TP de Simulación 2: estimación espectral

Integrantes

- de Otazua, Agustín
- González, José
- Urquiza, Elías

1. Técnicas no paramétricas

```
# Paquetes

import math as m
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
from scipy import signal
```

+ Code + Texte

1.1 - Estimador sesgado e insesgado de la autocorrelación

Generamos una secuencia de N = 10000 realizaciones de ruido blanco gaussiano de media nula y variancia unitaria. Con dicha secuencia, estimamos las secuencias de autocovarianza utilizando los estimadores sesgado e insesgado de la correlación:

$$Estimador\ sesgado: \frac{1}{N}\sum_{n=k}^{N-1} Y(n)Y^*(n-k), \qquad 0 \leq k \leq N-1$$

$$Estimador\ insesgado: \frac{1}{N-k}\sum_{n=k}^{N-1} Y(n)Y^*(n-k), \qquad 0 \leq k \leq N-1$$

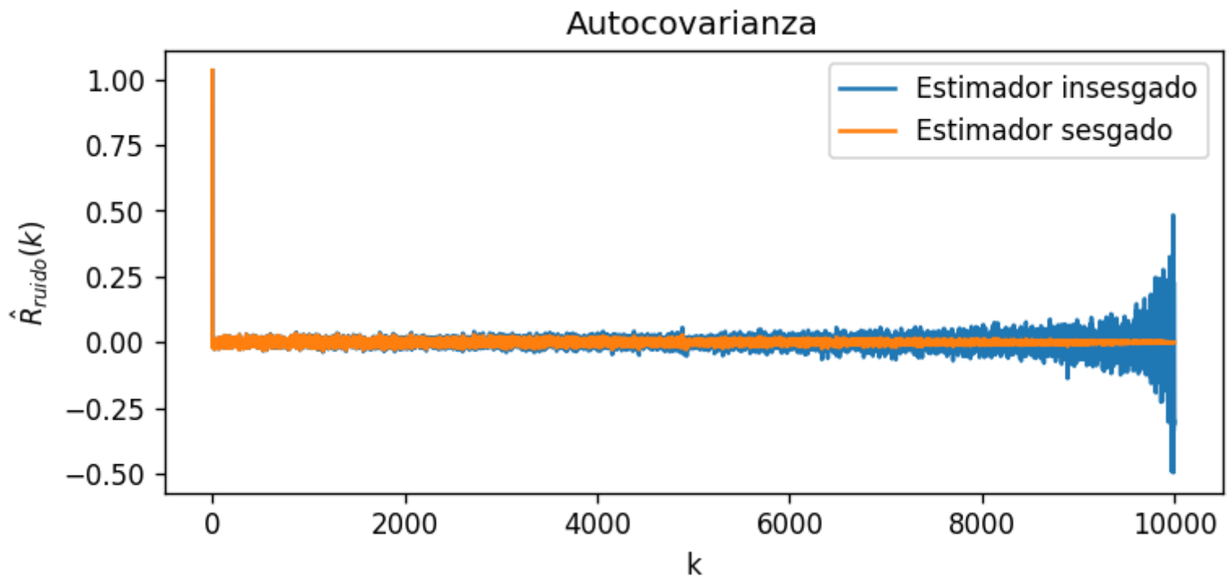
Tener en cuenta que los rangos se definen así porque consideramos $0 \leq n \leq N-1$ en vez de $1 \leq n \leq N$.

```
#Estimador de autocorrelacion insesgado
def autocorr_inses(Y, N, k):
  suma = 0
  for n in range(k, N):
    suma += Y[n] * Y[n-k]
  return 1 / (N-k) * suma

#Estimador de autocorrelacion sesgado
def autocorr_ses(Y, N, k):
  suma = 0
  for n in range(k, N):
    suma += Y[n] * Y[n-k]
  return 1 / N * suma
```

```
#Simulacion

N = 10000   #numero de realizaciones
noise = [np.random.randn() for i in range(N)]   #secuencia de ruido
sec_autocorr_inses = [autocorr_inses(noise, N, k) for k in range(N)]    #secuencia de autocorrelacion insesgada
sec_autocorr_ses = [autocorr_ses(noise, N, k) for k in np.arange(N)]    #secuencia de autocorrelacion sesgada
```

```
#Grafico

fig1=plt.figure(figsize=(7,3), dpi=120,facecolor='w',edgecolor='k')
plt.plot(sec_autocorr_inses)
plt.plot(sec_autocorr_ses)
plt.title('Autocovarianza')
plt.legend(['Estimador insesgado', 'Estimador sesgado'])
plt.xlabel('k')
plt.ylabel('$\hat{R}_{ruido} (k)$')
```

```
Text(0, 0.5, '$\\hat{R}_{ruido} (k)$')
```



Se puede ver que ambos estimadores de la autocovarianza valen 1 para $k = 0$ y que en los siguientes k decrece bruscamente, lo cual se aproxima a la autocovarianza ideal que vale 1 para $k = 0$ y 0 para $k \neq 0$.

A pesar de esto, el estimador insesgado presenta fuertes oscilaciones para valores grandes de k. Esto se debe a que el divisor de $\hat{R}_{ruido}(k)$ disminuye a medida que aumenta k; la cantidad de sumas, también. Por otro lado, tampoco tenemos garantía de que la matriz de correlación sea semidefinida positiva. Finalmente, podemos decir que para k grande se estiman valores pequeños con pocas sumas, y esto conduce a valores más erráticos y a una matriz potencialmente no semidefinida positiva.

En cambio, para el estimador sesgado las oscilaciones se reducen ya que la constante que acompaña a $\hat{R}_{ruido}(k)$ (o sea, el divisor) no depende de k y también porque este estimador garantiza que la matriz de autocovarianza sea semidefinida positiva.

1.2 - (In)consistencia del periodograma

Dado el siguiente sistema LTI con respuesta:

$$H(z) = \frac{1 - 1.3817z^{-1} + 1.5632z^{-2} - 0.8843z^{-3} + 0.4096z^{-4}}{1 + 0.3544z^{-1} + 0.3508z^{-2} - 0.1736z^{-3} + 0.2401z^{-4}}$$

Asumiendo que este es excitado por ruido blanco gaussiano de media nula y varianza unitaria denotado como X_{nn}, dando lugar a una señal $Y(n)$.

Denotaremos por N al número de muestras disponibles del proceso X y consideraremos el estimador de Blackman-Tuckey:

$$\hat{S}_{BT}(e^{j\omega}) = \sum_{k=-(M-1)}^{M-1} w(k)\hat{R}(k)e^{-j\omega k}$$

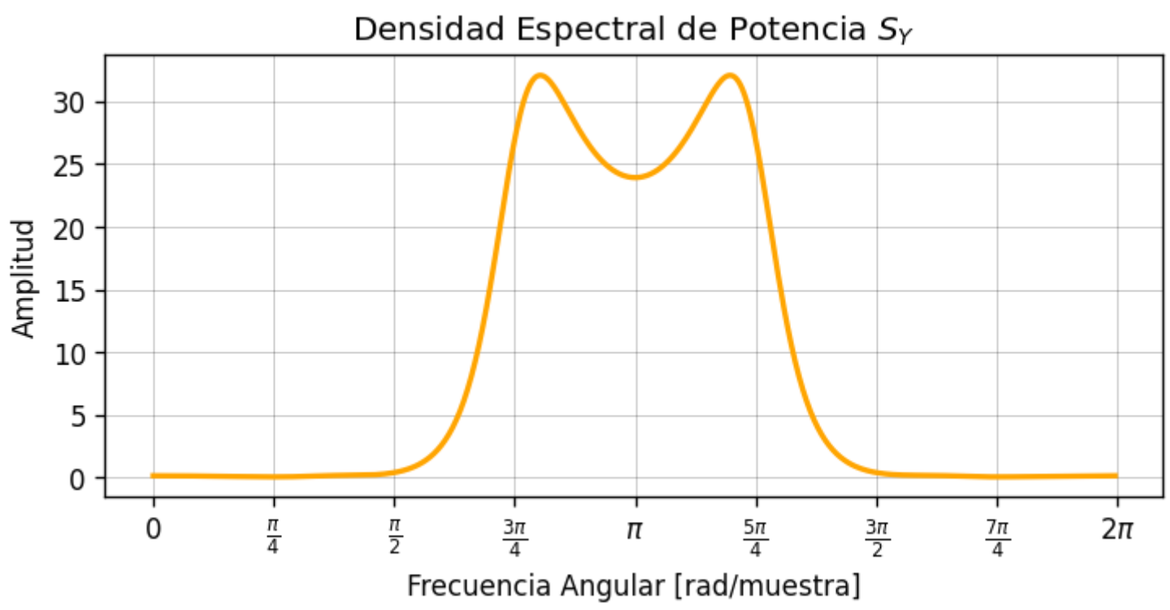
Tomando $w(k) = 1$ para todo k y $M = N$ se tiene el periodograma o correlograma clásico.

1.2.1 - Gráfico de Densidad Espectral de Potencia

Primeramente, graficamos la verdadera densidad espectral de potencia de Y.

$$S_Y(e^{j\omega}) = |H(e^{jw})|^2 S_X(e^{j\omega}) = |H(e^{jw})|^2$$

```
import numpy as np
from scipy.ndimage.interpolation import shift
import matplotlib.pyplot as plt
from scipy import signal
from scipy.signal import correlate
import numpy.fft as fft

h_denom = np.array([1, 0.3544, 0.3508, 0.1736, 0.2401])
h_num = np.array([1, -1.3817, 1.5632, -0.8443, 0.4096])

[eje_freq, Syy_teo] = signal.freqz(h_num, h_denom, whole=True) #En frecs rad/sample * 2pi

plt.figure(dpi=120, figsize = (7,3))
plt.plot(eje_freq, ((pow(np.abs(Syy_teo), 2))), linewidth = 2, color = 'orange')
plt.title(r'Densidad Espectral de Potencia $S_Y$')
plt.xlabel('Frecuencia Angular [rad/muestra]')
plt.ylabel('Amplitud')
plt.xticks([0, np.pi/4, np.pi/2, 3*np.pi/4, np.pi, 5*np.pi/4, 3*np.pi/2, 7*np.pi/4, 2*np.pi], ['$0$', r'$\frac{\pi}{4}$', r'$\frac{\pi}{2}$', r'$\frac{3\pi}{4}$', r'$\pi$', r'$\frac{5\pi}{4}$', r'$\frac{

plt.grid(b=True, color = 'black', linestyle = '-', linewidth = 0.4, alpha = 0.3)
```



1.2.2 - Estimación por Periodograma

Ahora, con 100 realizaciones de largo N de X, obtenemos para cada una el periodograma correspondiente y calculamos el periodograma promedio y su varianza.

$$\bar{\hat{S}}_Y(e^{j\omega}) = \frac{1}{J}\sum_{i=1}^{J} \hat{S}_{Y_i}(e^{j\omega})$$

$$\sigma^2_{\hat{S}_Y}(e^{j\omega}) = \frac{1}{J-1}\sum_{i=1}^{J} (\hat{S}_{Y_i}(e^{j\omega}) - \bar{\hat{S}}_Y(e^{j\omega}))^2$$

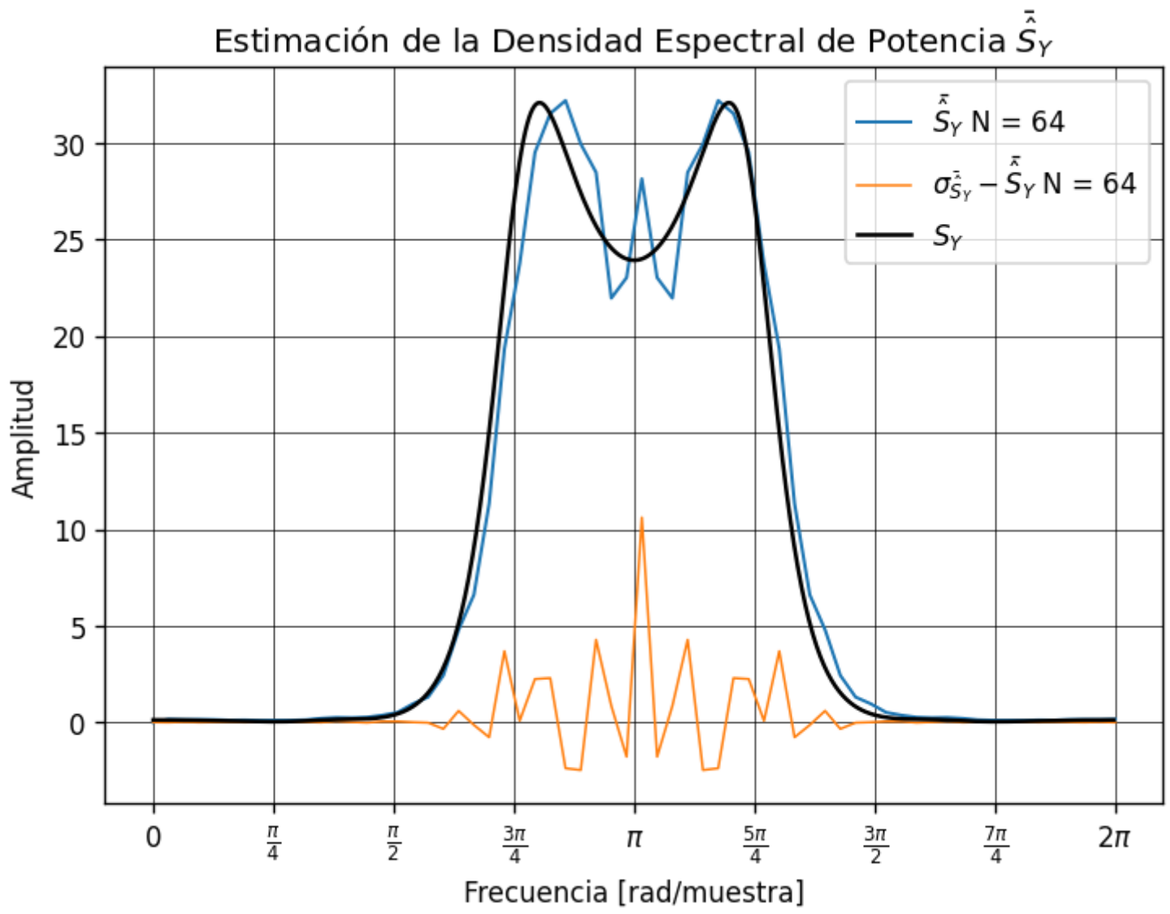
Se simulan entonces las curvas para $N = 64$ y $N = 512$, comparando así su comportamiento con el desarrollo teórico.

```
#X: Ruido de entrada
# Cantidad de realizaciones del proceso X
J = 100
# Cantidad de muestras de cada realización del proceso X
N = np.array([64, 512]) # N1 y N2
```

```
plt.figure(figsize=(7,5), dpi=120)

X = np.random.normal(0,1,(J,N[0]))
w = np.linspace(0,2*np.pi, N[0])

#Filtro de X con H(z)
Y = signal.lfilter(h_num, h_denom, X, axis = 1)

#Media de las J estimaciones del periodograma de Sy
Syy_prom = aux = np.zeros(int(N[0]))

for j in range(J):
    #Periodograma de Y en frecuencia rad/sample normalizado
    # con nfft = n
    (wyy, syy) = signal.periodogram(Y[j], axis = -1, return_onesided=False, scaling = 'density', nfft = N[0])
    aux += syy #Sumo las syy
Syy_prom = aux/J #Las promedio por J

#Varianza:
Syy_sigma = aux = np.zeros(int(N[0]))
#Iterar de vuelta y restar cada periodograma por la Syy_promedio
for j in range(J):
    #Periodograma de Y en frecuencia rad/sample normalizado
    # con nfft = n
    (wyy, syy) = signal.periodogram(Y[j], return_onesided=False, scaling = 'density', nfft = N[0])
    aux += pow( (syy - Syy_prom), 2 )
Syy_sigma = aux/(J-1) #Las promedio por J-1

plt.plot(w, Syy_prom, label = r'$\bar{\hat{S}}_Y$ ' + "N = {}".format(N[0]), linewidth=1.2)
```

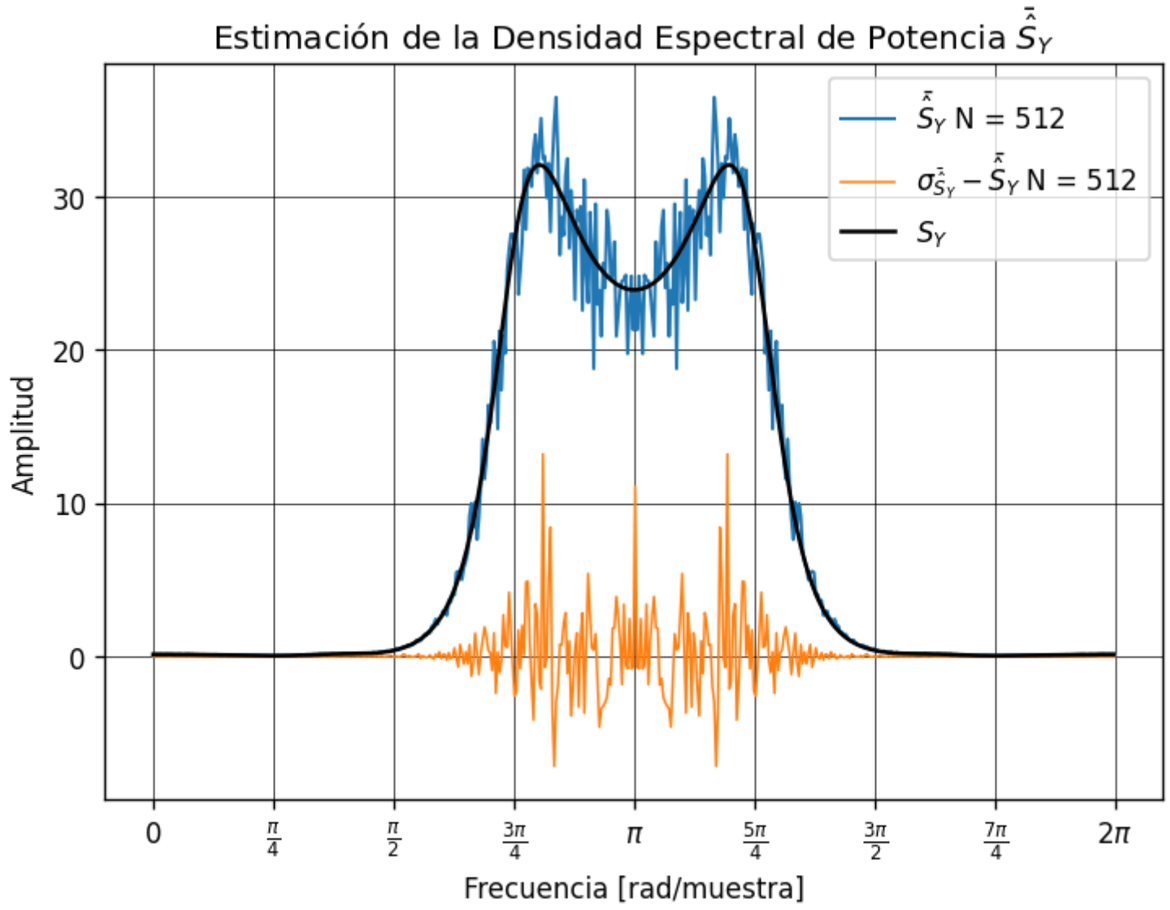
```
plt.plot(w, Syy_prom, label = r'$\bar{\hat{S}}_Y$ ' + "N = {}".format(N[0]), linewidth=1.2)
plt.plot(w, pow(Syy_sigma,1/2) - Syy_prom, label = r'$\sigma_{\bar{\hat{S}}_Y} - \bar{\hat{S}}_Y$ ' + "N = {}".format(N[0]), linewidth=0.9)

#Teórica
#plt.plot(w, (pow(np.abs(Syy_teo), 2) ), label = r'$S_Y$', color = 'black', linestyle = '-')
plt.plot(eje_freq, ((pow(np.abs(Syy_teo), 2))), label = r'$S_Y$', linestyle = '-', color = 'black')

plt.title(r'Estimación de la Densidad Espectral de Potencia $\bar{\hat{S}}_Y$')
plt.xlabel('Frecuencia [rad/muestra]')
plt.ylabel('Amplitud')
plt.xticks([0, np.pi/4, np.pi/2, 3*np.pi/4, np.pi, 5*np.pi/4, 3*np.pi/2, 7*np.pi/4, 2*np.pi], ['$0$', r'$\frac{\pi}{4}$', r'$\frac{\pi}{2}$', r'$\frac{3\pi}{4}$', r'$\pi$', r'$\frac{5\pi}{4}$', r'$\frac{
plt.grid(b=True, color = 'black', linestyle = '-', linewidth = 0.4)
plt.rc('font', size=10)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa3c62e5b50>
```



AFFICHER LE CODE

```
<matplotlib.legend.Legend at 0x7fa3c4af1910>
```



Podemos observar que el desvío del estimador $\sigma_{\hat{S}_Y}$ aumenta con el incremento de N. Esto lleva a que haya dispersión en la estimación, particularmente en las frecuencias $(0; \pi/4)$.

1.2.3 - Comparación con el estimador de Blackman-Tukey

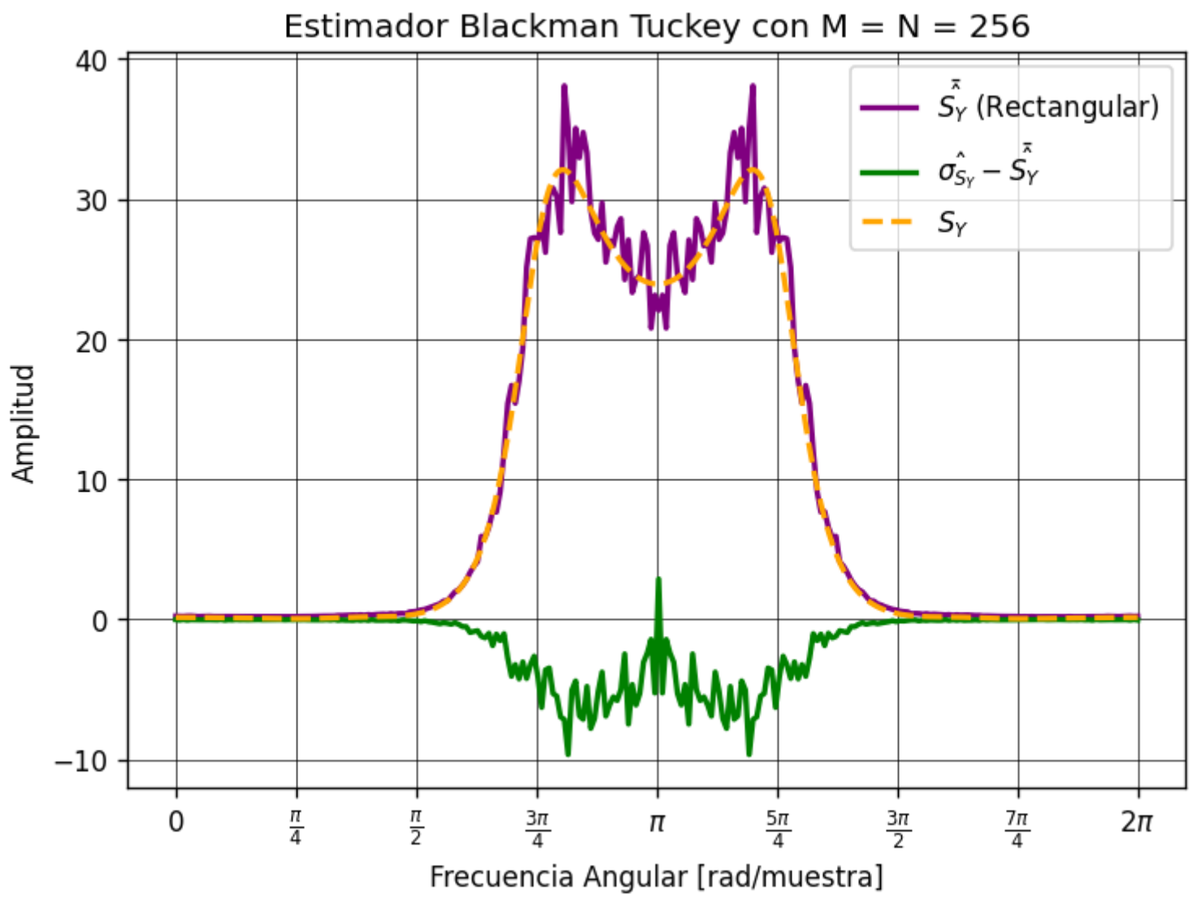
Ahora, con N = 256, se realiza el mismo tipo de gráfico en las condiciones del punto anterior. Pero superpondremos también los gráficos de las mismas curvas al obtenerse mediante una ventana de Bartlett con $M = N/4$ y $M = N/16$.

$$w(k) = (1 - \frac{|k|}{M})\mathbf{1}|k| \leq M$$

```
#Redefinimos el estimador sesgado de la correlación, entre -k y k
def estim_corr_sesgado(y):
    r = correlate(y,y)/len(y)
    l = np.arange(-(len(y)-1), len(y) )
    return r,l

#Definimos el estimador Blackman-Tuckey:
def blackman_tuckey(y,w, nfft):
    N = len(y)
    M = len(w)

    #Estimacion
    ryy, k = estim_corr_sesgado(y)
    #Ventaneo
    ryy = ryy[np.logical_and(k >= -(M/2), k < M/2)]
    ryw = ryy * w
    #FFT:
    Y = fft.fft(ryw, nfft)
```

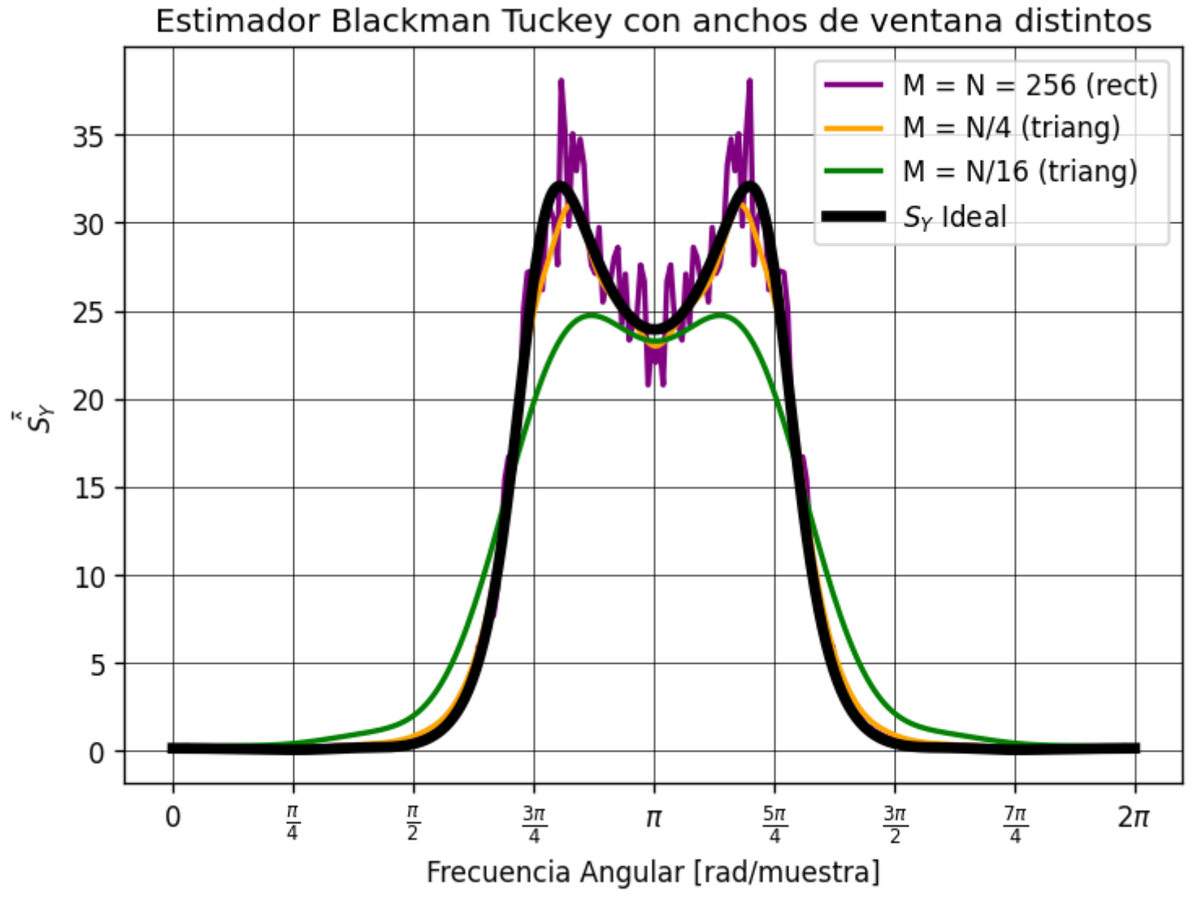
```
    f = np.arange(nfft)/nfft

    return (pow(np.abs(Y), 1), f)

#Definimos
nfft = 256
M2 = int(nfft/16)
M3 = int(nfft/4)

win_box = signal.boxcar(nfft)
win_tri1 = np.bartlett(M2)
win_tri2 = np.bartlett(M3)

X = np.random.normal(0,1, (J,256) )
Y = signal.lfilter(h_num, h_denom, X, axis=1)

#Estimador con M = N y ventana rectangular
psd_mean = aux = np.zeros(int(nfft))
for j in range(J):
    (psd, w) = blackman_tuckey(Y[j], win_box, nfft)
    aux += psd
psd_mean = aux/J #Estimador

#Varianza
sigma_psd = aux = np.zeros(int(nfft))
for j in range(J):
    (psd, w) = blackman_tuckey(Y[j], win_box, nfft)
    aux += pow((psd - psd_mean), 2)
sigma_psd /= (J-1) #Varianza

plt.figure(figsize=(7,5), dpi=120)

#ver plots
plt.plot(np.linspace(0, 2*np.pi, nfft), psd_mean, linewidth=2, color='purple', label=r'$\bar{\hat{S_Y}}$ (Rectangular)')
plt.plot(np.linspace(0, 2*np.pi, nfft), pow(sigma_psd, 1/2) - psd_mean, linewidth=2, color = 'green', label=r'$\hat{\sigma_{S_Y}}-\bar{\hat{S_Y}}$')
plt.plot(np.linspace(0, 2*np.pi, 512), (pow(np.abs(Syy_teo), 2)), linewidth=2, linestyle='dashed',color='orange',label=r'$S_Y$')

plt.title('Estimador Blackman Tuckey con M = N = 256')
plt.xlabel('Frecuencia Angular [rad/muestra]')
plt.ylabel('Amplitud')
plt.xticks([0, np.pi/4, np.pi/2, 3*np.pi/4, np.pi, 5*np.pi/4, 3*np.pi/2, 7*np.pi/4, 2*np.pi], ['$0$', r'$\frac{\pi}{4}$', r'$\frac{\pi}{2}$', r'$\frac{3\pi}{4}$', r'$\pi$', r'$\frac{5\pi}{4}$', r'$\frac{

plt.grid(b=True, color = 'black', linestyle = '-', linewidth = 0.4)
plt.rc('font', size=10)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa3c53d4810>
```



AFFICHER LE CODE

```
<matplotlib.legend.Legend at 0x7fa3c5306b50>
```



Se puede observar una menor dispersión en los datos, particularmente al disminuir M, por lo que tenemos una curva más suave. Sin embargo, este decremento de M nos lleva a perder resolución. Se puede notar entonces que el caso de $M = N/4$ es el que mejor se aproxima al real,

con una diferencia máxima de 3dB aproximadamente.

2. Técnicas paramétricas

2.1. Ejercicio teórico

Se tiene que la función de versimilitud de $\theta = [\sigma_W^2, \mathbf{a}]$ es:

$$L(\sigma_W^2, \mathbf{a}) = \frac{1}{(2\pi\sigma_W^2)^{\frac{N-m}{2}}} \exp\left\{ -\frac{1}{2\sigma_W^2} \|\tilde{\mathbf{y}} - \mathbf{Y}^T\mathbf{a}\|^2 \right\}$$

Para maximizar la función respecto de θ podemos maximizar en su lugar su logaritmo natural:

$$\log L(\sigma_W^2, \mathbf{a}) = -\frac{N-m}{2}\log(2\pi\sigma_W^2) - \frac{1}{2\sigma_W^2}\|\tilde{\mathbf{y}} - \mathbf{Y}^T\mathbf{a}\|^2$$

Luego, una de las cosas que se tiene que hacer es minimizar el segundo término ya que está restando. Por lo tanto se busca

$$\|\tilde{\mathbf{y}} - \mathbf{Y}^T\mathbf{a}\|^2 = 0$$

que implica

$$\begin{aligned} \tilde{\mathbf{y}} - \mathbf{Y}^T\mathbf{a} &= 0 \\ \tilde{\mathbf{y}} &= \mathbf{Y}^T\mathbf{a}. \end{aligned}$$

Después multiplicamos ambos lados por $\mathbf{Y}$:

$$\mathbf{Y}\tilde{\mathbf{y}} = \mathbf{Y}\mathbf{Y}^T\mathbf{a}.$$

Si $\mathbf{Y}\mathbf{Y}^T$ es inversible, podemos hallar una única solución para $\hat{\mathbf{a}}_{MV}$:

$$\hat{\mathbf{a}}_{MV} = (\mathbf{Y}\mathbf{Y}^T)^{-1}\mathbf{Y}\tilde{\mathbf{y}}.$$

Una vez obtenido $\hat{\mathbf{a}}_{MV}$ podemos obtener $\sigma^2_{W,\,MV}$ derivando $\log L(\sigma_W^2, \hat{\mathbf{a}}_{MV})$ respecto a dicho parámetro e igualando a 0:

$$\begin{aligned} \frac{\partial f}{\partial \sigma^2_{W,\,MV}} \log L(\sigma^2_{W,\,MV}, \hat{\mathbf{a}}_{MV}) &= -\frac{N-m}{2\sigma^2_{W,\,MV}} + \frac{1}{2\sigma^4_{W,\,MV}}\|\tilde{\mathbf{y}} - \mathbf{Y}^T\hat{\mathbf{a}}_{MV}\|^2 \\ &= 0. \end{aligned}$$

Despejando se obtiene:

$$\sigma^2_{W,\,MV} = \frac{\|\tilde{\mathbf{y}} - \mathbf{Y}^T\hat{\mathbf{a}}_{MV}\|^2}{N-m}.$$

2.2. Simulación de proceso AR-4

Se tienen muestras {Y(1),...,Y(N)} generadas a partir de un proceso AR-4 con la siguiente ecuación en diferencias:

$$Y(n) + 0.3544Y(n-1) + 0.3508Y(n-2) + 0.1736Y(n-3) + 0.2401Y(n-4) = W(n)$$

Donde W es ruido blanco Gaussiano de varianza unitaria. El objetivo es utilizar el criterio de Akaike para identificar el orden del modelo, sus coeficientes, y finalmente estimar su densidad espectral de potencia.

En el contexto del modelado autoregresivo se tiene que los modelos posibles son procesos AR-m, con $m = 1, 2, \ldots$ Por lo que el procedimiento según el criterio de Akaike para determinar el valor de m es el siguiente:

- Se parte de una única muestra $Y(1), \ldots, Y(N)$.
- Para cada orden m=1,2, del AR-m posible, se halla el estimador de máxima verosimilitud $\hat{a}_{MV}$ y la varianza $\hat{\sigma}^2_{W,MV}$. Luego, se calcula la métrica de Akaike:

$$AIC(G_i) = 2K_i - 2log(L(\hat{\theta}_{i,MV}))$$

- Finalmente, luego de hacer esto para distintos valores de m, se elige aquel orden y aquel modelo que minimiza la métrica.

2.3 Comparación entre el periodograma puro y el estimador de Welch

Considerando las muestras $Y(1), \ldots Y(N)$ del sistema AR-4 del ejercicio anterior, se procede a estimar la PSD comparando el método del periodograma puro y el del estimador de Welch. En todos los casos se aplica una FFT de 5000 puntos.

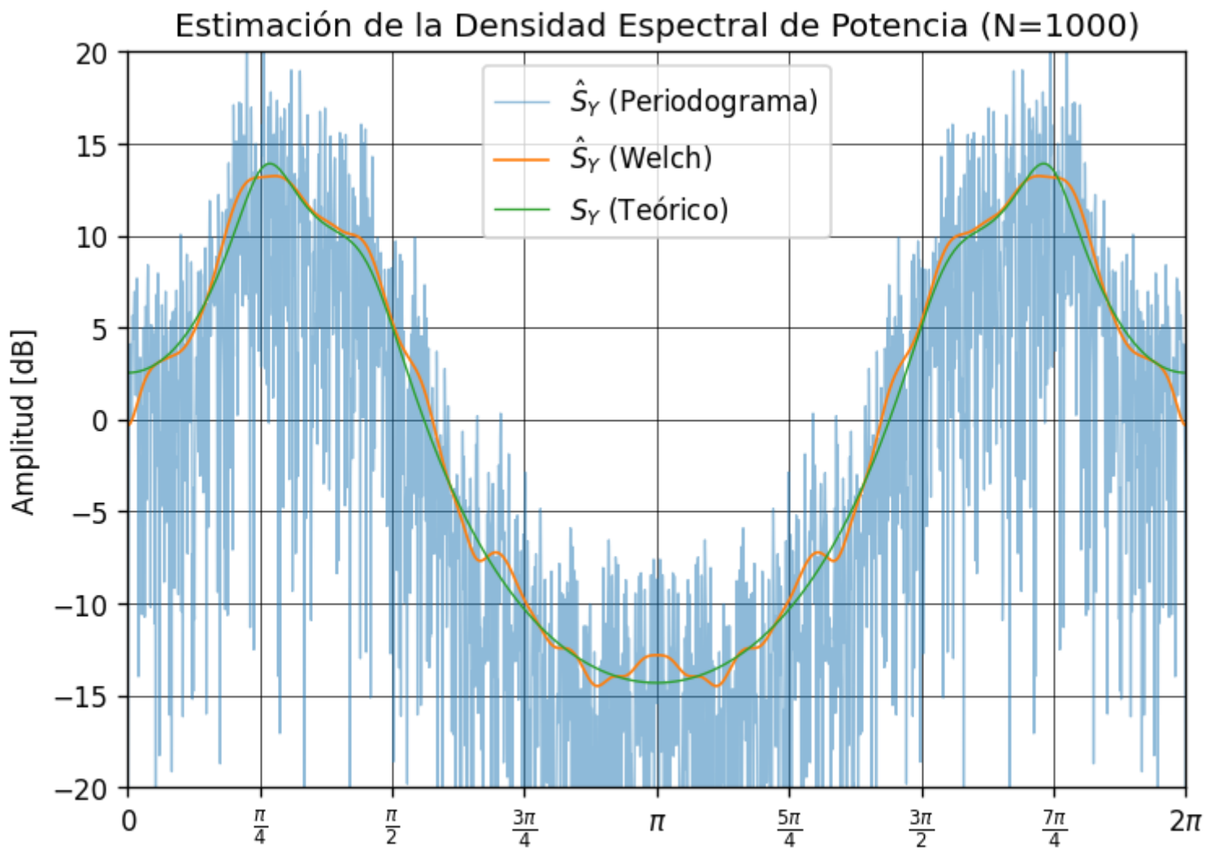
N = 1000

Para N = 1000 se genera una realización de las muestras de Y y se utiliza para obtener el estimador de Welch. Se emplean segmentos de 50 muestras con solapamiento del 50 %.

```
#@title
## Estimador de maxima verosimilitud de los coeficientes de a :
def estimadorCoefMV(y, m):

    N = y.size

    # y monio es (N-m-1)x1
    yy = np.zeros((N-m,1))
    for i in range(N-m):
        yy[i] = y[i+m]
    yy = yy[::-1]

    # para m <= n <= N se define y(n) = [y(n),...,y(n-m+1)] mx1
    YY = np.zeros((m,N-m))
    for i in range(m):
        for j in range(N-m):
            YY[i][j] = y[N-j-i-2]
    # ecuacion normal
    a_mv = np.dot( np.dot( np.linalg.inv( np.dot(YY,YY.T) ) , YY ) , yy )
    # En el calculo se definieron los coeficientes como negativos 1 - a1*Y(n-1) - a2*Y(n-2) - a3*Y(n-3) - a4*Y(n-4)
    # Entonces como en teoria son positivos, a_mv los devuelve negativos
    sigma_mv = pow(np.linalg.norm( yy - np.dot(YY.T,a_mv) ),2)/(N-m)

    return sigma_mv, a_mv

# Metrica de Akaike
def AIC(m, sigma):
    return 2 * (m + 1) + 2 * N * (1+np.log(2*np.pi*sigma))
```

AFFICHER LE CODE

```
<matplotlib.legend.Legend at 0x7fa3d26b7ed0>
```



N = 5000

Para N = 5000 se usan ventanas de 250 muestras con solapamiento del 50 %. En todos los casos utilice una FFT de 5000 puntos.

```
#@title
h_num = np.array([1])
h_denom = np.array([1, -1.3817, 1.5632, -0.8443, 0.4096])

#X: Ruido de entrada
# Cantidad de realizaciones del proceso X
J = 1
# Cantidad de muestras de cada realización del proceso X
N = np.array([1000, 5000]) # N1 y N2

plt.figure(figsize=(7,5), dpi=120)

#Entra ruido normal de varianza unitaria
X = np.random.normal(0,1,N[1])
w = np.linspace(0,2*np.pi, 5000)

#Filtro la entrada X con H(z)
Y = signal.lfilter(h_num, h_denom, X, axis=0)

#Periodograma y Welch
(wyy, Syy_period) = signal.periodogram(Y, nfft = 5000, axis = -1, return_onesided=False, scaling = 'density')
(wyy, Syy_welch) = signal.welch(Y, nperseg=250, noverlap=125, nfft = 5000, window="bartlett", axis = -1, return_onesided=False,

#Teórico
[eje_freq, Syy_teo] = signal.freqz(h_num, h_denom, whole=True)
Syy_teo = pow(np.abs(Syy_teo), 2)

#######
#######
#Plots
plt.plot(w, 10*np.log10(Syy_period), label = r'${\hat{S}}_Y$ ' + "(Periodograma)".format(N[1]), linewidth=0.8, alpha = 0.5)
plt.plot(w, 10*np.log10(Syy_welch), label = r'${\hat{S}}_Y$ ' + "(Welch)".format(N[1]), linewidth=1)
plt.plot(eje_freq, 10*np.log10(Syy_teo), label = r'$S_Y$ ' + "(Teórico)".format(N[1]), linewidth=0.8)
#plt.plot(w, 10*np.log10(Syy_welch))


plt.title(r'Estimación de la Densidad Espectral de Potencia (N=5000)')
plt.xlabel('Frecuencia Angular [rad/muestra]')
plt.ylabel('Amplitud [dB]')
plt.xticks([0, np.pi/4, np.pi/2, 3*np.pi/4, np.pi, 5*np.pi/4, 3*np.pi/2, 7*np.pi/4, 2*np.pi], ['$0$', r'$\frac{\pi}{4}$', r'$\f

plt.ylim([-20,20])
plt.xlim([0, 2*np.pi])
plt.grid(b=True, color = 'black', linestyle = '-', linewidth = 0.4)
plt.rc('font', size=10)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa3c5293610>
```

Observaciones

1) En el caso del periodograma, para ambos valores de N se tienen valores con mucha dispersión. Siendo la mejor entre ambas aquella correspondiente a $N = 1000$. Tiene sentido notar una gran dispersión puesto que estamos utilizando una sola realización, en vez de realizar varias y promediar el resultado, como en el punto **1.2.2**. 2) Para el caso del estimador de Welch, este presenta una menor varianza comparado con el periodograma, también comportandose mejor para el caso de $N = 1000$. Para este N tenemos menor varianza y un mejor ajuste a la PSD real, pero se introducen algunos lóbulos igualmente.

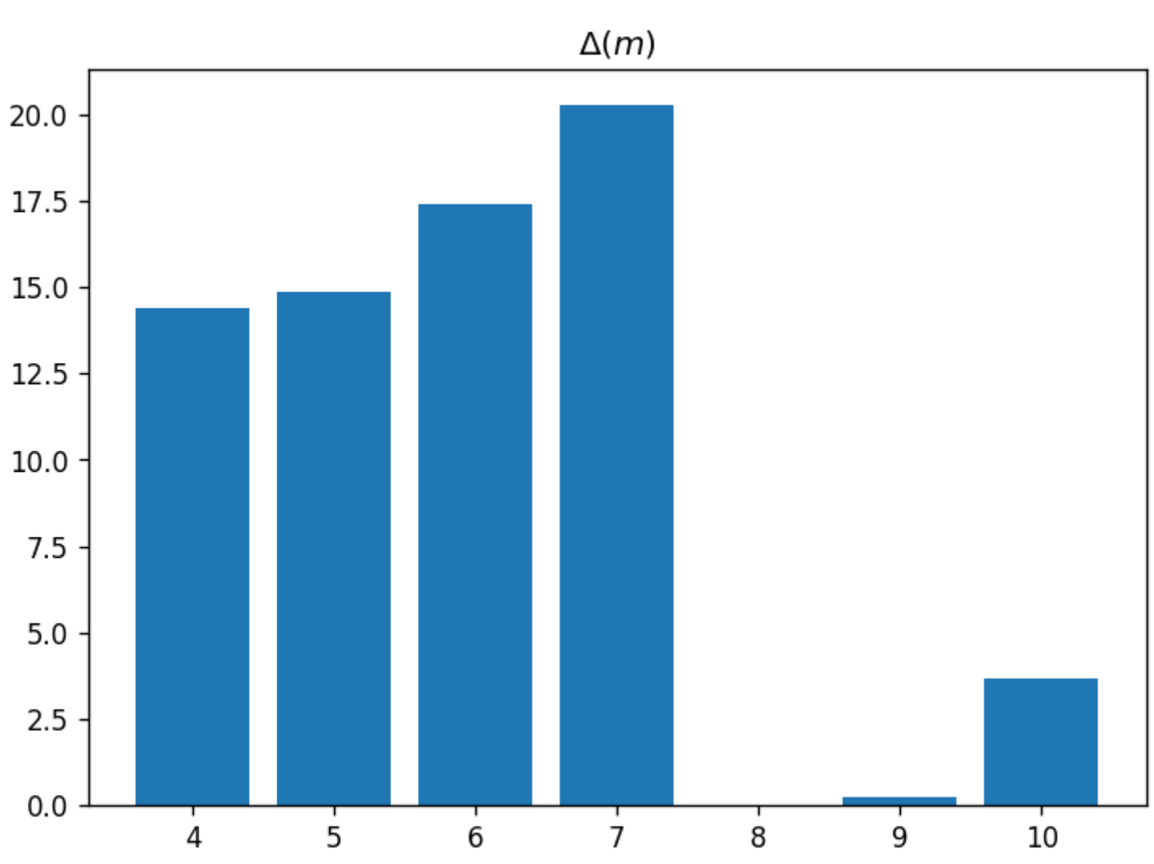
2.4 Comparación entre el estimador MV y el estimador de Welch

Para $N = 1000$ se genera una realización de las muestras de Y (del ejercicio **2.2**) *texte en italique* y se utiliza para obtener el estimador de Welch y el estimador MV de orden $1 \leq m \leq 10$. Se utiliza además la métrica de Akaike para determinar el orden del modelo

2.4.1. Tabla de coeficientes

Siguiendo el método descripto en la sección 2.2 para aplicar el criterio de Akaike, se obtuvieron los siguientes valores de $\Delta(m) = AIC(m) - min_{1 \leq m \leq 10}(AIC(m))$, eligiendo como m de Akaike a aquella que se anula.

```
#@title
h_num = np.array([1])
h_denom = np.array([1, -1.3817, 1.5632, -0.8443, 0.4096])

#X: Ruido de entrada
# Cantidad de muestras de cada realización del proceso X
N = 1000

#Entra ruido normal de varianza unitaria
X = np.random.normal(0,1,N)

#Filtro la entrada X con H(z)
Y = signal.lfilter(h_num, h_denom, X, axis=0)

###Akaike###
theta = [estimadorCoefMV(Y, m) for m in range(1, 11)]   # theta = (sigma, a_mv)

AIC_var = np.array([AIC(m, theta[m-1][0]) for m in range(1, 11)])
delta_AIC = AIC_var - AIC_var.min()

#guardamos el orden m que propone el modelo
for i in range(delta_AIC.size):
  if(delta_AIC[i]==0):
    m_aic = i+1

plt.figure(figsize=(7,5), dpi=120)
plt.bar(range(4, 11), delta_AIC[3:])
plt.title('$\Delta (m)$');
```




```
print(h_denom)
```

```
[ 1.     -1.3817  1.5632 -0.8443  0.4096]
```

```
print(np.append(1, -theta[4-1][1])) #coeficientes de MV de orden 4
```

```
[ 1.         -1.40583203  1.63962402 -0.93908979  0.45463081]
```

```
print(np.append(1, -theta[m_aic-1][1]))  #coeficientes de MV por criterio de Akaike
```

```
[ 1.         -1.38981527  1.61080814 -0.89044727  0.43696749 -0.01364447
  0.06496634 -0.04834291  0.0264996 ]
```

```
a_mv_4 = np.append(1, -theta[4-1][1])       #coef de orden 4
a_mv_Ak = np.append(1, -theta[m_aic-1][1])  #coef de Akaike

#@title
import plotly.graph_objects as go

fig = go.Figure(data=[go.Table(header=dict(values=[['Coef. verdaderos'], ['Coef. del orden 4'], ['Coef. de Akaike (orden {})'.format(m_aic)]]),
                 cells=dict(values=[h_denom, a_mv_4, a_mv_Ak],
                 align = 'left'))
                     ])

fig.update_layout(width=750)
fig.show()
```

Coef. verdaderos	Coef. del orden 4	Coef. de Akaike (orden 8)
1	1	1
-1.3817	-1.4058320344221795	-1.3898152676455469
1.5632	1.6396240236004496	1.6108081426369183
-0.8443	-0.9390897916625681	-0.8904472697163837
0.4096	0.45463080801122246	0.4369674851200661
		-0.013644474270914614
		0.06496634358120122
		-0.048342907505916306
		0.026499597768324598

2.4.2. Soporte empírico para cada orden m

El valor de la métrica $\Delta(m)$ indica el grado de soporte que le dan las muestras al modelo de orden m en relación al modelo elegido según la siguiente tabla.

$\Delta(m)$	Soporte empírico para el modelo de orden m
0-2	Sustancial
4-7	Considerablemente menos que el elegido
>10	Esencialmente ningún soporte empírico

A continuación se presenta el soporte para cada m registrado.

```
#@title
import plotly.graph_objects as go

def soporte(delta_AIC):
  #recibe el vector de deltas y devuelve un vector de soportes
  soporte = []
  for i in range(len(delta_AIC)):
    if np.float(delta_AIC[i]) <= 2: soporte.append('Sustancial')
    elif np.float(delta_AIC[i]) <= 10: soporte.append('Considerablemente menos que el elegido')
    else: soporte.append('Esencialmente ningún soporte empírico')
  return soporte

fig = go.Figure(data=[go.Table(header=dict(values=[['$\Delta (m)$'], ['Soporte empírico para el modelo de orden m']]),
                 cells=dict(values=[['$\Delta ({}) = {}$'.format(m, delta_AIC[m-1]) for m in range(1,11)],
                                    soporte(delta_AIC)],
                 align = 'left'))
                     ])

fig.update_layout(width=750)
fig.show()
```

$\Delta(m)$	Soporte empírico para el modelo de orden m
$\Delta(1) = 2583.5501878276573$	Esencialmente ningún soporte empírico
$\Delta(2) = 778.0967025249684$	Esencialmente ningún soporte empírico
$\Delta(3) = 473.848398943901$	Esencialmente ningún soporte empírico
$\Delta(4) = 14.37629137654767$	Esencialmente ningún soporte empírico
$\Delta(5) = 14.871996731346371$	Esencialmente ningún soporte empírico
$\Delta(6) = 17.400225078631593$	Esencialmente ningún soporte empírico
$\Delta(7) = 20.264938114004508$	Esencialmente ningún soporte empírico
$\Delta(8) = 0.0$	Sustancial
$\Delta(9) = 0.23685322561868816$	Sustancial
$\Delta(10) = 3.6642117817791586$	Considerablemente menos que el elegido

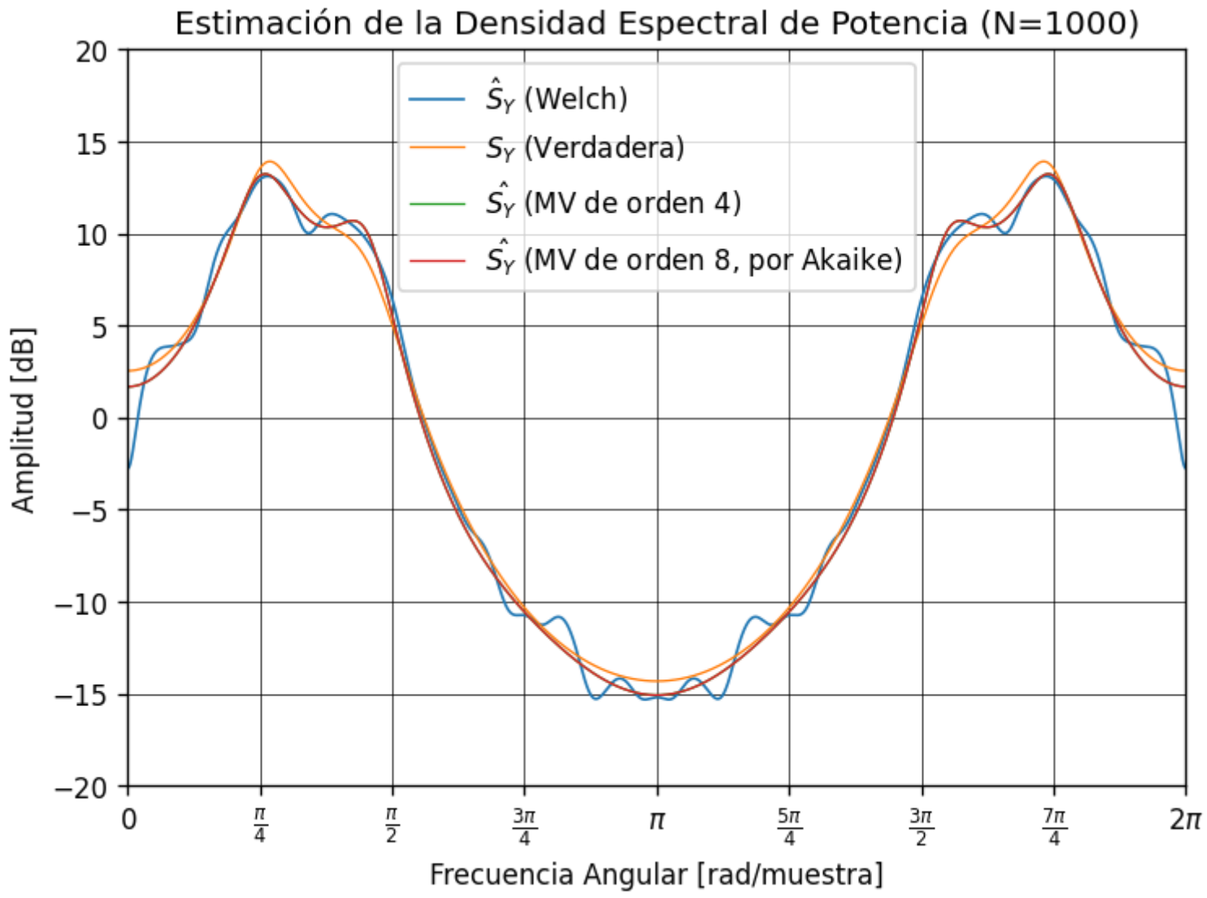
2.4.3. Gráfico comparativo entre Welch, PSD Verdadera, y estimadores de MV de orden 4 y por criterio de Akaike

Comparamos a continuación dichos estimadores mediante un gráfico en dB.

```
#@title
plt.figure(figsize=(7,5), dpi=120)

#Periodograma y Welch
(wyy, Syy_period) = signal.periodogram(Y, nfft = 5000, axis = -1, return_onesided=False, scaling = 'density')
(wyy, Syy_welch) = signal.welch(Y, nperseg=50, noverlap=25, nfft = 5000, window="bartlett", axis = -1, return_onesided=False, s

#Teórico
[eje_freq, Syy_teo] = signal.freqz(h_num, h_denom, whole=True)
Syy_teo = pow(np.abs(Syy_teo), 2)

#MV de orden 4
[eje_freq_mv, Syy_mv4] = signal.freqz(theta[3][0], np.append(1, -theta[3][1]), whole=True)
Syy_mv4 = pow(np.abs(Syy_mv4), 2)

#Akaike
[eje_freq_ak, Syy_ak] = signal.freqz(theta[m_aic-1][0], np.append(1, -theta[m_aic-1][1]), whole=True)
Syy_ak = pow(np.abs(Syy_ak), 2)

……….
```

```
#######
#######
#Plots
#plt.plot(w, 10*np.log10(Syy_period), label = r'${\hat{S}}_Y$ ' + "(Periodograma)", linewidth=0.8, alpha = 0.5)
plt.plot(np.linspace(0,2*np.pi, 5000), 10*np.log10(Syy_welch), label = r'${\hat{S}}_Y$ ' + "(Welch)", linewidth=1)
plt.plot(eje_freq, 10*np.log10(Syy_teo), label = r'$S_Y$ ' + "(Verdadera)", linewidth=0.8)

plt.plot(eje_freq_mv, 10*np.log10(Syy_ak), label = r'$\hat{S_Y}$ ' + "(MV de orden {})".format(4), linewidth=0.8)
plt.plot(eje_freq_ak, 10*np.log10(Syy_ak), label = r'$\hat{S_Y}$ ' + "(MV de orden {}, por Akaike)".format(m_aic), linewidth=0.



plt.title(r'Estimación de la Densidad Espectral de Potencia (N=1000)')
plt.xlabel('Frecuencia Angular [rad/muestra]')
plt.ylabel('Amplitud [dB]')
plt.xticks([0, np.pi/4, np.pi/2, 3*np.pi/4, np.pi, 5*np.pi/4, 3*np.pi/2, 7*np.pi/4, 2*np.pi], ['$0$', r'$\frac{\pi}{4}$', r'$\f

plt.ylim([-20,20])
plt.xlim([0, 2*np.pi])
plt.grid(b=True, color = 'black', linestyle = '-', linewidth = 0.4)
plt.rc('font', size=10)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa3c4d7fd10>
```



Observaciones

Podemos observar que el estimador MV de orden 4 y el de orden obtenido con el criterio de Akaike mantienen una excelente resolución y semejanza al comportamiento verdadero, particularmente entre $[\pi/2; \pi]$. Es decir, son más precisos que utilizar un estimador de la correlación.

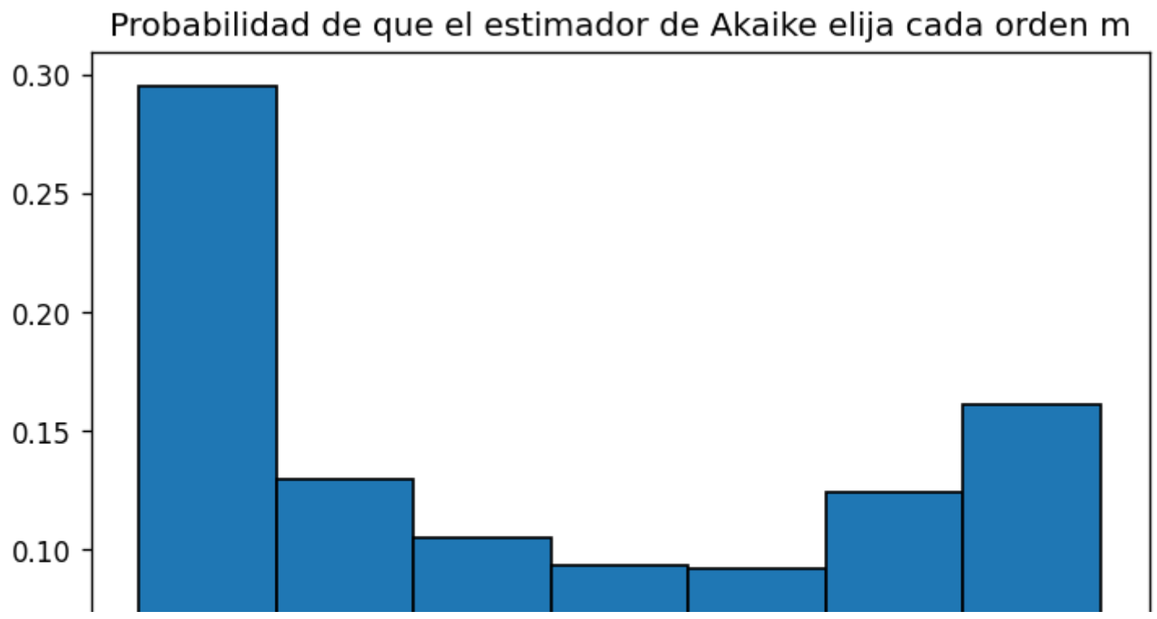
2.4.4. Histograma para cada orden posible según el criterio de Akaike

Finalmente, se repite 2000 veces el experimento de generar N=1000 muestras de Y(n). Calculamos sus coeficientes para distintos m, y estimos el orden del modelo usando el criterio de Akaike. Con esto se obtiene un histograma y se determina la probabilidad de que dicho criterio elija cada m posible entre 1 y 10.

```
#@title
h_num = np.array([1])
h_denom = np.array([1, -1.3817, 1.5632, -0.8443, 0.4096])

#X: Ruido de entrada
# Cantidad de muestras de cada realización del proceso X
N = 1000
J = 2000

#Entra ruido normal de varianza unitaria
X = [np.random.normal(0,1,N) for i in range(J)]

#Filtro la entrada X con H(z)
Y = [signal.lfilter(h_num, h_denom, X[i], axis=0) for i in range(J)]

###Akaike###
theta = [[estimadorCoefMV(Y[i], m) for m in range(1, 11)] for i in range(J)]   # theta = (sigma, a_mv)

AIC_var = [np.array([AIC(m, theta[i][m-1][0]) for m in range(1, 11)]) for i in range(J)]

#guardamos el orden m que propone el modelo
m_aic = []
for i in range(J):
  min = np.min(AIC_var[i])
  for j in range(10):
    if(AIC_var[i][j]==min):
      m_aic.append(j+1)
      break
```

```
#Histograma
plt.figure(figsize=(7,5), dpi=120)
plt.hist(m_aic, bins = [4,5,6,7,8,9,10,11], density = True, align = 'left', ec = 'black')
plt.title('Probabilidad de que el estimador de Akaike elija cada orden m')
plt.xlabel('Orden predicho (m)')
```

```
Text(0.5, 0, 'Orden predicho (m)')
```

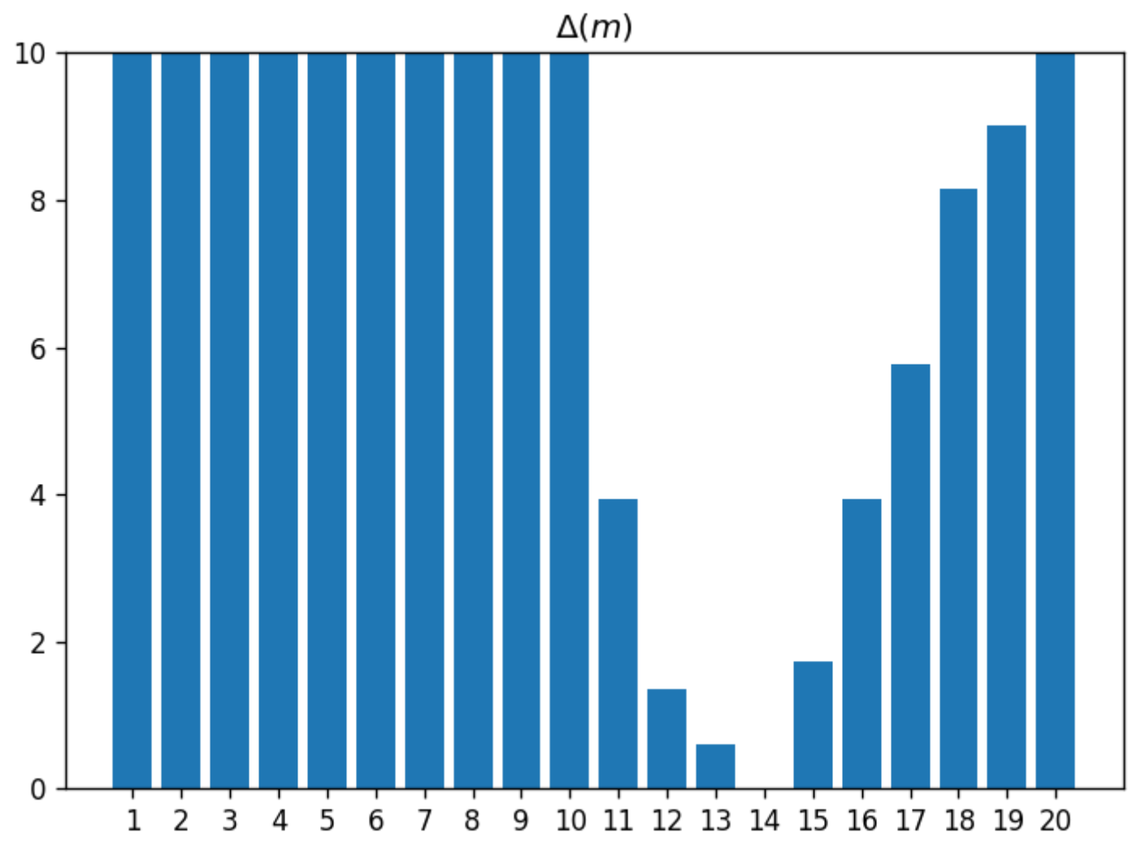


2.5. Aplicacion a un proceso con modelo desconocido

El archivo Ej4.csv (ruta: */content/Ej4.csv*) contiene 5000 muestras de un proceso estocástico cuuyo modelo es desconocido. Por ello, se busca obtener un modelo AR de la señal a partir del estimador MV utilizando la técnica de Akaike para estimar su orden. También se realiza un gráfico comparando la PSD del proceso AR obtenido con la que se obtiene con el estimador de Welch. Se considera que los órdenes posibles están en el rango $1 \leq m \leq 20$.

2.5.1. Busqueda del estimador de MV por Criterio de Akaike

AFFICHER LE CODE



AFFICHER LE CODE

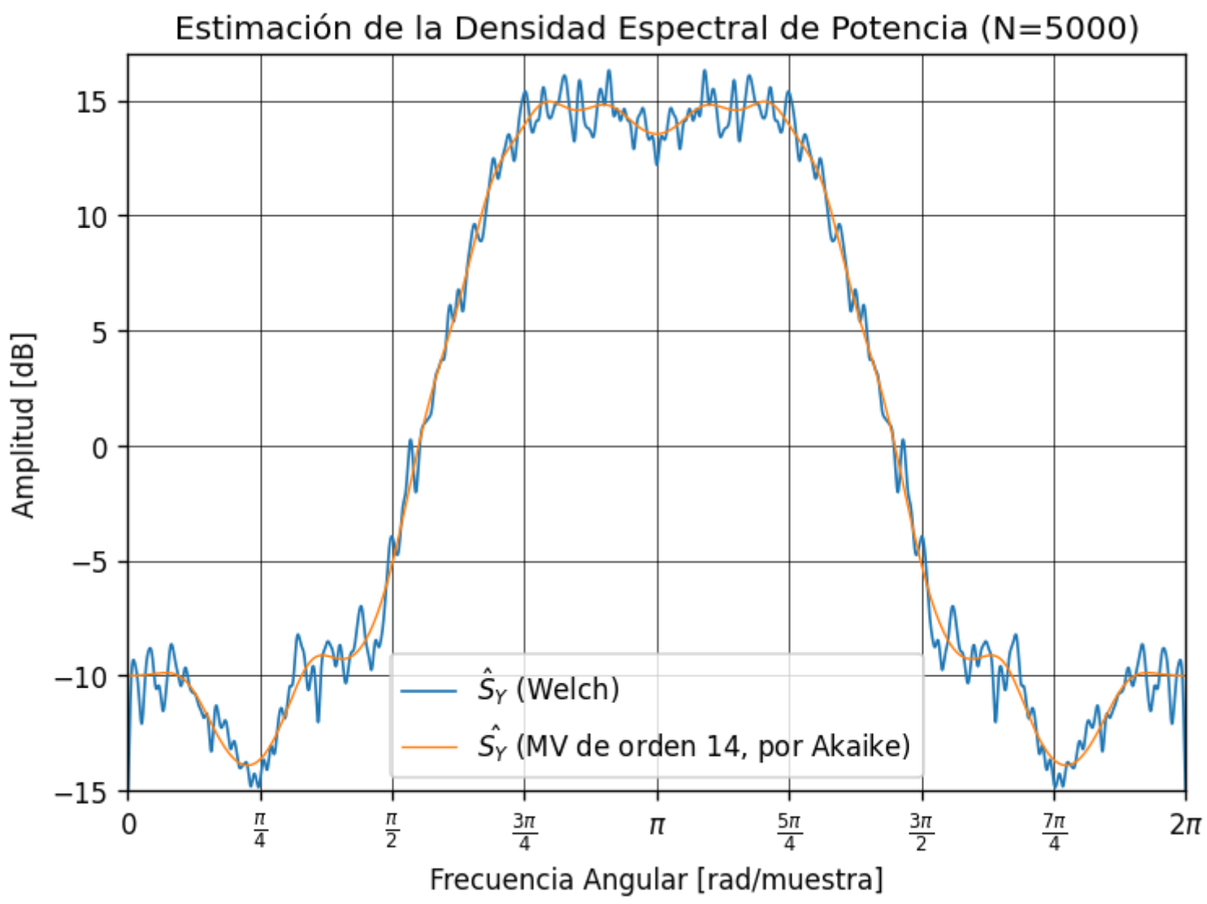
$\Delta(m)$	Soporte empírico para el modelo de orden m
$\Delta(1) = 2259.3156427216363$	Esencialmente ningún soporte empírico
$\Delta(2) = 499.5532537112049$	Esencialmente ningún soporte empírico
$\Delta(3) = 149.5058689697662$	Esencialmente ningún soporte empírico
$\Delta(4) = 135.18148091251533$	Esencialmente ningún soporte empírico
$\Delta(5) = 22.777945081058533$	Esencialmente ningún soporte empírico
$\Delta(6) = 13.458357252512542$	Esencialmente ningún soporte empírico
$\Delta(7) = 15.35261802128025$	Esencialmente ningún soporte empírico
$\Delta(8) = 17.503779677706007$	Esencialmente ningún soporte empírico
$\Delta(9) = 16.1447595521322$	Esencialmente ningún soporte empírico
$\Delta(10) = 13.101742480492248$	Esencialmente ningún soporte empírico
$\Delta(11) = 3.926587106666375$	Considerablemente menos que el elegido
$\Delta(12) = 1.3598029522327124$	Sustancial
$\Delta(13) = 0.592391492024035$	Sustancial
$\Delta(14) = 0.0$	Sustancial
$\Delta(15) = 1.7300035539847158$	Sustancial
$\Delta(16) = 3.9437936593139966$	Considerablemente menos que el elegido
$\Delta(17) = 5.775458654050453$	Considerablemente menos que el elegido
$\Delta(18) = 8.15071027443264$	Considerablemente menos que el elegido
$\Delta(19) = 9.01362271112248$	Considerablemente menos que el elegido
$\Delta(20) = 11.408412567614505$	Esencialmente ningún soporte empírico

2.5.2. Gráfico comparativo

Para aplicar Welch, se utilizaron segmentos de 250 muestras con solapamiento del 50%. El orden elegido para el estimador de máxima verosimilitud es de $m = 14$ por lo mostrado previamente. Se presenta a continuación el gráfico.

AFFICHER LE CODE

```
<matplotlib.legend.Legend at 0x7fa3c4e26d50>
```



Nuevamente, podemos observar que la varianza es mucho menor para el estimador de máxima verosimilitud, siendo este mejor que el estimador de Welch.