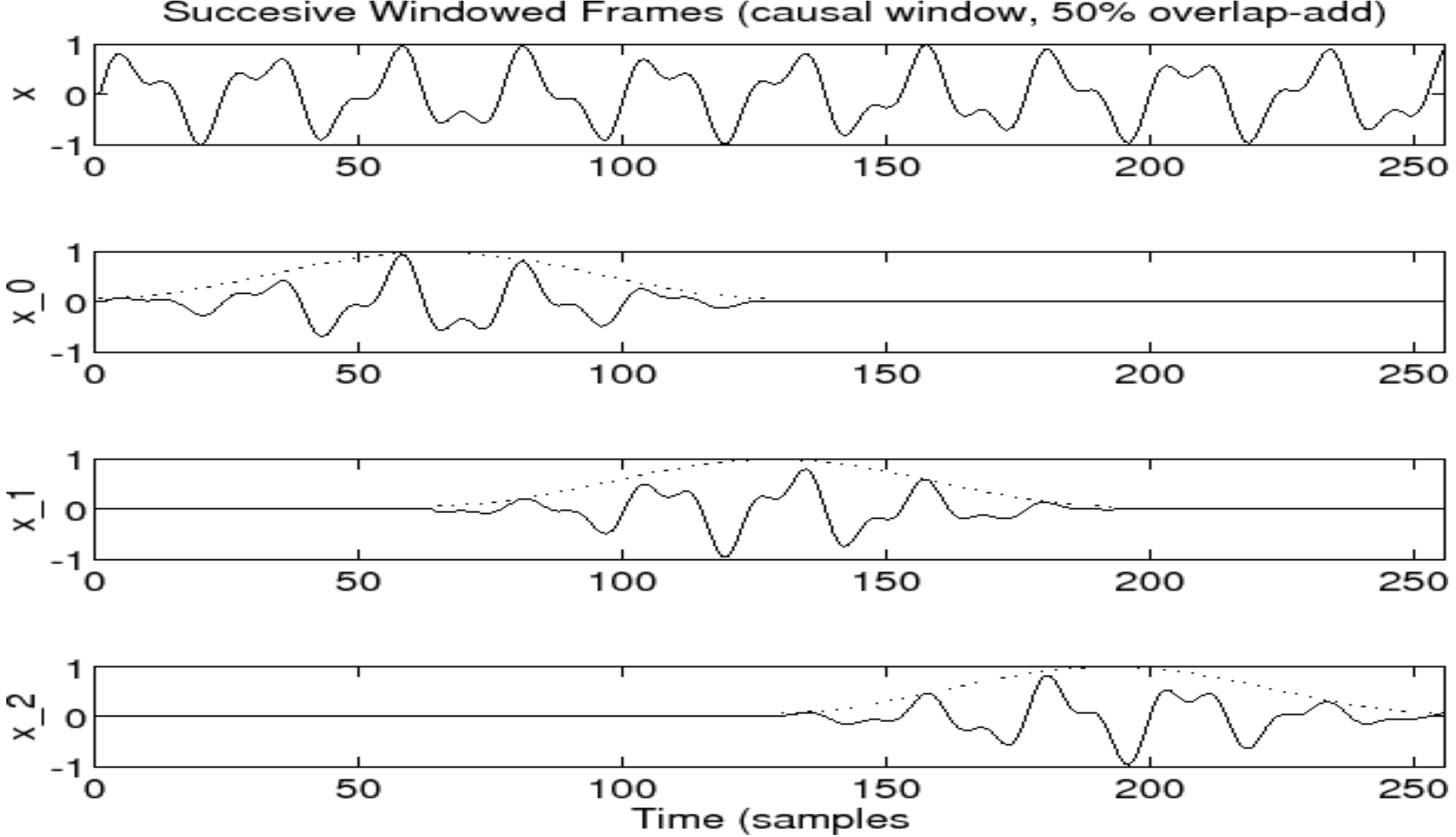
- Una virtud de una función Hann **larga** es que, al sumar la segunda mitad de la función con la primera, obtenemos una señal con una magnitud casi constante.



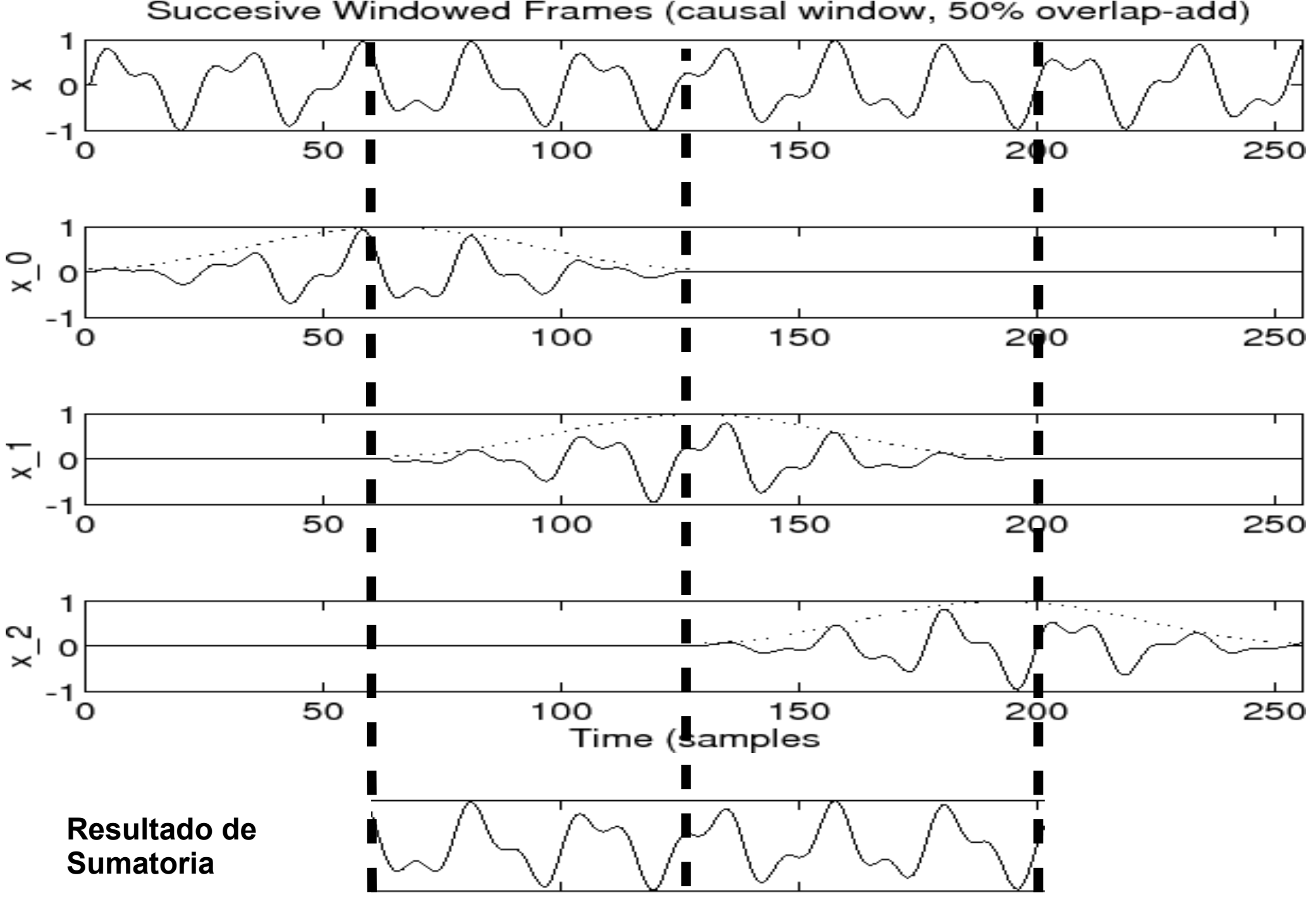


Sobrelape

- Normalmente ventanearíamos, con ventanas de tamaño *T*:
  - [1 T], [T+1 2T], [2T+1 3T], etc.
- Pero si ventaneamos de esta manera:
  - [1 T], [T/2+1 3T/2], [T+1 2T], [3T/2+1 5T/2] ,etc.
  - Nuestra ventanas siguen siendo de tamaño *T*, pero comenzamos cada *T/2*, en vez de cada *T*.



Esto es lo que veíamos en baudline, en la ventana del tiempo.



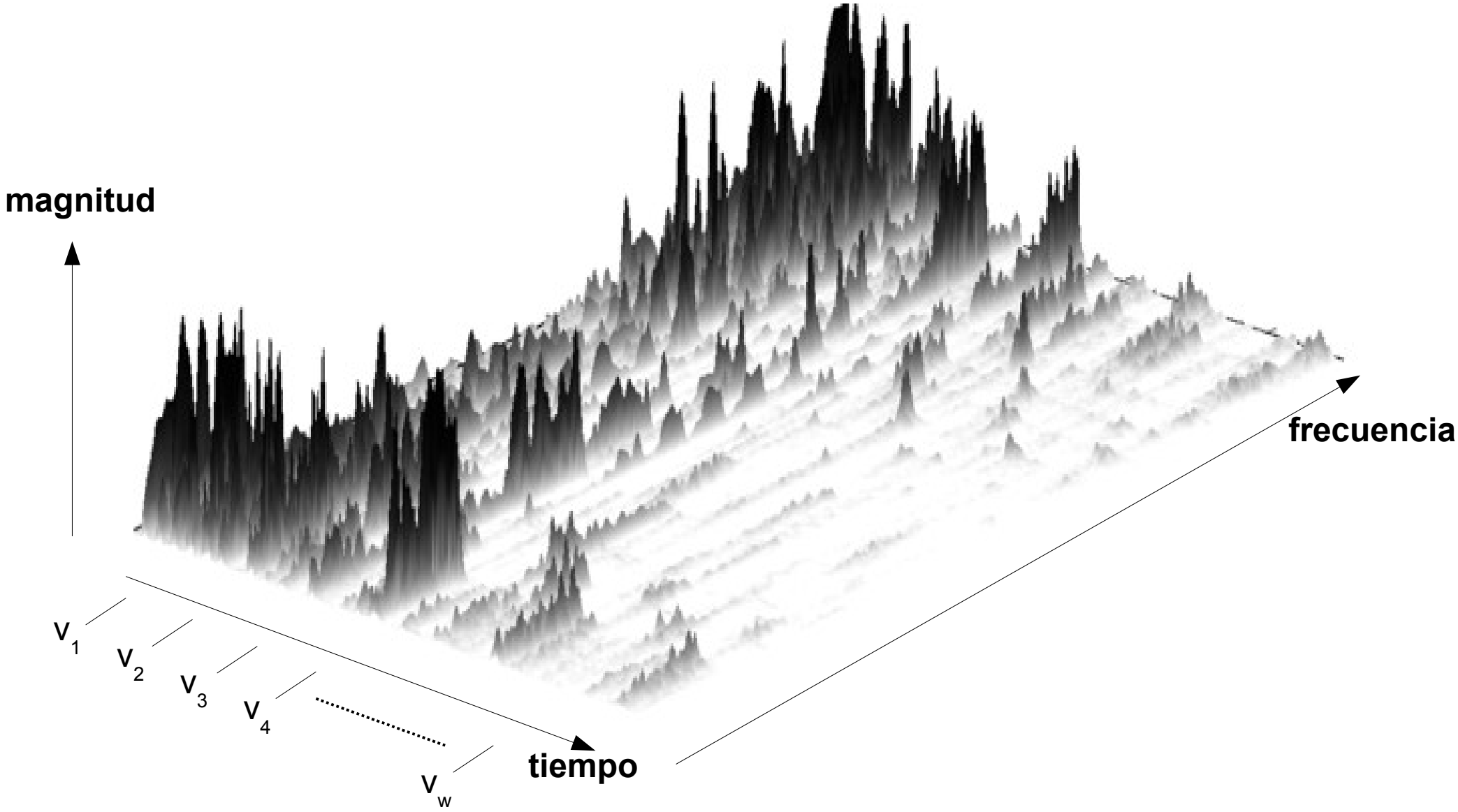
Resultado de Sumatoria

Sobrelape

- Este proceso se le conoce como “Sobrelapa-y-Suma” (*overlap-and-add*).
- Se puede con otro tipo de funciones como:
  - Función Hamming (con 50% sobrelape)
  - Función Blackman (con 33% sobrelape)
- Todos con sus pros y cons, pero Hann es una de las funciones más populares para llevarlo a cabo.

Esto significa que vamos a tener a una serie de ventanas de **tiempo** transformadas al dominio de **frecuencia**.

Espectrograma



Resolución en Tiempo vs. Resolución en Frecuencia

- Es importante hacer notar que el tamaño de la ventana impacta a las resoluciones de ambos dominios, pero de maneras opuestas:
  - Una ventana larga:
    - Más puntos de frecuencia para la transformada, por lo tanto, mayor resolución de frecuencia.
    - Menos ventanas disponibles (de una señal de tamaño finito), por lo tanto, menos resolución de tiempo.
  - Y viceversa.
- Ventanas entre 1024 y 2048 puntos de tiempo son un buen compromiso.

Ahora,

¿cómo programamos todo esto?

Nuevo mejor amigo: FFT de Librow

- Sus iniciales significan “Fast Fourier Transform”.
  - La transformada original suma señales para un coeficiente que se restan en el cálculo de otros coeficientes.
  - La FFT evita estas sumas redundantes, acelerando el cálculo al grado de poderse hacer en tiempo real.
    - Esta “aceleración” fue propuesta por Cooley y Tukey en 1965.
- Librow implementa esta “aceleración” en su librería FFT.
- Sitio Oficial:

  http://www.librow.com/articles/article-10

Otras Implementaciones de FFT

- FFTW3 *(mi favorita)*
  - Muy rápida y basada en C.
  - Implementación de números complejos compatible con la de C++.

    http://www.fftw.org/fftw3_doc/
- La función FFT de MatLab.
  - Estándar y muy probada.

    http://www.mathworks.com/help/matlab/ref/fft.html

¿Por qué la de Librow?

- Es muy portátil (4 archivos de código).
- Es pedagógica:
  - Toda la transformada y su inversa es una sóla función (*Perform*) de 20 líneas de código.
  - *Perform* hace sólo el mapeo; dependiendo de cómo están arreglados los datos, dicho mapeo puede ser la transformada o puede ser la inversa.
- Aunque la más rápida, es suficientemente rápida para nuestros propósitos.

FFT de Librow

- La librería se llama “fft.h”, y se tiene que compilar junto al programa que lo utiliza.
  - Se requiere hacer un makefile específico para esto.
- Hace uso de una librería para manipular números complejos, llamado “complex.h”.

FFT de Librow

- Está codificado en C++, por lo que:
  - Los archivos tendrán extensión “.cpp” en vez “.c”
    - Necesario para que gcc sepa que es C++ no C.
  - Se pueden hacer trucos de apartado de memoria (malloc):

    int a[20]; mismo que int *a = malloc(20*sizeof(int));
  - Al mandar llamar una función de la librería “fft.h” se tiene que preceder de CFFT:: (convención de C++).

Tamaño de Ventana

- Para que la aceleración funcione, es esencial que el tamaño de ventana sea una potencia exacta de 2:
  - $T = 2^x$, donde x es un número entero.
    - 1024 y 2048 satisfacen esta condición.
    - Además de que es compatible con las necesidades de sobrelape de la función Hann.

Tamaño de Ventana

- Si no cumple con esa condición, se requiere que los datos se “expandan” a potencia exacta de 2 más cercana y mayor, llenando con ceros los que sobran.
- Ejemplo:
  - Tamaño de ventana, *T*: 5 → [1 2 3 4 5]
  - Potencia exacta de 2 más cercana: 8
  - Nueva ventana con *T*:8 → [1 2 3 4 5 0 0 0]
- No le quita exactitud al proceso ni al ventaneo.

Ejemplo Básico

```
// Incluir el header de fft.h
#include "fft.h"

...Ya adentro de la función de procesamiento de audio
// Apartar memoria para señal
complex *pSignal = new complex[1024];
...aquí se llena a pSignal de datos de tiempo
// Aplicar la FFT
CFFT::Forward(pSignal, 1024);
...pSignal está ahora en el dominio de la frecuencia

// No olviden liberar memoria al terminar
delete[] pSignal;
```

Ejemplo Completo

```
// “in” es nuestra entrada en el tiempo
// y “out” es nuestro salida en el tiempo

// convirtiendo al dominio de frecuencia
complex *pSignal = new complex[1024];
for(int i = 0; i<1024;i++){
     pSignal[i] = in[i];
}
CFFT::Forward(pSignal, 1024);

pSignal[5] = 0;                 //filtrando la quinta frecuencia
pSignal[1024-1-5] = 0;      //espejeando dicha manipulación

// convirtiendo de reversa al dominio del tiempo
CFFT::Inverse(pSignal, 1024);
for(int i = 0; i<1024;i++){
   out[i] = pSignal[i].re();       //el resultado en tiempo es sólo la parte real
}

// Liberando memoria
delete[] pSignal;
```

Ejemplo Completo

- En la página del curso está el ejemplo completo en el que una ventana (o, periodo) de JACK es convertida y regresada al tiempo sin hacerle nada y sin ventaneo.
  - Se llama jack_fft.cpp
  - También descarguen a fft.cpp, fft.h, complex.cpp, complex.h, y makefile a una carpeta aparte.
- Compilen y verifiquen que funciona.

Compilación

- Se ha creado, además del programa “jack_fft”, un archivo extra llamado “fft.o”.
- Esta es la librería que también se compila al hacer “make”.
- Esta librería es la que contiene todo lo que hemos platicado y tiene que estar en el mismo directorio que el programa.

Compilación

- El makefile tiene algo que no hemos visto antes:
  - all: <algo>
  - Make tiene la posibilidad de revisar si algún archivo ha cambiado desde la última compilación.
    - En este caso, fft.o
- Además, tiene una línea adicional que crea a “fft.o” si fft.cpp o fft.h han cambiado.

Compilación

- Make, entonces, hace las siguientes verificaciones:
  - Verifica fft.o
  - Para verificar ftt.o, verifica fft.cpp y fft.h
    - Si ha habido cambios, o no hay fft.o, créa un fft.o nuevo.
      - “gcc -c” crea librerías en vez de programas.
    - Si no, lo deja así.
  - Compila jack_fft.cpp

Otras Observaciones del Ejemplo Completo

- Estamos verificando que la transformada y su inversa se hayan llevado a cabo.
  - Las funciones “CFFT::Forward” y “CFFT::Inverse” regresan 0 si no fueron exitosas.
- No estamos haciendo nada con la información.

Ejercicio #1

- Filtren todas las frecuencia menores a 500 Hz.
- ¿Cómo sabemos en cual índice de pSignal se úbican dichas frecuencias?
- Recuerden verificar su salida con baudline.

- Si hacemos este filtrado, ¿cómo se escucha? ¿Por qué?

Ejercicio #2

- Librow nada más proporciona la conversión e inversa de la transformada.
- El ventaneo lo tenemos que hacer nosotros.
- Recuerden:
  - Verificar que se está utilizando una ventana de tamaño igual a una potencia exacta de 2.
  - Utilizar la técnica “sobrelapa-y-suma” con la función Hann.

Examen Parcial

- Hacer un agente de JACK que filtren todas las frecuencias afuera de un rango que el usuario entregue como argumentos numéricos en Hz.
- Ejemplo:

  ./programa 1000 4000

  Filtra todas las frecuencias afuera del rango entre 1 kHz y 4 kHz.