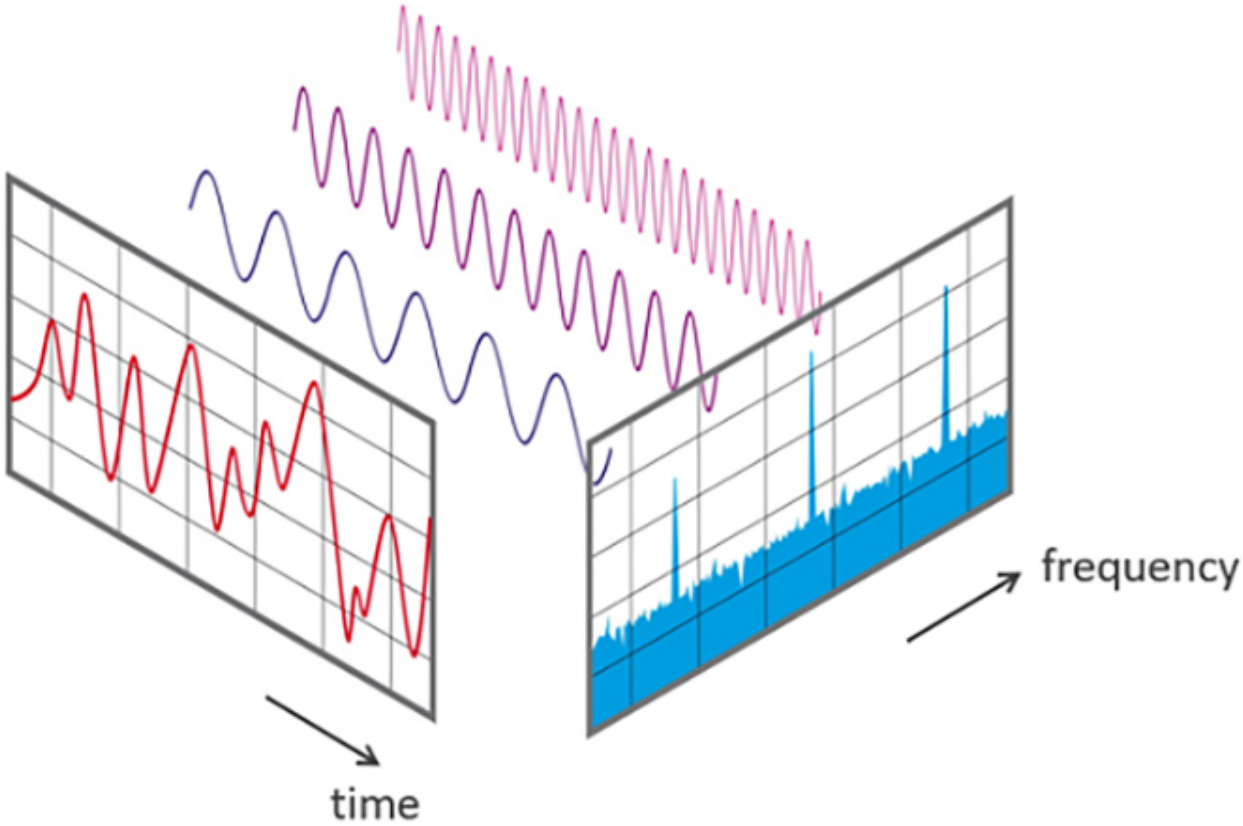
La Transformada de Fourier como herramienta en el analisis de datos

Aplicaciones de la teoria de Fourier

Fabian Trigo[1] Nicolas Gatica[2]

[1,2]Licenciatura en Física
Universidad de Valparaíso

Estadística, diciembre 2022


frequency
time

Existen varias transformadas de Fourier

Para una funcion $f(x)$:

Series de Fourier

Se toma un intervalo T donde se refiere al perdiodo de la funcion, se obtienen los coeficienes correspondientes a cada frecuencia

Transformada de Fourier

Pasa una funcion del dominio del tiempo al dominio de frecuencias, similar a las series, pero el intervalo es infinito

Transformada de Fourier Discreta

Identica a la serie de Fourier, Se ocupa de convertir una secuencia de datos y_n en una secuencia de coeficientes correspondientes a cada frecuencia presente

Series de Fourier

$$f(x) = \sum_{-\infty}^{\infty} c_k e^{i\omega_k x}$$

$$c_k = \int_0^T f(x) e^{-i\omega_k x} dx$$

todas las frecuencias son multiplos de la frecuencia fundamental $\omega = \frac{2\pi}{T}$
tambien la version utilizando funciones trigonometricas:

$$f[x] = \sum_{k=0}^{\infty} a_k \cos[k\frac{2\pi}{T}x] + b_k \sin[k\frac{2\pi}{T}x]$$

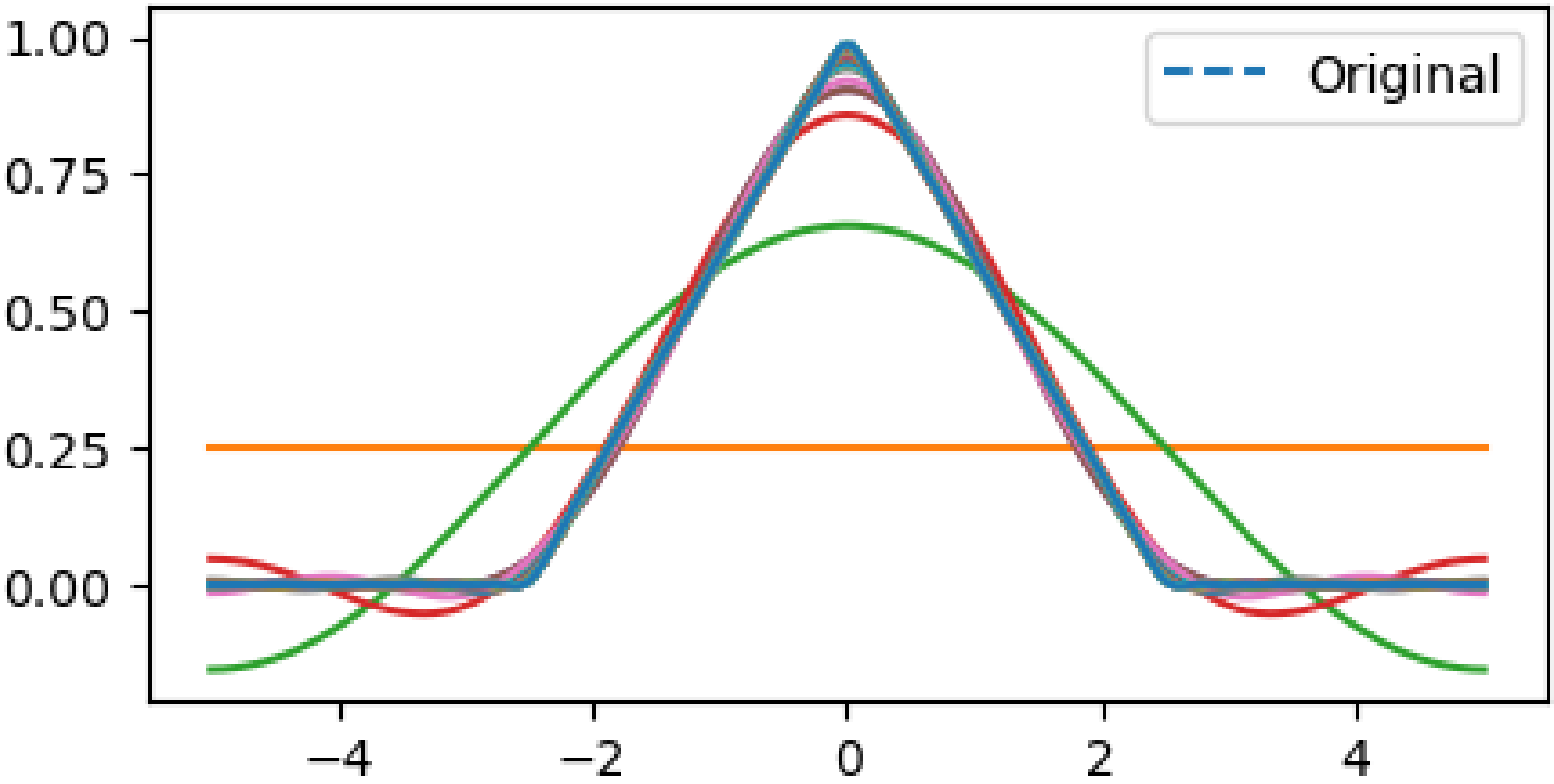


Figure: Serie de Fourier para funcion sombrero definida a partes

Serie trigonometrica

los componentes pueden ser calculados mediante el producto interno entre la funcion y la funcion trigonometrica en cuestion

$$a_k = \frac{2}{T} \int_{t_0}^{t_0+T} f[x] \cos[\omega k x) dx$$

$$b_k = \frac{2}{T} \int_{t_0}^{t_0+T} f[x] \sin[\omega k x) dx$$

de donde viene el $\frac{2}{T}$?

teniendo la sumatoria original, multiplicamos por $\cos(\omega jx)$ e integramos de $t_0 \to t_o + T$

$$\int f(x)\cos(\omega jx)dx = \int \sum_{k=0}^{\infty} a_k \cos(k\omega x)\cos(\omega jx) + b_k \sin(k\omega x)\cos(\omega jx)dx$$

donde $\int_0^T \sin(k)\cos(\omega jx) = 0$ y en el caso del coseno por coseno para $k \neq 0$:

$$\int_{t_0}^{t_0+T} \cos(k\omega x)\cos(\omega jx)dx = \begin{cases} 0 & j \neq k \\ \frac{T}{2} & j = k \end{cases}$$

ya que es solo cuando $j = k$ que no da 0, entonces:

$$\int_{t_0}^{t_0+T} f(x)\cos(\omega jx)dx = a_j \frac{T}{2} \to a_j = \frac{2}{T}\int_{t_0}^{t_0+T} f(x)\cos(\omega jx)dx$$

sin embargo cuando $k = 0$, entonces $j = 0$ para valores no 0 de la integral

$$\int_{t_0}^{t_0+T} \cos(k\omega x)\cos(\omega j x)dx = \int_{t_0}^{t_0+T} \cos(0)\cos(0)dx = \int_{t_0}^{t_0+T} dx = T$$

por tanto:

$$\int_{t_0}^{t_0+T} f(x)\cos(0)dx = a_0 T \rightarrow a_0 = \frac{1}{T}\int_{t_0}^{t_0+T} f(x)dx$$

Para los coeficientes del seno ocurre lo mismo, para $k \neq 0$

$$\int_{t_0}^{t_0+T} \sin(k\omega x)\sin(\omega j x)dx = \begin{cases} 0 & j \neq k \\ \frac{T}{2} & j = k \end{cases}$$

mientras que el caso $k = 0$ es simplemente 0, pues el $\sin 0 = 0$, entonces:

$$\int_{t_0}^{t_0+T} f(x)\sin(\omega j x)dx = b_j \frac{T}{2} \rightarrow b_j = \frac{2}{T}\int_{t_0}^{t_0+T} f(x)\sin(\omega j x)dx$$

Regla trapezoidal I

Sea la integral $I[a,b] = \int_a^b f[x]dx$, usamos N divisiones, cada trapezoide tendra un ancho

$$h = \frac{b-a}{N}$$

k es el iterador, un trapozoide tiene dos alturas, a mano derecha:

$$x = a + kh \rightarrow f[a + kh]$$

a mano izquierda del k trapezoide:

$$x = a + kh - h \rightarrow f[a + k(h-1)]$$

entonces el area de un trapezoide:

$$A_k = \frac{h}{2}[f[a + k(h-1)] + f[a + kh]]$$

Regla trapezoidal II

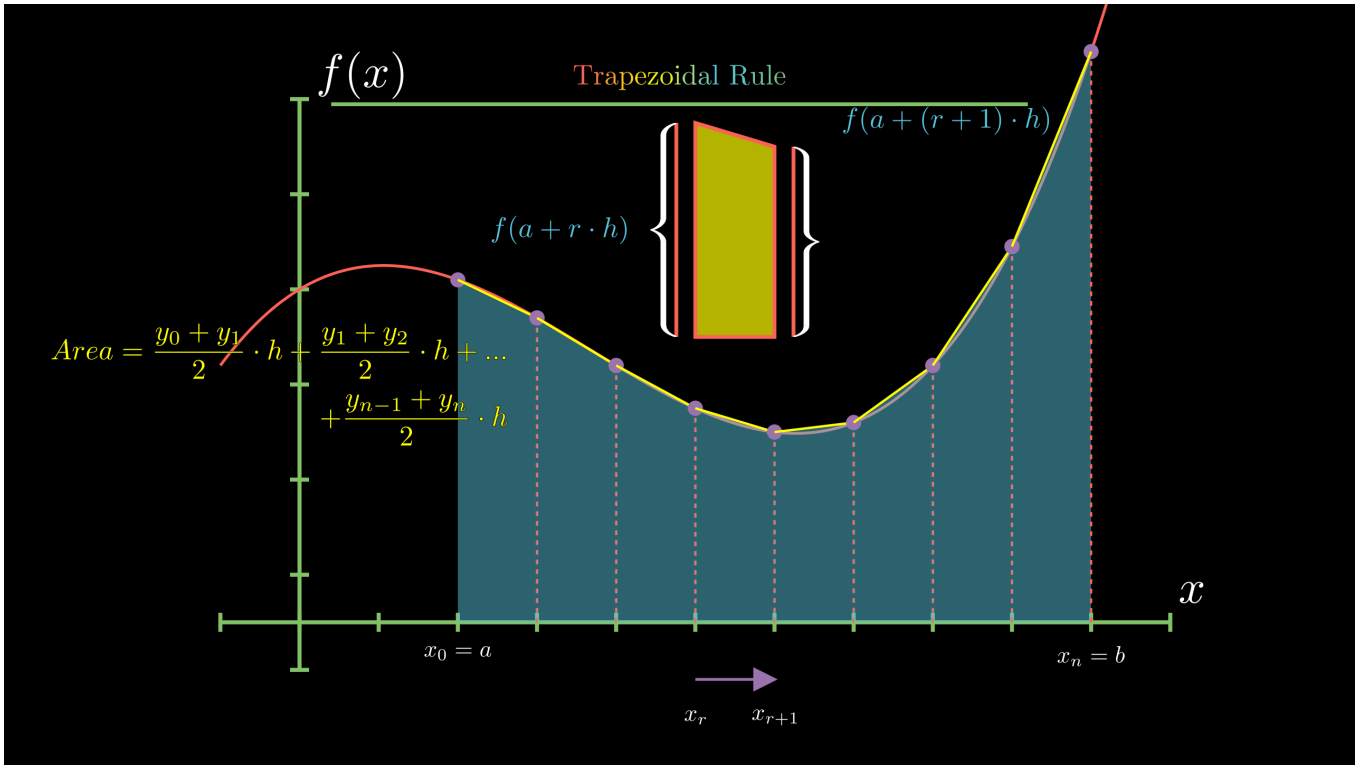


Figure: Regla Trapezoidal, https://miro.medium.com/max/5120/1*2jcjgSAw$_q$*oimpVm*$_2$*JNrg.png*

Regla trapezoidal III

donde reordenando veremos terminos repetidos excepto por los extremos

$$I[a,b] \approx \sum_k A_k = h(\frac{f[a]}{2} + \frac{f[b]}{2} + \sum_{k=1}^{N-1} f[a+kh])$$

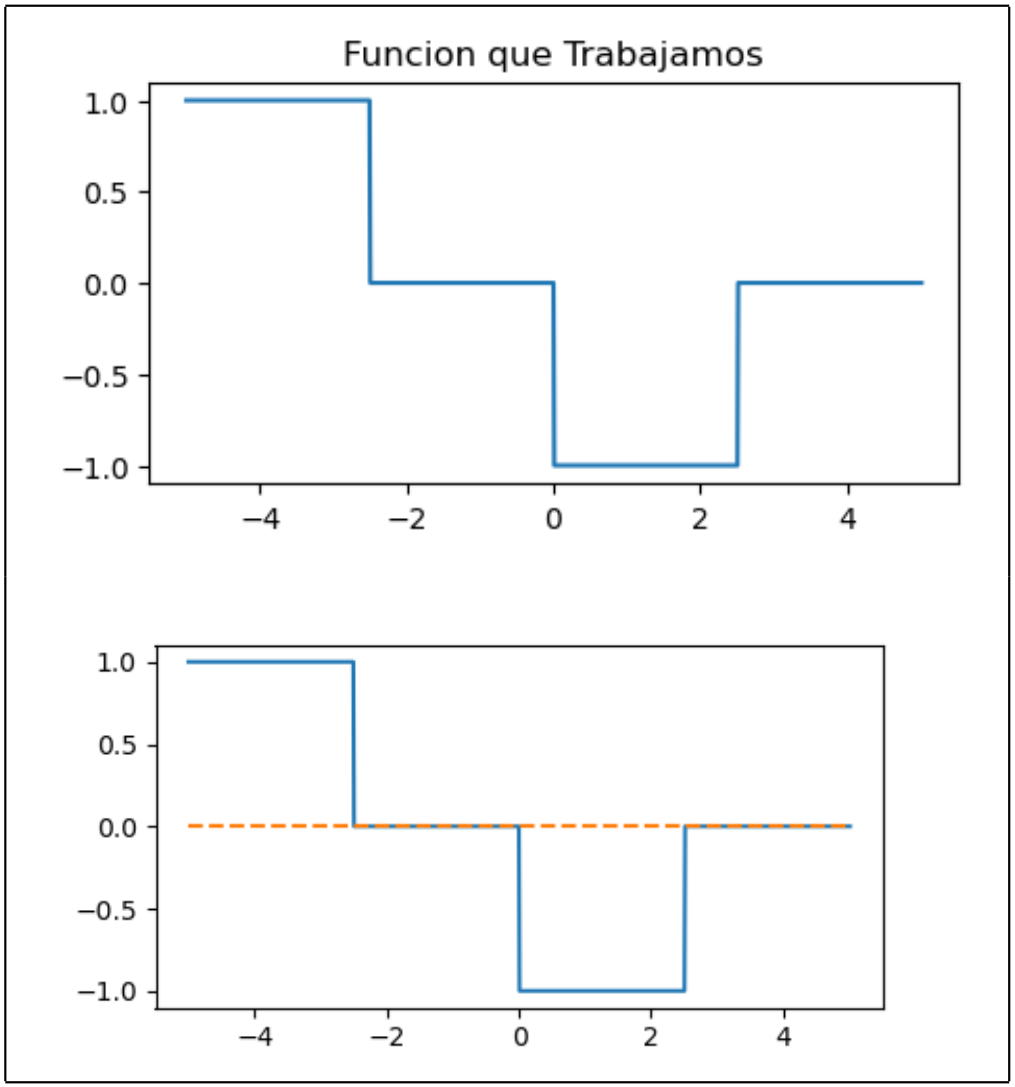
y en el caso de funciones periodicas donde $f[a] = f[b]$

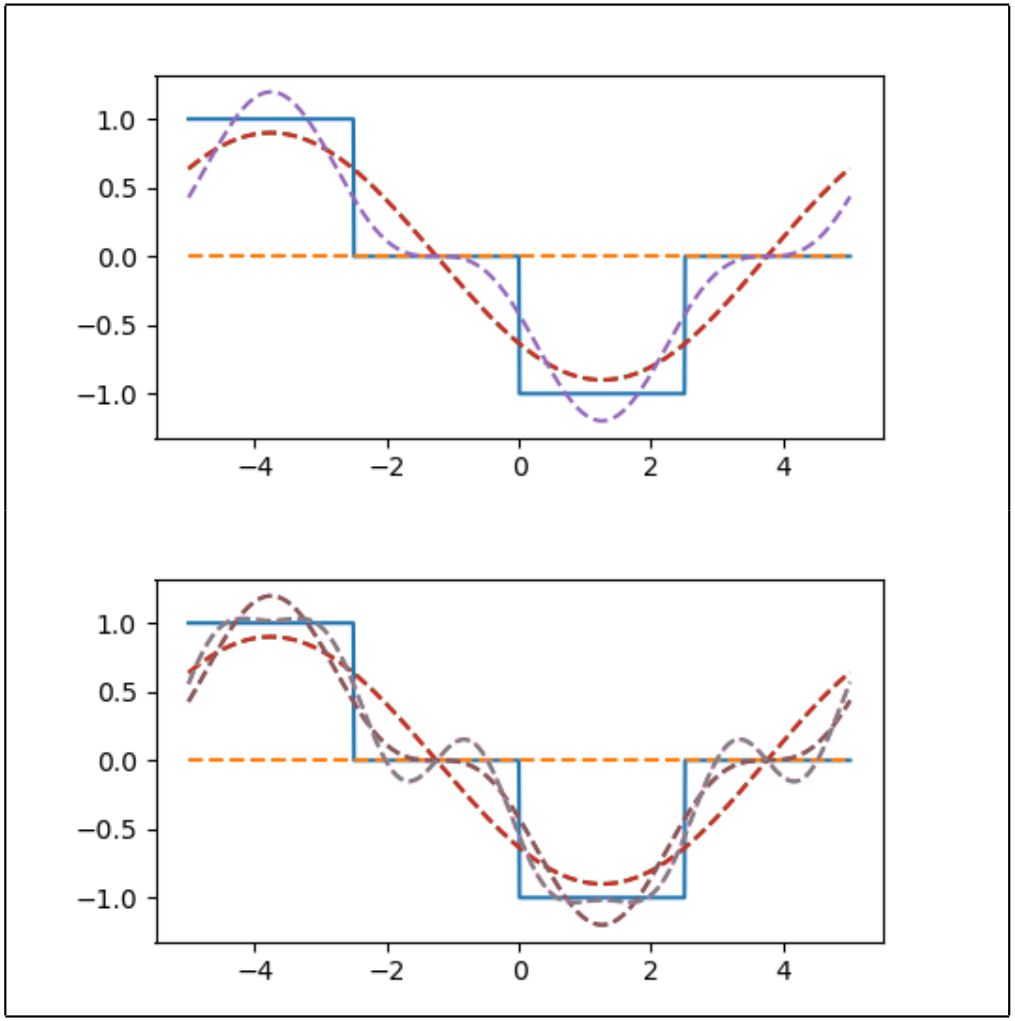
$$I[a,b] \approx h(\sum_{k=0}^{N-1} f[a+kh])$$

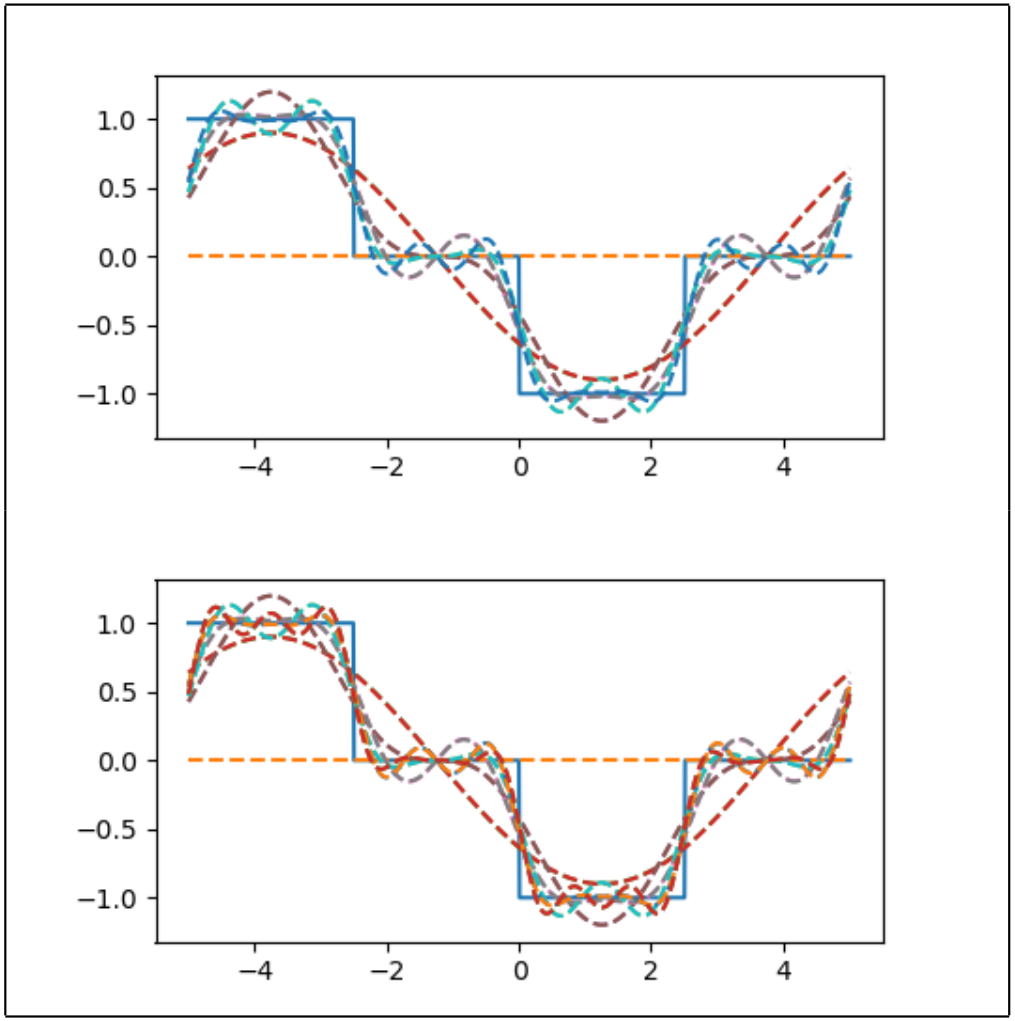
Series Trigonometricas en Python

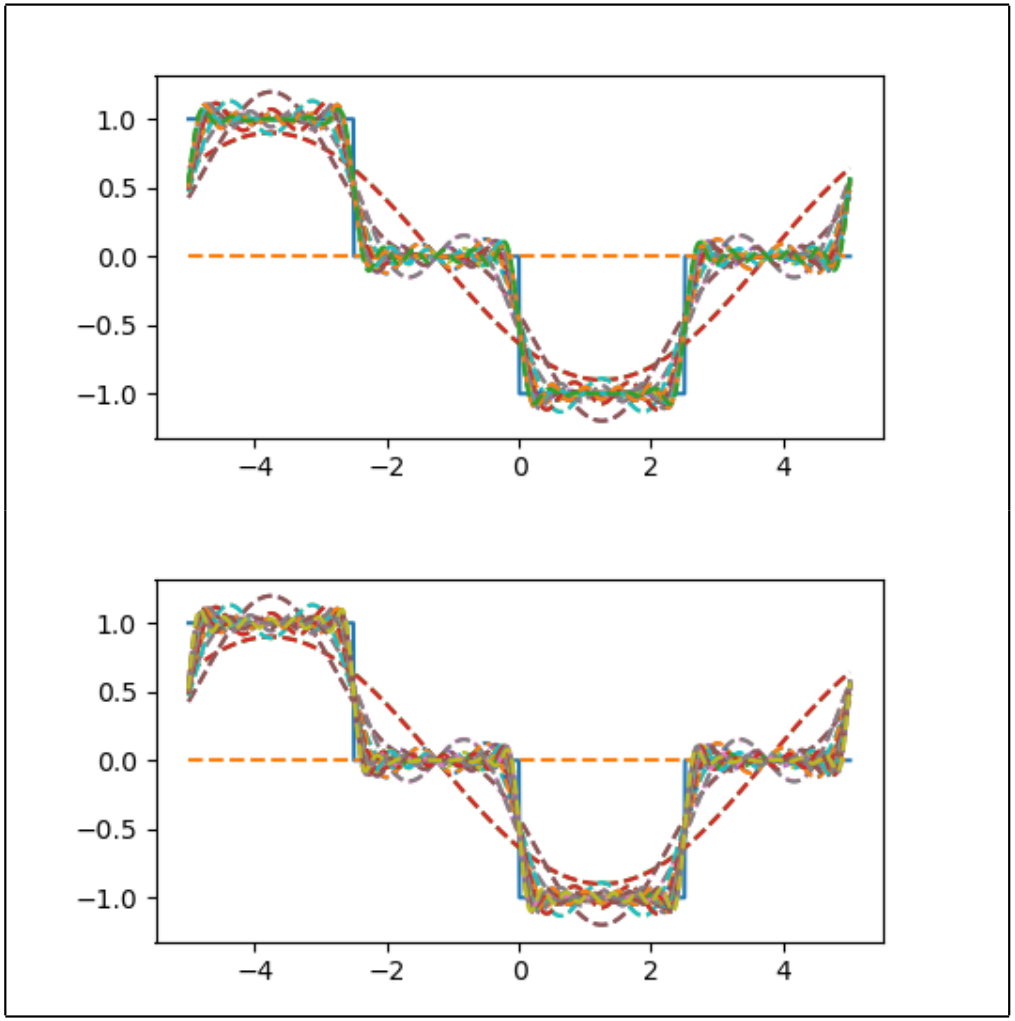
Discretizando las integrales mediante la regla trapezoidal para realizarla computacionalmente:

```
M = 30
a = np.zeros(M)
b = np.zeros(M)

for k in range(0,M):
    a[k] = (2/T)*np.sum(yn*np.cos(omega* k* x) * dx )
    b[k] = (2/T)*np.sum(yn*np.sin(omega* k* x) * dx )

a[0] = a[0] / 2 # el primer componene a_0 lleva
```


Funcion que Trabajamos
1.0
0.5
0.0
−0.5
−1.0
−4
−2
0
2
4
1.0
0.5
0.0
−0.5
−1.0
−4
−2
0
2
4


1.0
0.5
0.0
−0.5
−1.0
−4
−2
0
2
4
1.0
0.5
0.0
−0.5
−1.0
−4
−2
0
2
4


1.0
0.5
0.0
−0.5
−1.0
−4
−2
0
2
4
1.0
0.5
0.0
−0.5
−1.0
−4
−2
0
2
4


1.0
0.5
0.0
−0.5
−1.0
−4
−2
0
2
4
1.0
0.5
0.0
−0.5
−1.0
−4
−2
0
2
4

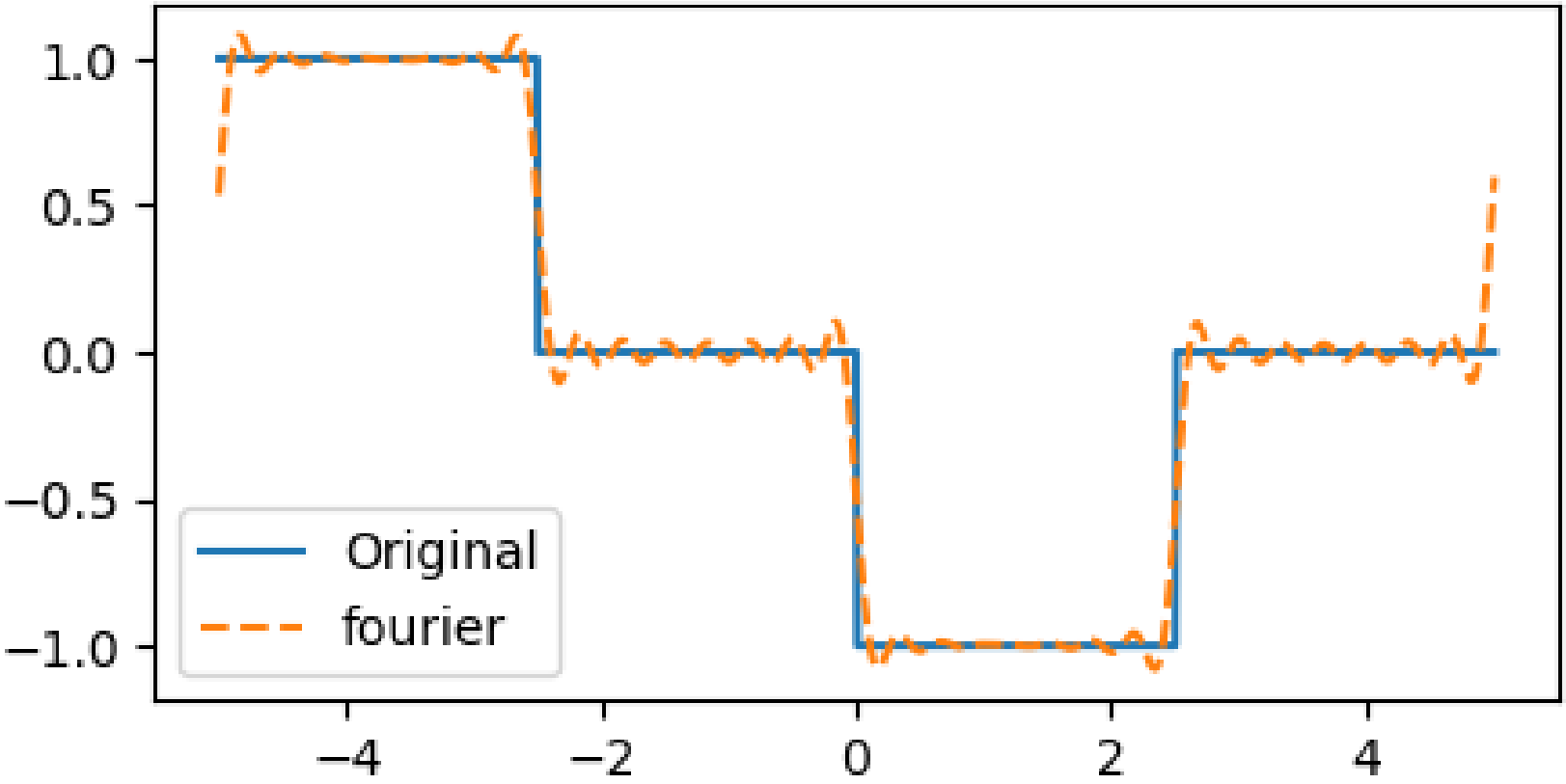


Figure: Visualizacion del total de las sumas que aproximan a la funcion

Este caso es util para partir pues comenzamos a ver el fenomeno de Gibbs

Transformada de Fourier

Lleva el problema a un espacio de frecuencias ξ

$$\mathcal{F}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\xi x} dx$$

$$f(x) = \int_{-\infty}^{\infty} \mathcal{F}(\xi) e^{i2\pi\xi x} d\xi$$

la condicion es: $\int_{-\infty}^{\infty} |f(x)| dx < \infty$, como nota en fisica se acostumbra la siguiente definicion con frecuencia angular ω y es la que usaremos de aqui en adelante:

$$\mathcal{F}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx$$

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \mathcal{F}(\xi) e^{i\omega x} d\xi$$

$$\nabla^2\psi[r,t] - \frac{1}{c^2}\frac{d^2}{dt^2}\psi[r,t] = 0$$

y representando ψ como la inversa de su transformada

$$\psi[r,t] = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\phi[r,\omega]e^{-i\omega t}d\omega$$

reemplazando queda la ecuación en el espacio de frecuencias:

$$\nabla^2\phi[r,\omega] - \frac{\omega^2}{c^2}\phi[r,\omega] = 0$$

entonces el problema se vuelve mucho más simple comparado al anterior

Serie de Fourier Compleja

La version compleja nos entrega una version más compacta pues aprovecha la identidad de euler para la exponencial compleja;

$$y(x) = \sum_{k=-\infty}^{\infty} \gamma_k e^{i\frac{2\pi}{T}kx}$$

$$\gamma_k = \frac{1}{T}\int_{t_0}^{t_0+T} f[x)\exp[-i\omega kx)dx$$

mediante la regla trapezoidal se consigue la serie de Fourier compleja en puntos en lugar de funciones continuas, lo que se llama transformada discreta de Fourier, $h = \frac{T}{N}$; $x = nT/N$

$$\gamma_k = \frac{1}{T} h \sum_{n=0}^{N-1} y_n \exp[-i\omega kx)$$

donde ocurre
$-i\omega kx = -i\frac{2\pi}{T} k \frac{nT}{N} = -i2\pi kn/N$
$\frac{h}{T} = \frac{1}{N}$

$$\gamma_k = \frac{1}{N} \sum_{n=0}^{N-1} y_n \exp[-i2\pi kn/N)$$

sin embargo, la transformada discreta por convencion es sin el factor 1/N, ese factor va ligado a la transformada inversa (la matematica funciona perfecta cualquiera sea la eleccion):

$$c_k = N\gamma_k = \sum_{n=0}^{N-1} y_n \exp[-i2\pi kn/N)$$

$$y_n = \frac{1}{N} \sum_{k=0}^{N-1} c_k \exp[i2\pi kn/N)$$

Codigo a la Transformada Discreta

allowframebreaks]

```
M = N
c = np.zeros(M, complex)

for k in range(0, M): # k = 0 hasta k = M-1
    for n in range(0, N):
        c[k] += f[n] * np.exp(-1j * 2*np.pi * k * n / N)
print("c[k]:",c)
```

[allowframebreaks]

```
M = N
c = np.zeros(M, complex)

for k in range(0, M): # k = 0 hasta k = M-1
    for n in range(0, N):
        c[k] += f[n] * np.exp(-1j * 2*np.pi * k * n / N)
print("c[k]:",c)
```

Reconstruir la función con los coeficientes, veasé el video adjunto en el github

```
F = np.zeros_like(x, complex)
# IDFT
for k in range(0, M):
    for n in range(0, N): # suma interna de todos los punt
        F[n] += c[k]*np.exp(1j*2*np.pi*k*n/N) / N

fig, ax = plt.subplots(figsize=(5,2.5))
# la parte real es la función original
plt.plot(x, F.real, '--')
```

La Transformada Rapida: FFT

La fast fourier transform, es un algoritmo para la transformada discreta de fourier (DFT)
La idea radica en separar las potencias par e impar, entonces al tomar los punto, comencemos con los componentes:

$$c_k = \sum_{n=0}^{N-1} y_n \exp[-i2\pi kn/N]$$

$$c_k = \sum_{n=0}^{N/2-1} y_{2n} e^{-i2\pi k(2n)/N} + \sum_{n=0}^{N/2-1} y_{2n+1} e^{-i2\pi k(2n+1)/N}$$

$$c_k = \sum_{n=0}^{N/2-1} y_{2n} e^{-i2\pi kn/\frac{N}{2}} + e^{-i2\pi k/N} \sum_{n=0}^{N/2-1} y_{2n+1} e^{-i2\pi kn/\frac{N}{2}}$$

ahorá tenemos que calcular la mitad de terminos para la exponencial y aparte tenemos presente una simetria cuando $k \to k + \frac{N}{2}$

$$c_{k+\frac{N}{2}} = \sum_{n=0}^{N/2-1} y_{2n} e^{-i2\pi(k+\frac{N}{2})n/\frac{N}{2}} + e^{-i2\pi(k+\frac{N}{2})/N} \sum_{n=0}^{N/2-1} y_{2n+1} e^{-i2\pi(k+\frac{N}{2})n/\frac{N}{2}}$$

donde $e^{-i2\pi(k+\frac{N}{2})n/\frac{N}{2}} = e^{-i2\pi kn/\frac{N}{2}} e^{-i2\pi n}$ y ocurre que $e^{-i2\pi n} = 1$; el componente $e^{-i2\pi(k+\frac{N}{2})/N} = e^{-i\pi} e^{-i2\pi k/N} = -e^{-i2\pi k/N}$

por tanto se simplifica a que se tengan estas dos ecuaciones, notesé que los componentes internos son identicos y solo varián por un signo, he aqui la transformada rapida de Fourier, dividimos el problema en 2

$$c_{k+\frac{N}{2}} = \sum_{n=0}^{N/2-1} y_{2n} e^{-i2\pi(k)n/\frac{N}{2}} - e^{-i2\pi k/N} \sum_{n=0}^{N/2-1} y_{2n+1} e^{-i2\pi(k)n/\frac{N}{2}}$$

$$c_k = \sum_{n=0}^{N/2-1} y_{2n} e^{-i2\pi kn/\frac{N}{2}} + e^{-i2\pi k/N} \sum_{n=0}^{N/2-1} y_{2n+1} e^{-i2\pi kn/\frac{N}{2}}$$

Ahora la manera de usar el algoritmo es a cada componente $\sum_{n=0}^{N/2-1} y_{2n} e^{-i2\pi kn/\frac{N}{2}}$ aplicar nuevamente este algoritmo para dividir, se puede hacer facilmente en python mediante recursión

```
def FFT(P: np.array):
    N = len(P)
    if N == 1:
        return P
    w = np.exp(-2 * np.pi * 1j / N)
    Ppar, Pimpar = P[0::2], P[1::2] # coefciientes polinom
    c_par, c_impar = FFT(Ppar), FFT(Pimpar)
    c = [0] * N

    for j in range(N//2):
        c[j] = c_par[j] + w**j * c_impar[j]
        c[j+ N//2] = c_par[j] - w**j * c_impar[j]
    return c
```

por otro lado la transformada inversa tambien se puede implementar FFT:

```
def IFFT(P: np.array):
    N = len(P)

    if N == 1:
        return P
    w = (1/N) * np.exp(2 * np.pi * 1j / N)
    Ppar, Pimpar = P[0::2], P[1::2] # coefciientes polinom
    y_par, y_impar = IFFT(Ppar), IFFT(Pimpar)
    y = [0] * N
    for j in range(N/2):
        y[j] = y_par[j] + w**j * y_impar[j]
        y[j+ N/2] = y_par[j] - w**j * y_impar[j]
    return y
```

Para la DFT son N^2 operaciones como vimos nuestro codigo esta dentro de dos for loops.
Para la FFt tenemos, 2 operaciones por cada DFT al dividir en 2 el problema, más $(\frac{N}{2})^2$ operaciones que es aplicar el DFT en $N/2$ elementos y N operaciones al sumar la exponencial junto a los componentes par e impar:

$$2 \times (\frac{N}{2})^2 + N = \frac{N^2}{2} + N$$

luego al ir dividiendo el problema p veces:

$$\frac{N^2}{2^p} + pN = \frac{N^2}{N} + N \log_2 N$$

Transformada Discreta de Fourier DFT:

$$O(N^2)$$

Transformada Discreta de Fourier DFT usando FFT:

$$O(N \log_2 N)$$

digamos 1 nano segundo por operación, si tenemos $N = 10^9$
DFT

$$N^2 = 10^{18} ns \rightarrow 31.2 años$$

FFT

$$N \log_2 N = 30 \times 10^9 ns \rightarrow 30 segundos$$

Es posible definir la transformacion como una operacion sobre los datos y con una matrix W para obtener el vector de coeficientes c, notar que incluiremos el factor $1/\sqrt{N}$ de manera de tener matrices unitarias

$$\vec{c} = W\vec{y}$$

$$W = \frac{1}{\sqrt{N}}\omega^{ij}$$

$$W = \frac{1}{\sqrt{N}}\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix}$$

```
# matrix de fourier compleja
W = np.zeros((N, N), dtype=np.complex_)
for k in range(N):
    for n in range(N):
        W[n,k] = 1.0/np.sqrt(N)*np.exp(1.0j *2*np.pi*n*k/N)

# matrix de fourier compleja
W_inv = np.zeros((N, N), dtype=np.complex_)
for k in range(N):
    for n in range(N):
        W_inv[n,k] = 1.0/np.sqrt(N)*np.exp(-1.0j*2*np.pi*n*
```

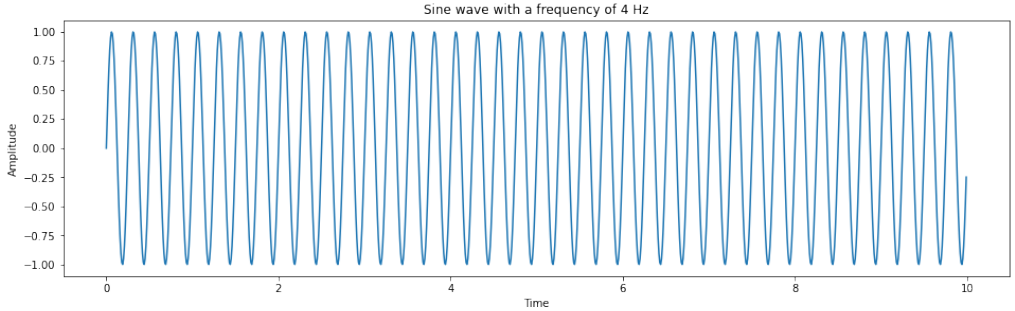
Teorema Shannon-Nyquist

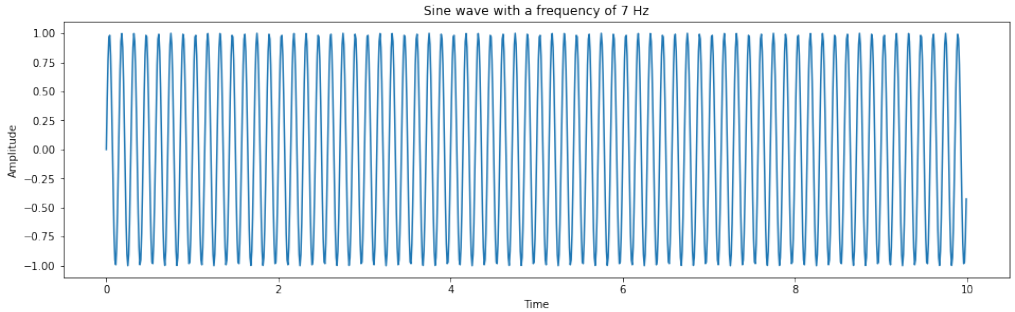
Shannon, 1948 y Nyquist, 1928, investigadores en Bell Labs que se encargaron de construir tal como la conocemos hoy en dia.
Shannon se encargaba principalmente de la criptografia en tiempo de la 2da guerra mundial, el paralelo americano de Turing.
Con una frecuencia de muestro $2\omega\, Hz$ es posible resolver hasta la frecuencia $\omega\, Hz$ en la señal sin tener error.

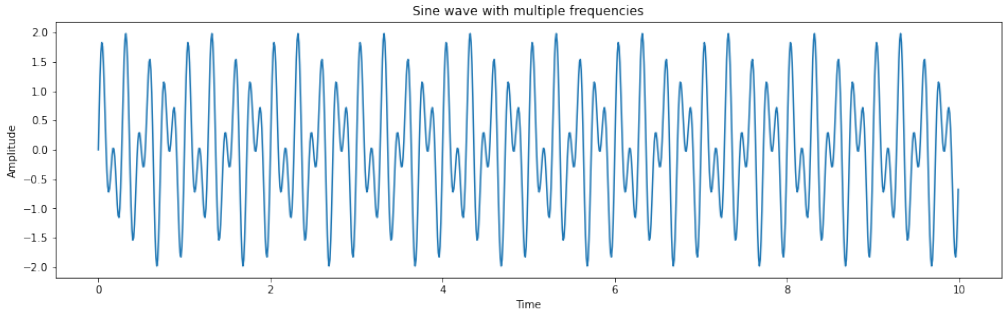
$$\Delta t = \frac{1}{2\omega\, Hz}$$

Partiendo con unos 100 puntos, entre un tiempo de 0 a 10;

```
samplingFrequency = 100 # N = 100
samplingInterval  = 1 / samplingFrequency

beginTime = 0
endTime = 10

signal1Frequency = 4 # frecuencias en Hz
signal2Frequency = 7

time = np.arange(beginTime, endTime, samplingInterval)

amplitude1 = np.sin(2*np.pi* signal1Frequency *time)
amplitude2 = np.sin(2*np.pi* signal2Frequency *time)

amplitude = amplitude1 + amplitude2
```


Sine wave with a frequency of 4 Hz
Amplitude
Time

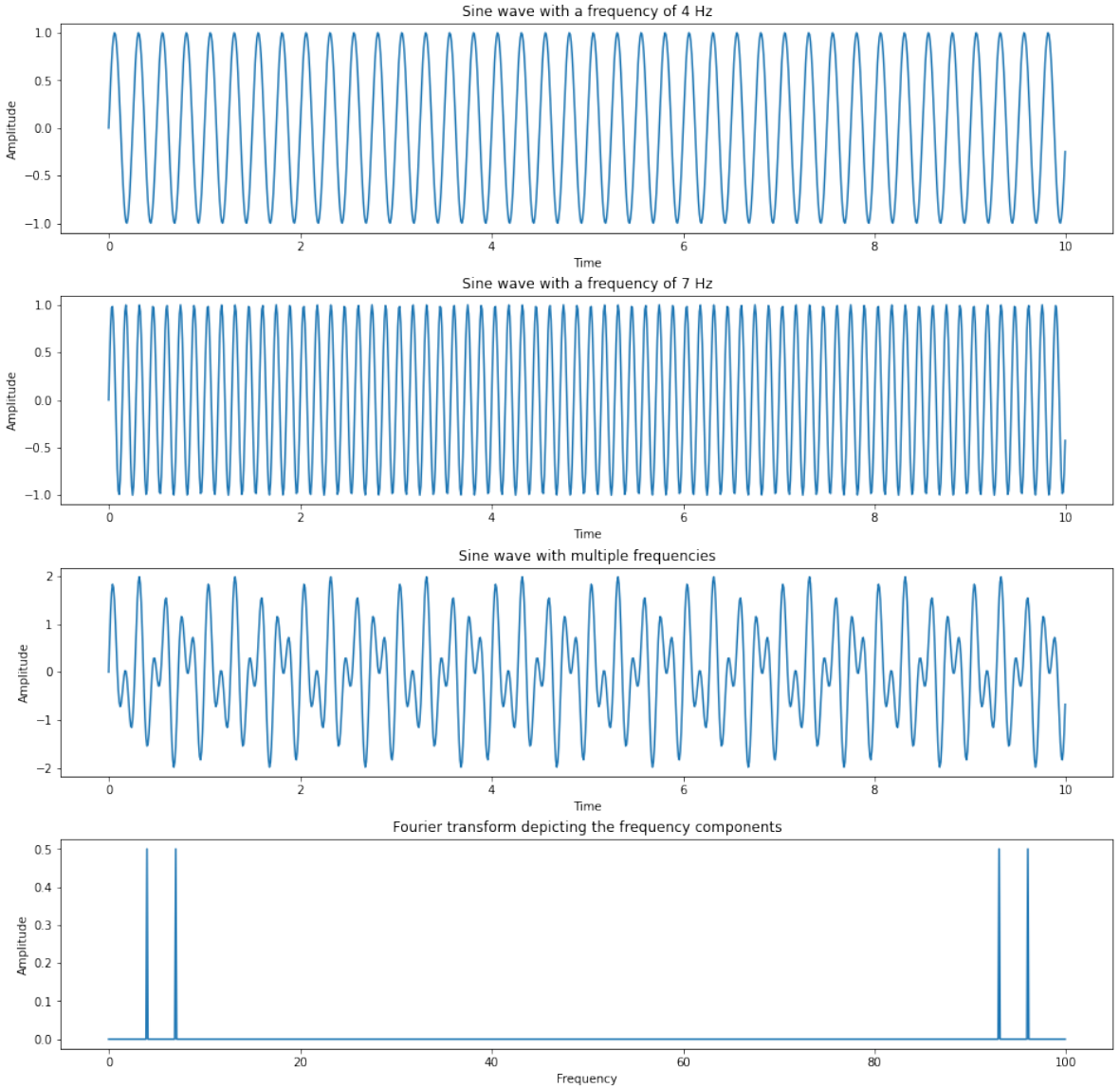

Sine wave with a frequency of 7 Hz
Amplitude
Time


Sine wave with multiple frequencies
Amplitude
Time

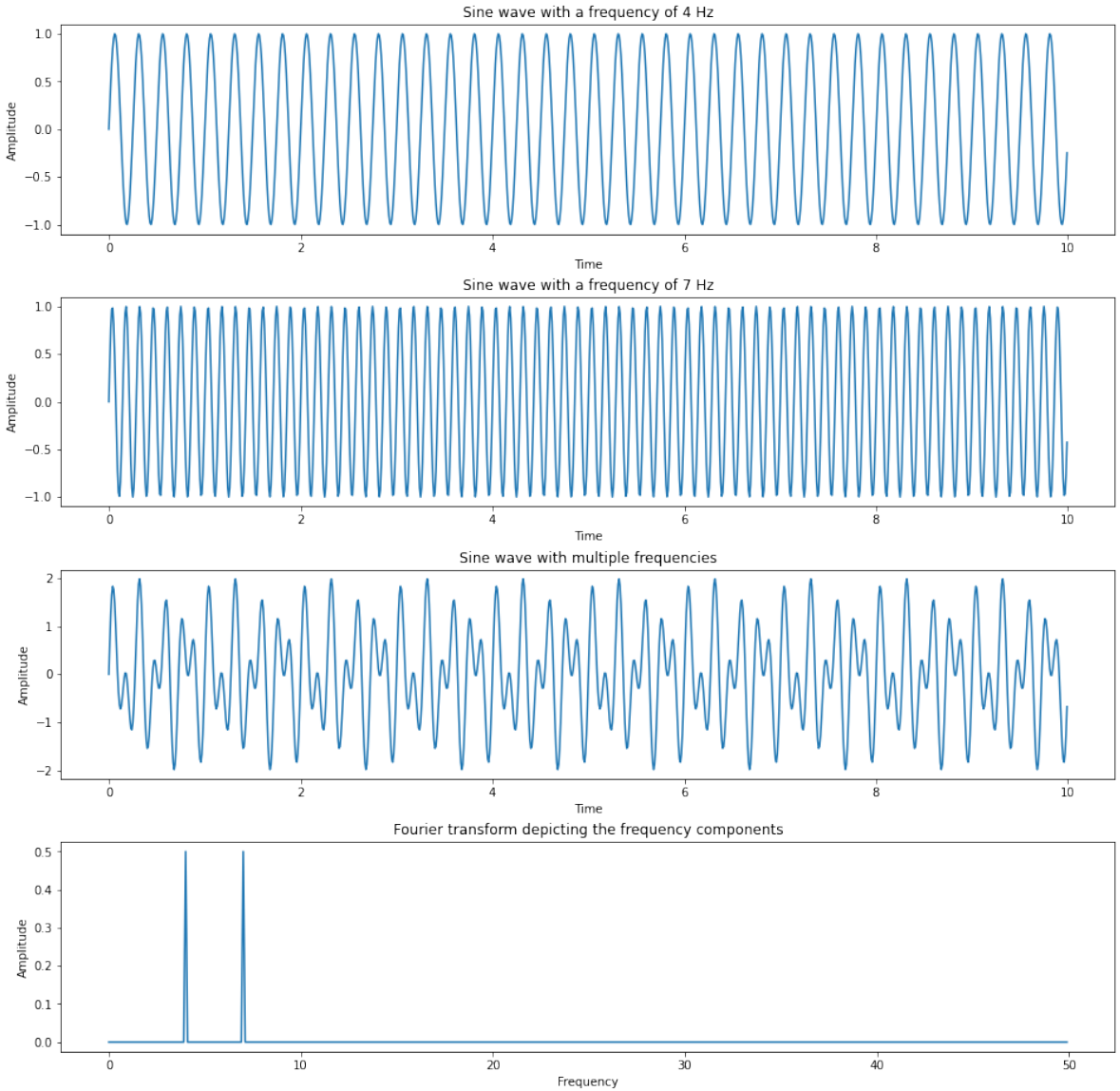
Partiendo con unos 100 puntos, entre un tiempo de 0 a 10;

```
N = len(amplitude)
fourierTransform = np.fft.fft(amplitude) / N
# Teorema Shannon Nyquist
# Sea la mayor frecuencia de la funcion omegaHz
# Podemos determinar todas las frecuencias con un sampling

# sampling es de 100Hz -> determino hasta 50Hz
fourierTransform = fourierTransform[ range(int(N/2)) ]
values  = np.arange(int(N/2))

timePeriod  = N * samplingInterval
frequencies = values / timePeriod
```

Sin el uso del teorema, se observa ruido, pues solo existe la frecuencia 4 y 7 Hz.



aplicando el teorema



Varianza

Representamos una señal como $x(i)$ y $X(i)$ como su transformada de fourier discreta (DFT); la estimacion de los coeficientes y su varianza es de suma importancia pues nos entrega la informacion sobre la señal.

$$x(i) = x_0(i) + \epsilon(i)$$

- $x_0(i)$ son muestras de la señal sin ruido
- $\epsilon(i)$ se refiere al ruido

Es una practica común dividirla en periodos de manera que se calculen los coeficientes en cada periodo, promediarlos y analizar la varianza, esto de acuerdo a Welch, 1967; para detalles sobre el analisis de varianza Balogh et al., 2002.

La transformada de Fourier sera:

$$F(x) = F(x_0) + F(\epsilon) = F_0 + E$$

donde por el teorema de Parseval:

$$\frac{1}{N}\sum_{k}^{N}|E_k|^2 = \sum_{n}^{N}|\epsilon_n|^2$$

asumiendo ruido con distribución normal con media 0:
$\sum_n |\epsilon_n|^2 = N\sigma_e^2$ por tanto el ruido al ser ruido tendra frecuenicas por todos lados, las cuales con N suficientemente grande
$\sum_k^N |E_k|^2 = N|E_0|^2$ donde E_0 es el promedio de los coeficientes

$$N\sigma_e^2 = \frac{N}{N}|E_0|^2$$

Tenemos asi la varianza

$$\mathbb{V}ar[x] = \sigma_e^2 = \frac{|E_0|^2}{N}$$

Transformada de Karhunen–Loeve

En teoría de procesos estocásticos, el teorema de Karhunen-Loeve es una representación de un proceso estocástico como una combinación lineal infinita de funciones ortogonales.
Los coeficientes de de la transformada de Karhunen-Loeve son variables aleatorias, sea $X(t)$ una variable aleatoria que depende del tiempo, como una señal con ruido gaussiano

$$X(t) = \sum_{n=0}^{N} Z_n e_n(t)$$

donde las bases $e_n(t)$ son autovectores de la matriz de covarianza.

las bases $e_n(t)$ son autovectores de la matriz de covarianza;

```
t = np.linspace(0,10, 100)
y = np.sin(t) + epsilon(t)
# el vector columna y traspuesto
Y = y[:,np.newaxis]
YT = y[np.newaxis, :]
eigenval, eigenvec = np.linalg.eig( np.cov(Y @ YT) )
```

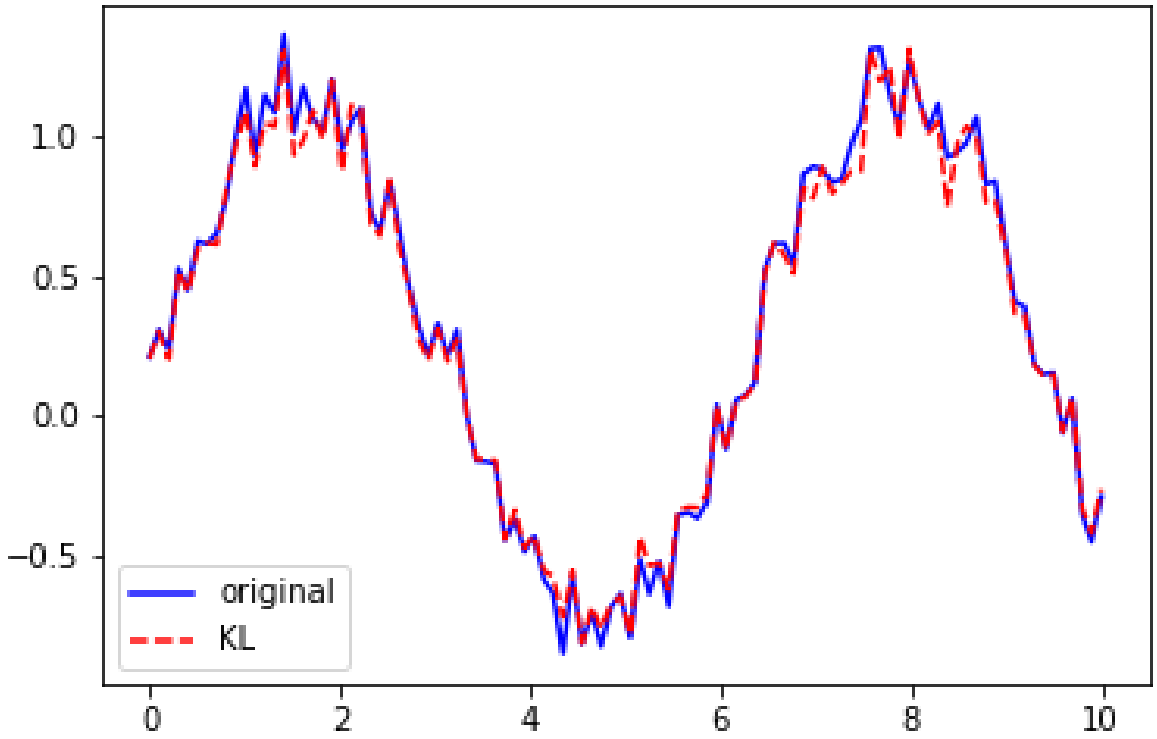
los coeficientes z se obtienen como el producto punto:

$$Z = \vec{e}(t) \cdot \vec{Y}$$

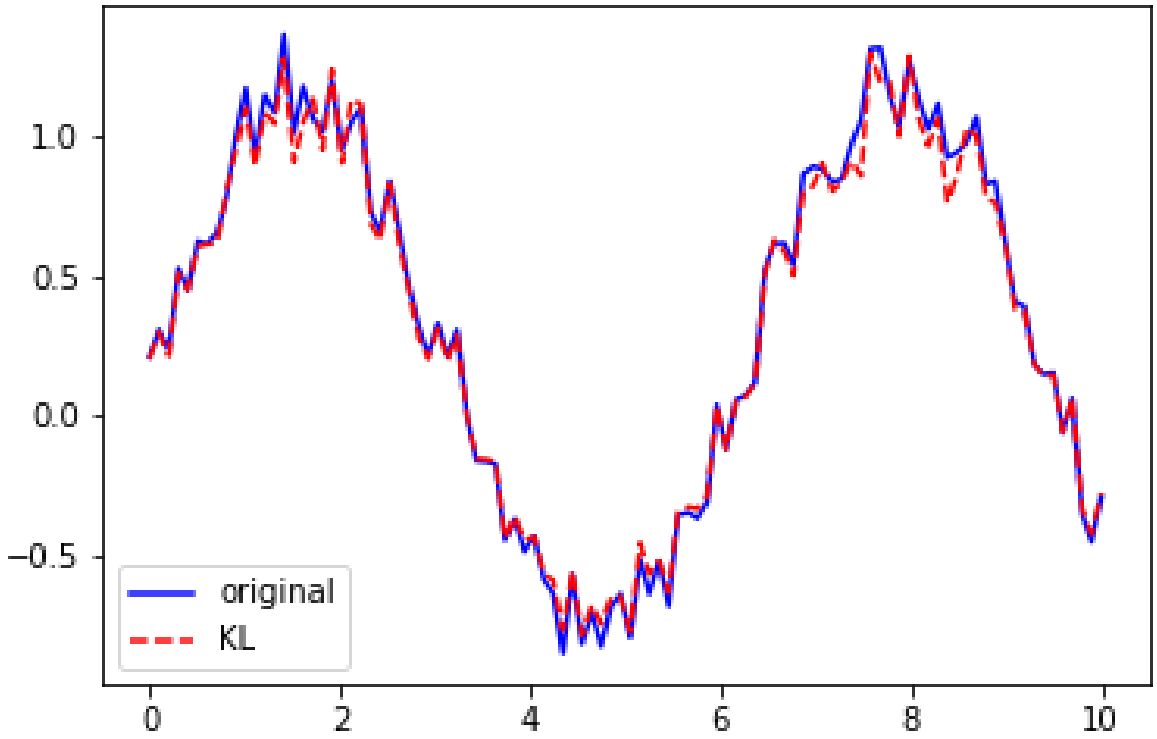
```
z = [np.dot(eigenvector[:,tt], y)
    for tt in range( eigenvec.shape[0] ) ]
```

Paralelo a Principal Component Analysis Los autovalores de la matrix guardan la variabilidad de la data en una base ortogonal, la cual captura tanto de la variabilidad de la data como es posible en las primeras funciones base $e_n(t)$

Partamos con 7 componentes y 200 puntos, he aqui la función objetivo



Hemos reducido el ruido: Cuantos componentes crees que hay aqui?

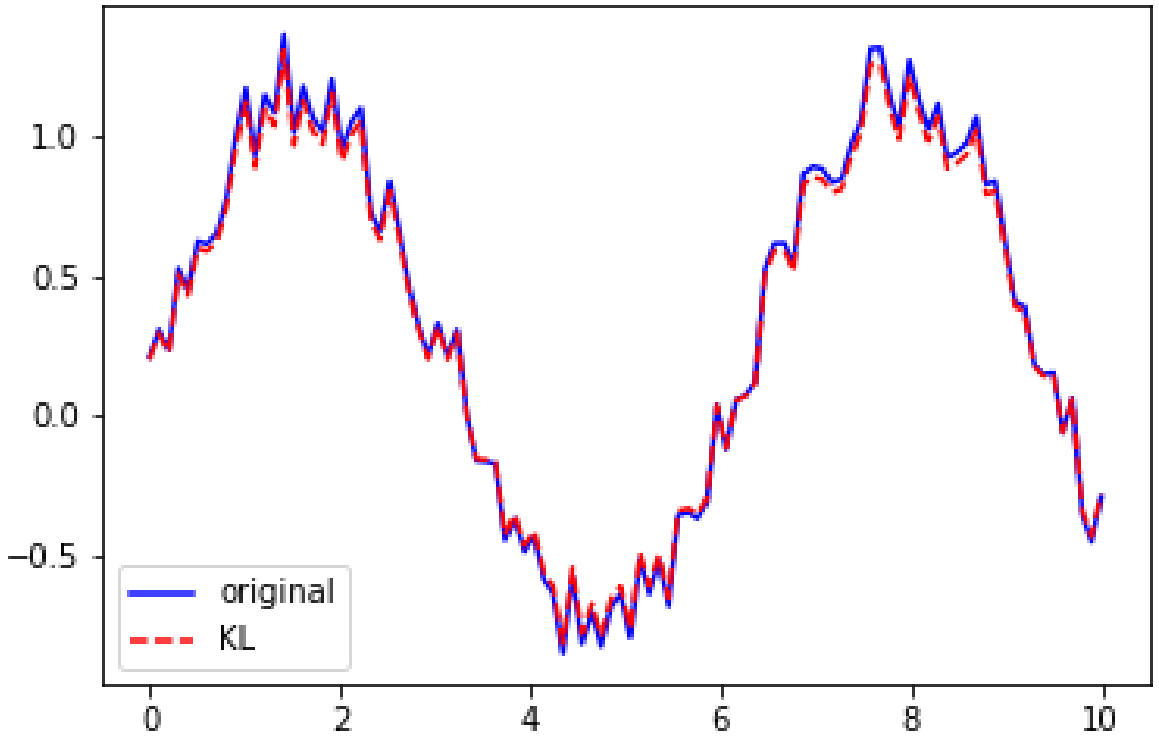


3 componentes:

$$Y(t) = \sum_{i}^{3} Z_i e_i(t)$$

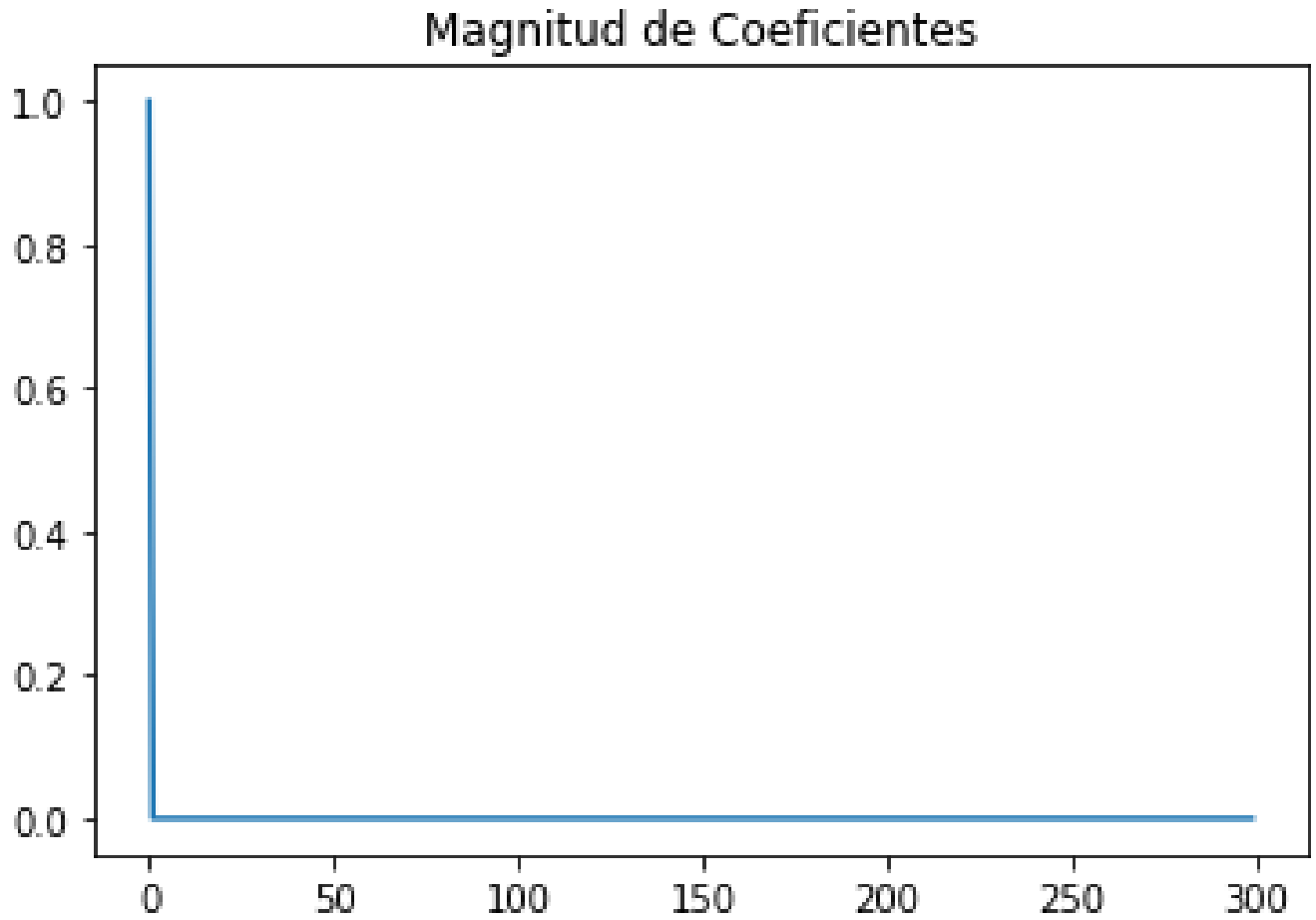
y es que la transformada contiene tanto a los datos y su variabilidad que con solo unos pocos componentes puede describir los datos

Hemos reducido el ruido: Cuantos componentes crees que hay aqui?



1 componente:

$$Y(t) = Z_0 e_0(t)$$



Matrices circulantes y ecuaciones algebráicas

Considere una matriz circulante C

- ¿Qué es una matriz circulante C?
- Propiedades de C
- Matriz de Fourier circulante

¿Qué es una matriz circulante C?

Sea $a \in \mathbf{C}$ y sean $p, q \in 0, \cdots, n-1$, entonces la entrada (p, q) de una matriz circulante está dada por:

$$(\mathrm{Cn(a)})_{p,q} = \begin{cases} a_{p-q} & p \geq p \\ a_{n+p-q} & p < q \end{cases}$$

¿Qué es una matriz circulante C?

Una matriz circulante genérica tiene la siguiente forma:

$$\begin{bmatrix} x_0 & x_{T-1} & x_{T-2} & \cdots & x_1 \\ x_1 & x_0 & x_{T-1} & \cdots & x_2 \\ x_2 & x_1 & x_0 & \cdots & x_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{T-1} & x_{T-2} & x_{T-3} & \cdots & x_0 \end{bmatrix} \tag{1}$$

Existe una correspondencia uno a uno entre todos los grados polinomiales menores que T.

Matrices circulantes en python

```
def construct_circulant(row : np.array):
    N = row.size
    C = np.empty((N, N))
    for i in range(N):
        C[i, i:] = row[:N - i]
        C[i,:i] = row[N - i:]
    return C
```

```
A = construct_circulant(np.array([i for i in range(2,10+2)]
A
array([[ 2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11.],
       [11.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.],
       [10., 11.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.],
       [ 9., 10., 11.,  2.,  3.,  4.,  5.,  6.,  7.,  8.],
       [ 8.,  9., 10., 11.,  2.,  3.,  4.,  5.,  6.,  7.],
       [ 7.,  8.,  9., 10., 11.,  2.,  3.,  4.,  5.,  6.],
       [ 6.,  7.,  8.,  9., 10., 11.,  2.,  3.,  4.,  5.],
       [ 5.,  6.,  7.,  8.,  9., 10., 11.,  2.,  3.,  4.],
       [ 4.,  5.,  6.,  7.,  8.,  9., 10., 11.,  2.,  3.],
       [ 3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11.,  2.]])
```

¿Qué es una matriz circulante C?

Sea el operador

$$K_T = [e_1, \cdots, e_{T-1}, e_0]$$

Formada por la matriz identidad I_T, esta matriz es ortonormal cuya transpuesta es la inversa

$$K_T' K_T = K_T K_T' = I_T$$

¿Qué es una matriz circulante C?

Recordando la matriz circulante:

$$\begin{bmatrix} x_0 & x_{T-1} & x_{T-2} & \cdots & x_1 \\ x_1 & x_0 & x_{T-1} & \cdots & x_2 \\ x_2 & x_1 & x_0 & \cdots & x_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{T-1} & x_{T-2} & x_{T-3} & \cdots & x_0 \end{bmatrix} \tag{2}$$

Si x(z) es un polinomio de grado menor a T, entonces existe su correspondiente matriz circulante:

$$X = x(K_T) = x_0 I_T + x_1 K_T + \cdots + x_{T-1} K_T^{T-1}$$

Propiedades de C

La representación polinómica es suficiente para establecer que la matriz circulante conmuta: Si

$$X = x(K_T) \wedge Y = y(K_T)$$

Son matrices circulantes, entonces:

$$XY = YX$$

también es una matriz circulante.

```
A = construct_circulant(np.array([i for i in range(2,10+2)]
A
array([[ 2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11.],
       [11.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.],
       [10., 11.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.],
       [ 9., 10., 11.,  2.,  3.,  4.,  5.,  6.,  7.,  8.],
       [ 8.,  9., 10., 11.,  2.,  3.,  4.,  5.,  6.,  7.],
       [ 7.,  8.,  9., 10., 11.,  2.,  3.,  4.,  5.,  6.],
       [ 6.,  7.,  8.,  9., 10., 11.,  2.,  3.,  4.,  5.],
       [ 5.,  6.,  7.,  8.,  9., 10., 11.,  2.,  3.,  4.],
       [ 4.,  5.,  6.,  7.,  8.,  9., 10., 11.,  2.,  3.],
       [ 3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11.,  2.]])
```

```
# conmutatividad
A * C
array([[ 0.,  3.,  8., 15., 24., 35., 48., 63., 80., 99.],
       [99.,  0.,  3.,  8., 15., 24., 35., 48., 63., 80.],
       [80., 99.,  0.,  3.,  8., 15., 24., 35., 48., 63.],
       [63., 80., 99.,  0.,  3.,  8., 15., 24., 35., 48.],
       [48., 63., 80., 99.,  0.,  3.,  8., 15., 24., 35.],
       [35., 48., 63., 80., 99.,  0.,  3.,  8., 15., 24.],
       [24., 35., 48., 63., 80., 99.,  0.,  3.,  8., 15.],
       [15., 24., 35., 48., 63., 80., 99.,  0.,  3.,  8.],
       [ 8., 15., 24., 35., 48., 63., 80., 99.,  0.,  3.],
       [ 3.,  8., 15., 24., 35., 48., 63., 80., 99.,  0.]])
```

```
# conmutatividad
C * A
array([[ 0.,  3.,  8., 15., 24., 35., 48., 63., 80., 99.],
       [99.,  0.,  3.,  8., 15., 24., 35., 48., 63., 80.],
       [80., 99.,  0.,  3.,  8., 15., 24., 35., 48., 63.],
       [63., 80., 99.,  0.,  3.,  8., 15., 24., 35., 48.],
       [48., 63., 80., 99.,  0.,  3.,  8., 15., 24., 35.],
       [35., 48., 63., 80., 99.,  0.,  3.,  8., 15., 24.],
       [24., 35., 48., 63., 80., 99.,  0.,  3.,  8., 15.],
       [15., 24., 35., 48., 63., 80., 99.,  0.,  3.,  8.],
       [ 8., 15., 24., 35., 48., 63., 80., 99.,  0.,  3.],
       [ 3.,  8., 15., 24., 35., 48., 63., 80., 99.,  0.]])
```

```
# conmutatividad
(A * C == C * A)
array([[ True,  True,  True,  True,  True,  True,  True,  T
         True],
       [ True,  True,  True,  True,  True,  True,  True,  T
         True],
       [ True,  True,  True,  True,  True,  True,  True,  T
         True],
       [ True,  True,  True,  True,  True,  True,  True,  T
         True],
       [ True,  True,  True,  True,  True,  True,  True,  T
         True],
       [ True,  True,  True,  True,  True,  True,  True,  T
         True],
       [ True,  True,  True,  True,  True,  True,  True,  T
         True],
       [ True,  True,  True,  True,  True,  True,  True,  T
         True],
       [ True,  True,  True,  True,  True,  True,  True,  T
         True],
```

```
# conmutatividad comprobada
np.all((A * C == C * A))
True
```

Matriz DFT en C

El operador de la matriz K_T tiene una factorización espectral particularmente util en el analisis de las propiedades de la DFT. Para la demostración de esto haremos uso de la matríz de Fourier definida anteriormente:

$$W = \frac{1}{\sqrt{T}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(T-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{T-1} & \omega^{2(T-1)} & \cdots & \omega^{(T-1)(T-1)} \end{bmatrix}$$

Matriz DFT en C

Esta es una matriz simétrica

$$U = T^{-1/2}[W^{jt}; t, j = 0, \cdots, T-1]$$

donde

$$W = e^{-i2\pi/T}$$

Matriz DFT en C

Tomando en cuenta la $T - periocidad$ de W^q, la matriz queda expresada como:

$$U = \frac{1}{\sqrt{T}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W & W^2 & \cdots & W^{T-1} \\ 1 & W^2 & W^4 & \cdots & W^{T-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W^{T-1} & W^{T-2} & \cdots & W \end{bmatrix}$$

Matriz DFT en C

La segunda fila y segunda columna de esta matriz contienen la T raices de la unidad.
La matriz conjugada se define como:

$$U = T^{-1/2}[W^{-jt}; t, j = 0, \cdots, T-1]$$

Matriz DFT en C

Usando $W^{-q} = W^{T-q}$, la matriz queda descrita como:

$$\overline{U} = \frac{1}{\sqrt{T}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W^{T-1} & W^{T-2} & \cdots & W \\ 1 & W^{T-2} & W^{T-4} & \cdots & W^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W & W^2 & \cdots & W^{T-1} \end{bmatrix}$$

Matriz DFT en C

La matriz U es unitaria, por lo que cumple la condición

$$\overline{U}U = U\overline{U} = I$$

El operador K, puede ser factorizado como:

$$K = \overline{U}DU = U\overline{DU}$$

Donde

$$D = diag(1, W, W^2, \cdots, W^{T-1})$$

Matriz DFT en C

Finalmente, juntando todo. Dada la matriz $X = x(k)$ la DFT y su inversa puede ser representada de la siguiente manera factorizandola espectralmente:

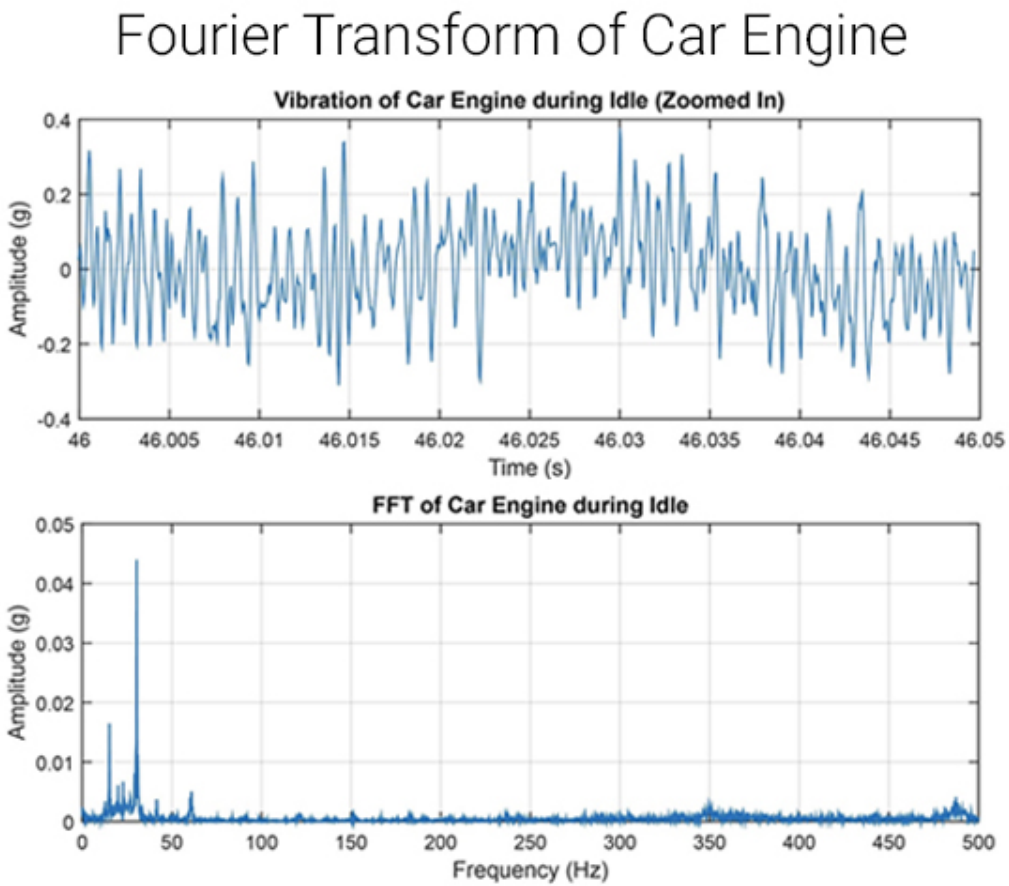
$$X = \overline{U}x(D)U \wedge x(D) = UX\overline{U}$$

Matriz DFT en C

U

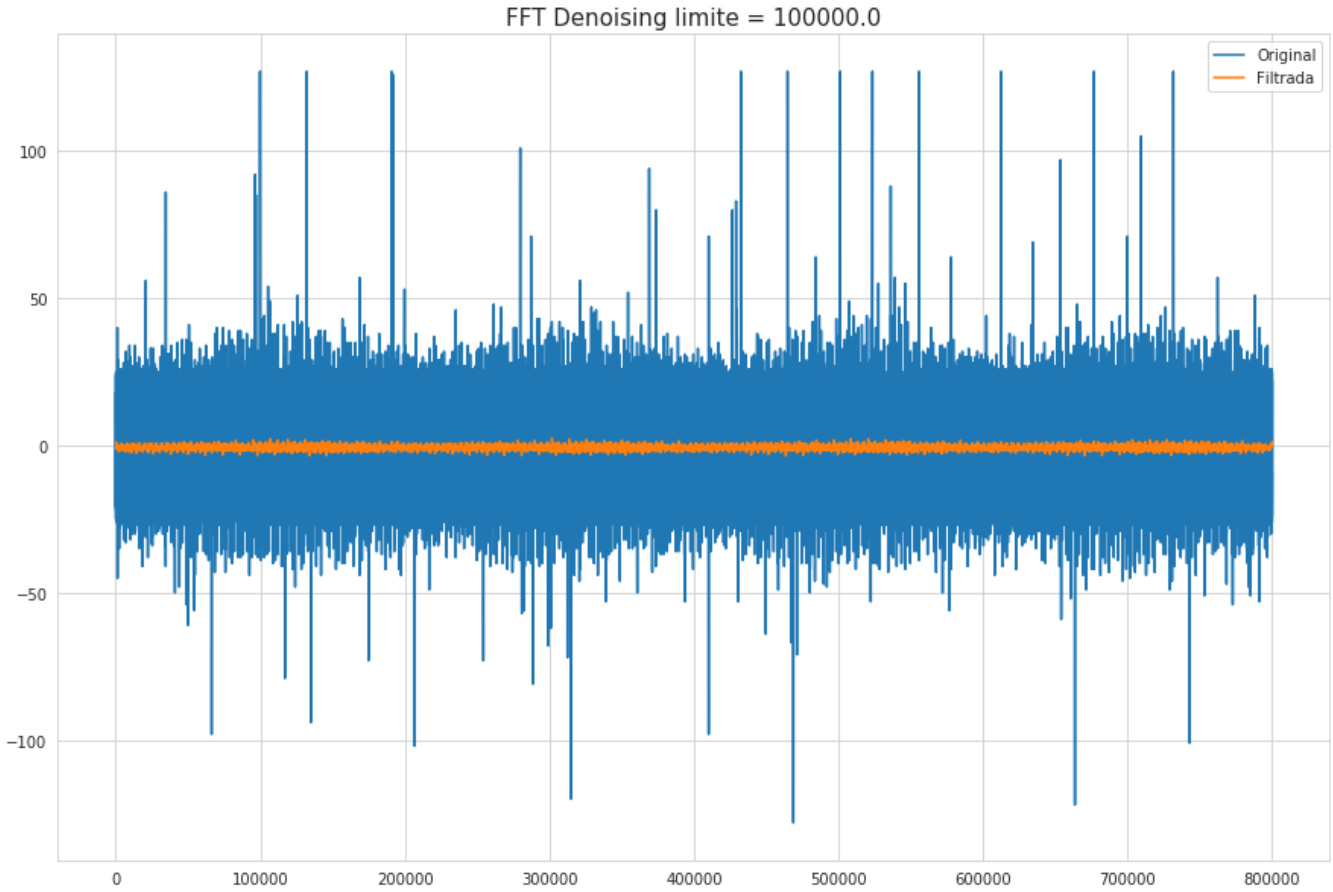
tiles en simulaciones, compresión de imagenes, etc.

Aplicaciones I

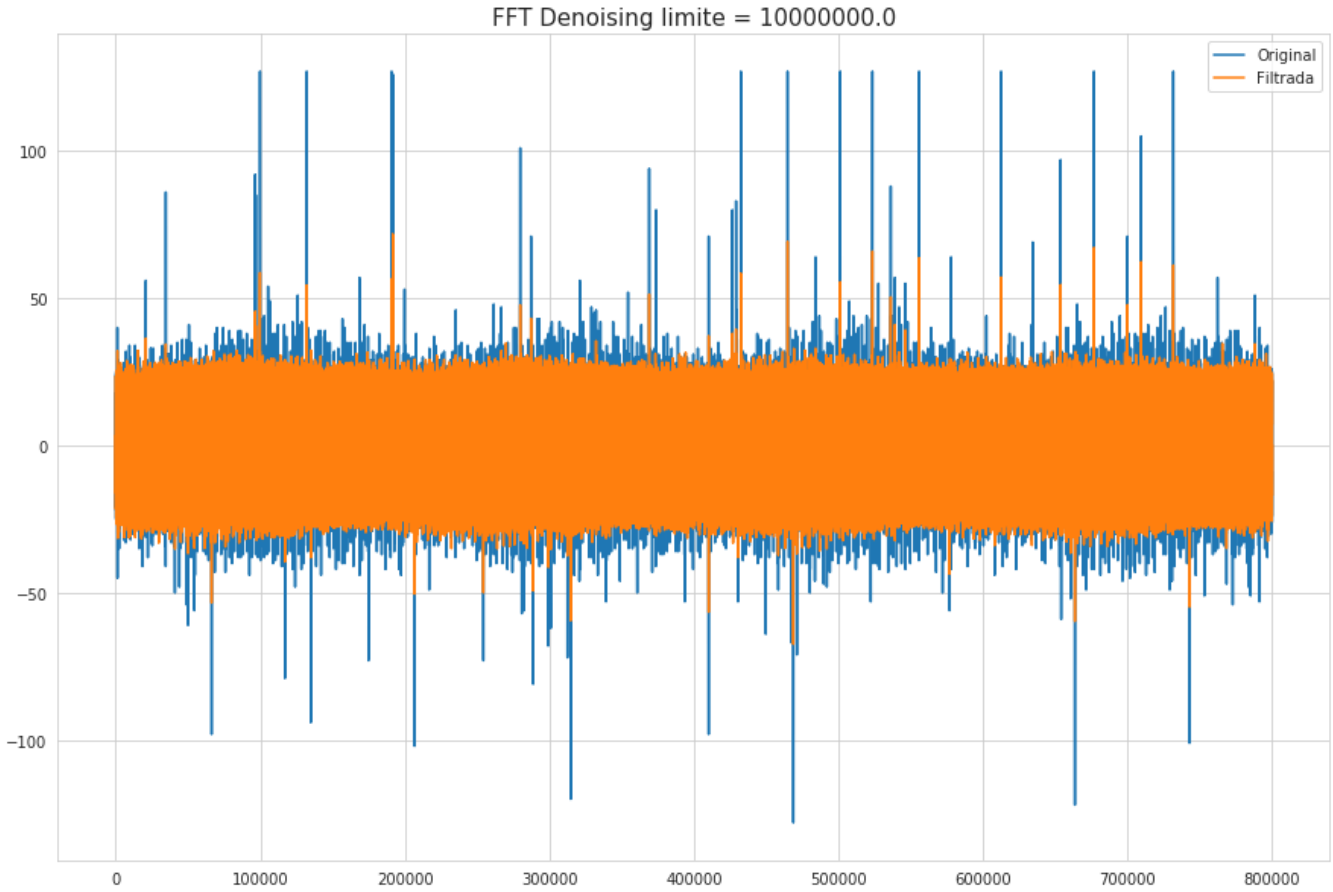


Aplicaciones II

Aplicaciones III



Aplicaciones IV



Bibliografia I

La bibliografia principal nace de: Newman, 2012; Wall and Jenkins, 2012; Pollock, 2008


Nyquist, H. (1928). Certain topics in telegraph transmission theory. *Proceedings of the IEEE, 90(2)*, 280–305. https://doi.org/https://doi.org/10.1109/5.989875


Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal, 27(4)*, 623–656. https://doi.org/https://doi.org/10.1002/j.1538-7305.1948.tb00917.x


Welch, P. (1967). The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *IEEE Transactions on Audio and Electroacoustics, 15*(2), 70–73. https://doi.org/10.1109/TAU.1967.1161901

Bibliografia II


Balogh, L., Kollár, I., & Gueret, G. (2002). Variance of fourier coefficients calculated from overlapped signal segments for system identification. *Conference Record - IEEE Instrumentation and Measurement Technology Conference*, *2*, 1065–1070 vol.2. https://doi.org/10.1109/IMTC.2002.1007103


Pollock, D. (2008). Statistical fourier analysis: Clarifications and interpretations. *Journal of Time Series Econometrics*. https://doi.org/10.2202/1941-1928.1004


Newman, M. (2012). *Computational physics*. CreateSpace Independent Publishing Platform.


Wall, J. V., & Jenkins, C. R. (2012). *Practical statistics for astronomers*. Cambridge University Press. https://doi.org/https://doi.org/10.1017/CBO9781139031998