אלגוריתמים סטטיסטיים לעיבוד אותות – מטלת תוכנה
מגישים: יואב אלינסון - 206036949 וחן סרור - 203531645

שאלה 1- חישוב אנליטי: – חישוב הפרמטרים:

נתונה האנרגיה של האותות, משם נחלץ את שונות הרעש:

$1 = E\left[x_1^2[n]\right] = E[(w_1[n] - 3w_1[n-1] - 4w_1[n-2])^2] =$

$= E\left[w_1^2[n] - 6w_1[n]\cdot w_1[n-1] - 8w_1[n]\cdot w_1[n-2] + 9w_1^2[n-1] + 24w_1[n-1]\cdot w_1[n-2] + 16w_1^2[n-2]\right]$

$\underbrace{=}_{iid} E\left[w_1^2[n]\right] - E[6w_1[n]\,\cdot] - E\left[8w_1[n]\cdot w_1[n-2]\right] + E\left[9w_1^2[n-1]\right] + E\left[24w_1[n-1]\cdot w_1[n-2]\right] + E\left[16w_1^2[n-2]\right]$

$= \sigma_1^2 - \cancel{E\left[6w_1[n]\right]\cdot E\left[w_1[n-1]\right]} - \cancel{E\left[8w_1[n]\right]\cdot E\left[w_1[n-2]\right]} + 9\sigma_1^2 + \cancel{E\left[24w_1[n-1]\cdot w_1[n-2]\right]} + 16\sigma_1^2 = \sigma_1^2 + 9\sigma_1^2 + 16\sigma_1^2 \Rightarrow \sigma_1^2 + 9\sigma_1^2 + 16\sigma_1^2 = 1 \Rightarrow \sigma_1^2 = \frac{1}{26}$

$\sigma_1^2 = \frac{1}{26}$

$1 = \left[x_1^2[n]\right] = E[(0.7x_2[n-1] + w_2[n])(0.7x_2[n-1] + w_2[n])] =$

$= E\left[0.14x_2^2[n-1] + 1.4x_2[n-1]w[n] + w_2^2[n]\right]$

$= 0.14E\left[x_2^2[n-1]\right] + E\left[w_2^2[n]\right] = 0.14 + \sigma_2^2 \Rightarrow 0.14 + \sigma_2^2 = 1$
$\Rightarrow \sigma_2^2 = 0.86$

חישוב הספקטרום האמיתי של האותות:

$R_{X_1X_1}(l) = E\left[x_1[n]\cdot x_1[n]\right] =$

$= E[(w_1[n] - 3w_1[n-1] - 4w_1[n-2])\cdot(w_1[n-l] - 3w_1[n-l-1] - 4w_1[n-l-2])] =$

$= E\left[w_1[n]\cdot w_1[n-l] - 3w_1[n]\cdot w_1[n-l-1] - 4w_1[n]\cdot w_1[n-l-2] - 3w_1[n-1]\cdot w_1[n-l] + 9w_1[n-1]\cdot w_1[n-l-1] + 12w_1[n-1]\cdot w_1[n-l-2] - 4w_1[n-2]\cdot w_1[n-l] + 12w_1[n-2]\cdot w_1[n-l-1] + 16w_1[n-2]\cdot w_1[n-l-2]\right] =$

$= R_{x_1x_1}(l) - 3R_{x_1x_1}(l+1) - 4R_{x_1x_1}(l+2) - 3R_{x_1x_1}(l-1) + 9R_{x_1x_1}(l) + 12R_{x_1x_1}\cdot(l+1) - 4R_{x_1x_1}(l-2) + 12R_{x_1x_1}(l-1) + 16R_{x_1x_1}(l)$

$= R_{x_1x_1}(l) + 9R_{x_1x_1}(l) + 9R_{x_1x_1}(l+1) - 4R_{x_1x_1}(l+2) + 4R_{x_1x_1}(l-2) - 9R_{x_1x_1}(l-1) + 16R_{x_1x_1}(l)$

$$= f(x) = \begin{cases} 26\sigma_1^2, & l = 0 \\ 9\sigma_1^2, & l = \pm 1 \\ -4\sigma_1^2, & l = \pm 2 \end{cases} = \begin{cases} 1, & l = 0 \\ \frac{9}{26}, & l = \pm 1 \\ -\frac{2}{13}, & l = \pm 2 \end{cases}$$

$$R_{x_1x_1}(l) = \delta(l) + \frac{9}{26}\big(\delta(l+1) + \delta(l-1)\big) - \frac{2}{13}\big(\delta(l+2) + \delta(l-2)\big)$$

$$S_{x_1x_1}(Z) = 1 + \frac{9}{26}(Z^{-1} + Z^{1}) - \frac{2}{13}(Z^{-1} + Z^{1})$$

$$S_{x_1x_1}(e^{j\omega}) = 1 + \frac{9}{26}\left(e^{-j\omega} + e^{j\omega}\right) - \frac{2}{13}\left(e^{-j\omega} + e^{j\omega}\right) = 1 + \frac{9}{26}\cdot 2\cos(\omega) - \frac{2}{13}\cdot 2\cos(\omega)$$

$$S_{x_1x_1}(e^{j\omega}) = 1 + \frac{9}{26}\cdot 2\cos(\omega) - \frac{2}{13}\cdot 2\cos(\omega)$$

2. תהליך זה הינו $AR(1), \alpha = 0.7$ ולכן: $R_{x_2x_2}(l) = \frac{\sigma_2^2}{1-\alpha^2}\cdot \alpha^{|l|} = \frac{0.86^2}{1-0.7^2}\cdot 0.7^{|l|}$

אכן מתקיים עבור $l = 0$

$$R_{x_2x_2}(0) = \frac{\sigma_2^2}{1 - 0.14}$$

$$R_{x_2x_2}(l) = \frac{(0.7)^{|l|}\cdot \sigma_2^2}{(1 - 0.7^2)}$$

$$R_{x_2x_2}(l) = (0.7)\hat{}|l|$$

$$S_{x_2x_2}(e^{j\omega}) = \frac{\sigma_2^2}{|1 - \alpha e^{-j\omega}|^2}\frac{\sigma_2^2}{|1 - 0.7e^{-j\omega}|^2} = \frac{0.86}{|1 - 0.7e^{-j\omega}|^2}$$

$$S_{x_2x_2}(e^{j\omega}) = \frac{0.86}{|1 - 0.7e^{-j\omega}|^2}$$

שאלה 2- הגרלת האותות:

א. הגרלנו את $x_1[n], x_2[n]$ על ידי הפונקציה הבאה:

```python
def gen_signals(sigma_w1=np.sqrt(1/26), n1=1024, sigma_w2=np.sqrt(0.86), n2=2048):
    w1 = np.random.normal(0, sigma_w1, n1)
    x1 = np.array([w1[n]-3*w1[n-1]-4*w1[n-2] for n in range(n1)])

    w2 = np.random.normal(0, sigma_w2, n2)

    def gen_x2(x2, n):
        return 0.7*x2[n-1] + w2[n]
    x2_tmp = np.zeros(n2)
    for n in range(1, n2):
        x2_tmp[n] = gen_x2(x2_tmp, n)

    x2 = x2_tmp[-1024:]
    return x1, x2
```

1. למעשה – הגרלנו את $w_1[n], w_2[n]$ ואז דגמנו מהם את הסיגנלים לפי משוואת ההפרש.
   את $x_1[n]$ אין בעיה לדגום משום שאינו משתמש בערכי עבר של עצמו – אלא רק בערכים של $w_1[n]$. ואילו עבור $x_2[n]$ ישנו שימוש (ברקורסיה) בכל ערכי העבר של האות, לכן נאלצנו לייצר מ2048 דגימות תחילה ואז לקחת את ה1024 האחרונות. הסיבה לכך הינה שתנאי ההתחלה משפעים מאוד על הערכים הבאים, ועל מנת להימנע מהשפעה של תנאי התחלה לא מתאימים לאורך כל האות, תחילה נתנו לאות להתרגל לתנאי ההתחלה (1024 דגימות ראשונות) ואז על בסיס דגימות אלו יצרנו את $x_2[n]$.

2. על מנת ליצור את $x_2[n]$ באופן ישיר (ללא מספר כפול של דגימות תחילה), נצטרך להשתמש בתנאי התחלה מתאימים עבור $x_2[-1]$ בלבד. כדי להמנע מהגרלה נוספת פשוט ניקח את הערך של $x_2[1023]$ שנוצר בהגרלה של 2048 הדגימות. במקרה שלנו קיבלנו
   $x_2[1023] = -0.5880857651749811$

לכן הפונקציה תראה כך:

```python
def gen_signals(sigma_w1=np.sqrt(1/26), n1=1024, sigma_w2=np.sqrt(0.86), n2=1024):
    w1 = np.random.normal(0, sigma_w1, n1)
    x1 = np.array([w1[n]-3*w1[n-1]-4*w1[n-2] for n in range(n1)])

    w2 = np.random.normal(0, sigma_w2, n2)

    def gen_x2(x2, n):
        return 0.7*x2[n-1] + w2[n]
    x2_tmp = np.zeros(n2)
    x2_tmp[-1] = -0.5880857651749811
    for n in range(1, n2):
        x2_tmp[n] = gen_x2(x2_tmp, n)

    return x1, x2_tmp
```

ב. **פריודוגרמה:**
את חישוב הפריודוגרמה ביצענו על ידי הפונקציה הבאה:

```python
def calc_periodegram(x, M=4096):
    N = len(x)
    X = np.fft.fft(x, n=M)
    Sx_per = (1/N)*(np.abs(X)**2)[:int(M/2)+1]
    w_M = np.linspace(0, 2*np.pi, M)[:int(M/2)+1]
    return Sx_per, w_M
```

מימוש לפי הגדרה – כאשר numpy.fft.fft יודע לרפד באפסים על מנת לחשב dft ברזולוציה הנדרשת.

ג. **קורלוגרמה:**
את חישוב הקורלוגרמה ביצענו על ידי הפונקציה הבאה:

```python
def calc_correlogram(x, M=4096):
    N = len(x)
    x = np.pad(x, (0, int(M/2)+1-x.shape[0]), 'constant')
    w_M = np.linspace(0, 2*np.pi, M)[:int(M/2)+1]
    Rx_hat = np.correlate(x, x, 'full')/N
    Sx_cor = np.dot(np.exp(-1j*WL), Rx_hat).real[-2049:]
    return Sx_cor, w_M
```

את משערך האוטו קורלציה של האות חישבנו בעזרת הפונקציה np.correlate אשר מקבילה לפונקציה xcorr במטלב.
האינדקס בו קיבלנו את $R[0]$ הינו $l = 2048$.

כאשר WL הינה המטריצה הבאה:

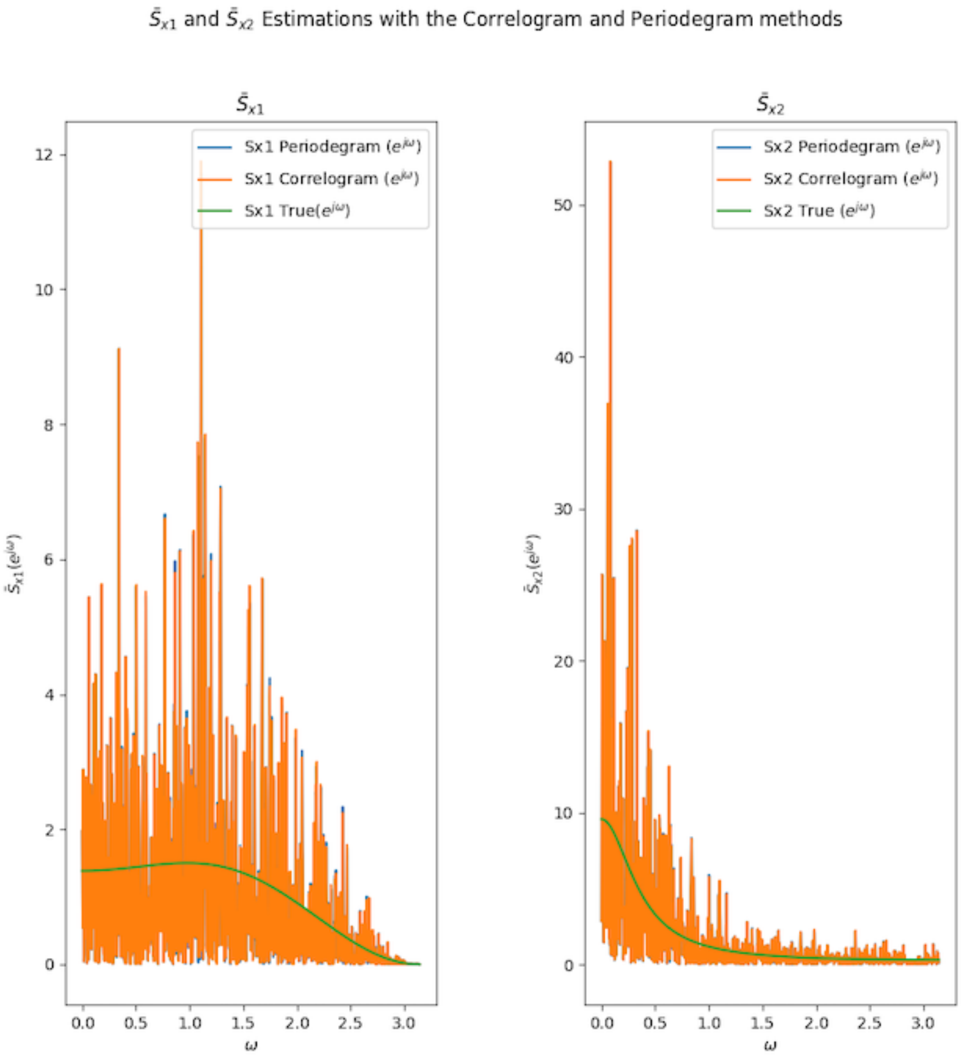
$$\begin{pmatrix} \omega_{-(N-1)} \cdot l_{-(N-1)} & \cdots & \omega_{-(N-1)} \cdot l_{(N-1)} \\ \vdots & \ddots & \vdots \\ \omega_{(N-1)} \cdot l_{-(N-1)} & \cdots & \omega_{(N-1)} \cdot l_{(N-1)} \end{pmatrix}$$

כלומר כל עמודה של המטריצה כוללת את כל התדרים בתחום $[-\pi, \pi]$ בקפיצות הדרושות (2049 תדרים), וכמו כן כל עמודה מוכפלת ב$l \in [-(N-1), N-1]$.
פונקציה ליצירת המטריצה:

```python
def create_wl_mat(M=4096):
    w_M = np.linspace(0, 2*np.pi, M)[:int(M/2)+1]
    w = np.concatenate((-w_M[::-1], w_M[1:]))
    w_mat = np.repeat(w.reshape(w.shape[0], 1), w.shape[0], axis=1)
    l = np.zeros(w_mat.shape)
    l[0, :] = np.arange(-int(M/2), int(M/2)+1, 1)
    return np.dot(w_mat, l)
```

בעזרת כפל מטריצות הצלחנו להאיץ את הפעולה בכ90% לעומת ריצה בלולאה.

ד. שרטוטים:



ניתן לראות שאכן השיטות זהות (עד כדי טעות נומרית זעירה) עבור שני האותות!

שאלה 3 – השוואת משערכים:

נדרשנו לממש את המשערכים הבאים:

א. שיטת $\boldsymbol{Bartlet}$:

בשיטה זו מחלקים את האות למקטעים באורך L, ומחשבים לכל מקטע את משערך הפריודוגרמה שלו. לאחר מכן ממצעים את התוצאות לקבלת משערך לכל האות.

הפונקציה שמימשנו:

```python
def calc_bartlet(x, k, M=4096):
    N = len(x)
    if(not N % k):
        L = int(N/k)
        x_frames = [np.pad(x[(i*L):(i*L)+L], (0, M-L), 'constant')
                    for i in range(k)]
        Sx_k = np.array(
            [(np.abs(np.fft.fft(x_frames[i]))**2)/L for i in range(k)])
        Sx_b = np.array(1/k)*(np.sum(Sx_k, axis=0))[:int(M/2)+1]
        w_M = np.linspace(0, 2*np.pi, M)[:int(M/2)+1]
        return Sx_b, w_M
    else:
        print('not a valid value for k')
        return None, None
```

ב. שיטת $\boldsymbol{Welch}$:

שיטה זו דומה לשיטת $Bartlet$ אך כאן החלוקה היא למקטעים עם חפיפה באורך D.

הפונקציה שמימשנו:

```python
def calc_welch(x, k, D, L, M=4096):
    N = len(x)
    x_frames = np.array([np.pad(x[i:i+L], (0, M-L), 'constant')
                        for i in np.multiply(list(range(k)), L-D)])
    Sx_k = np.array([(np.abs(np.fft.fft(x_frames[i]))**2)/L for i in range(k)])
    Sx_b = np.array((1/k)*(np.sum(Sx_k, axis=0)))[:int(M/2)+1]
    w_M = np.linspace(0, 2*np.pi, M)[:int(M/2)+1]
    return Sx_b, w_M
```

בשתי השיטות האלו בחרנו לרפד באפסים בעצמינו ולא אוטומטית על ידי numpy, רק על מנת להראות שהשיטות שקולות.

ג. שיטת $\boldsymbol{Blackman-Tukey}$:

שיטה זו מניחה שסדרת הקורלציה האמיתי של האות מתאפסת בשלב מסויים לכן מכפילה את משערך הקורלציה המוטה בחלון (השתמשנו בחלון ריבועי במקרה זה) ולאחר מכן ממשיכה בחישוב הספקטרום לפי הקורלוגרמה.

הפונקציה שמימשנו:

```python
def calc_blackman_tukey(x, L=1, M=4096):
    N = len(x)
    x = np.pad(x, (0, int(M/2)+1-x.shape[0]), 'constant')
    w_M = np.linspace(0, 2*np.pi, M)[:int(M/2)+1]

    Rx_hat = np.correlate(x, x, 'full')/N
    l0 = np.argmax(Rx_hat)
    win = np.zeros(Rx_hat.shape)
    win[l0-L+1:l0+L] = np.ones((2*L-1))
    Rx_hat_win = np.multiply(Rx_hat, win)
    Sx_cor = np.dot(np.exp(-1j*WL), Rx_hat_win).real[-2049:]
    return Sx_cor, w_M
```

כמובן שאם נבצע שערוך אחד של הספקטרום על האות המוגרל נקבל תוצאה רחוקה (מאוד) מהספקטרום האמיתי של האות, לכן על מנת לבחון את יכולות המשערכים שלנו השתמשנו בניסוי מונטה קרלו על 100 הגרלות.
ונדרשנו להציג את הפרמטרים הבאים:

ממוצע המשערך	$\bar{S}(e^{j\omega}) = 1/M_C \cdot \sum_{m=1}^{M_C} \hat{S}^{(m)}(e^{j\omega})$
הטיה	$B(e^{j\omega}) = \bar{S}(e^{j\omega}) - S(e^{j\omega})$
שונות המשערך	$V(e^{j\omega}) = 1/M_C \cdot \sum_{m=1}^{M_C} \left\|\hat{S}^{(m)}(e^{j\omega}) - \bar{S}(e^{j\omega})\right\|^2$
שגיאה ריבועית ממוצעת	$MSE(e^{j\omega}) = V(e^{j\omega}) + B^2(e^{j\omega})$

את הפעולה הזו ביצענו בעזרת לולאה שרצה לפי מספר הניסויים, כאשר בכל פעם מגרילה את האותות $x_1[n], x_2[n]$ ואז מחשבת את כל המשערכים לפי השיטות שמימשנו.
לאחר מכן (בסוף הלולאה) כל הפרמטרים מחושבים עבור כל שיטה.
הפונקציות שמימשנו:

```
def get_Sx_bar(Sx, M=4096):
    Mc = len(Sx)
    Sx = np.array(Sx).reshape(Mc, int(M/2)+1)
    Sx_mean = np.sum(Sx, axis=0)/Mc
    return Sx_mean.flatten()


def get_bias(Sx_bar, Sx_true):
    if np.array(Sx_true).shape == np.array(Sx_bar).shape:
        return Sx_bar - Sx_true
    else:
        return None


def get_var(Sx, Sx_bar, M=4096):
    Mc = len(Sx)
    Sx = np.array(Sx).reshape(Mc, int(M/2)+1)
    var = np.zeros(Sx_bar.shape)
    for m in range(Mc):
        var += np.abs(Sx[m] - Sx_bar)**2
    return (var/Mc).flatten()


def get_MSE(var, bias):
    return var + bias**2
```

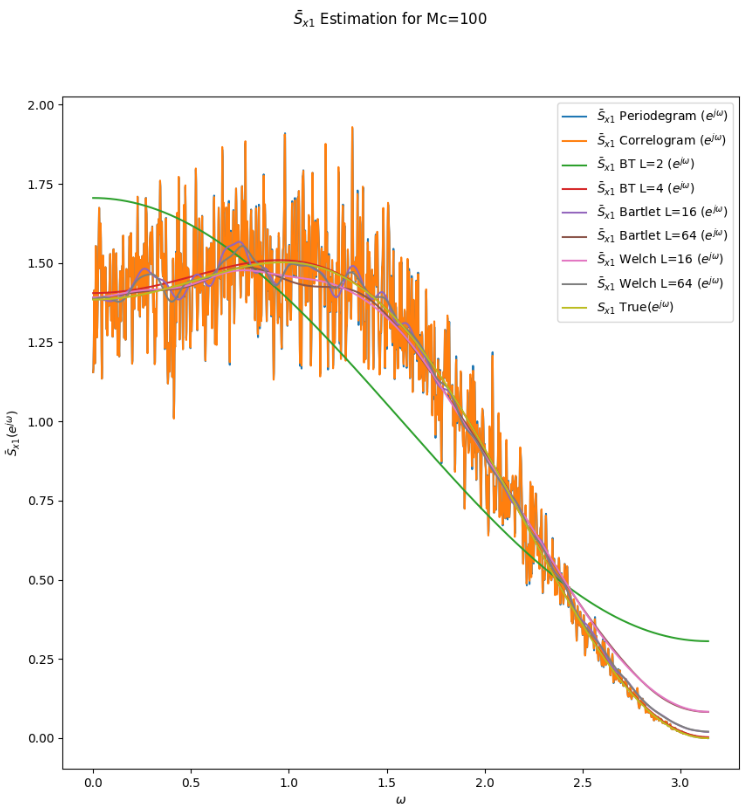
את שלושת הפונקציות האלו שמנו בפונקציה אחת שקוראת לכולן יחדיו (גם לפונקציות של סעיף 4) ומחזירה את כל הנתונים.
הפונקציה:

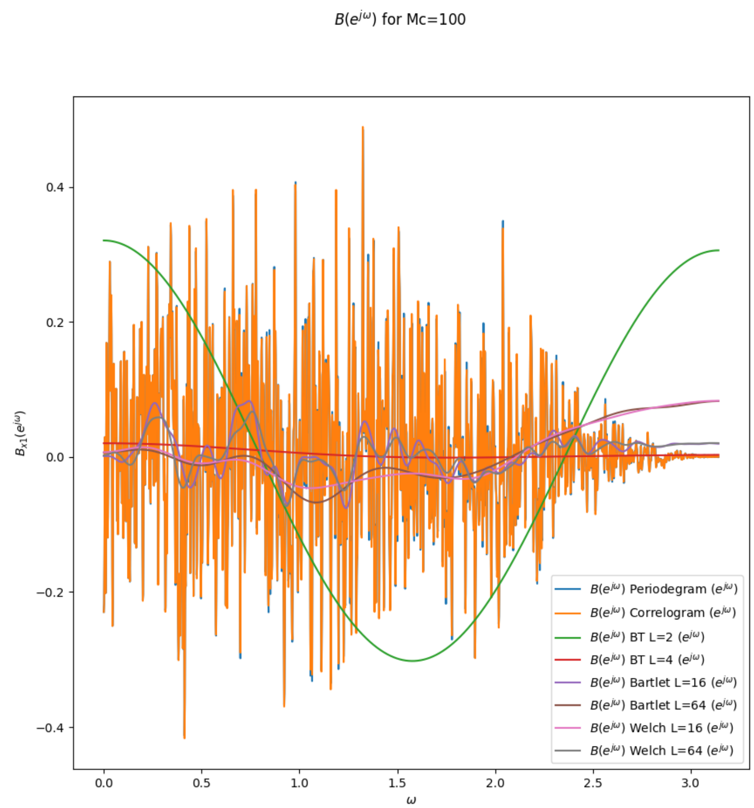
```
def get_all_stats(Sx, Sx_true, M=4096):
    Sx_bar = get_Sx_bar(Sx)
    B = get_bias(Sx_bar, Sx_true)
    var = get_var(Sx, Sx_bar)
    mse = get_MSE(var, B)
    bias_total = total_bias(B)
    mse_total = total_mse(mse)
    var_total = total_var(var)
    return Sx_bar, B, var, mse, bias_total, var_total, mse_total
```

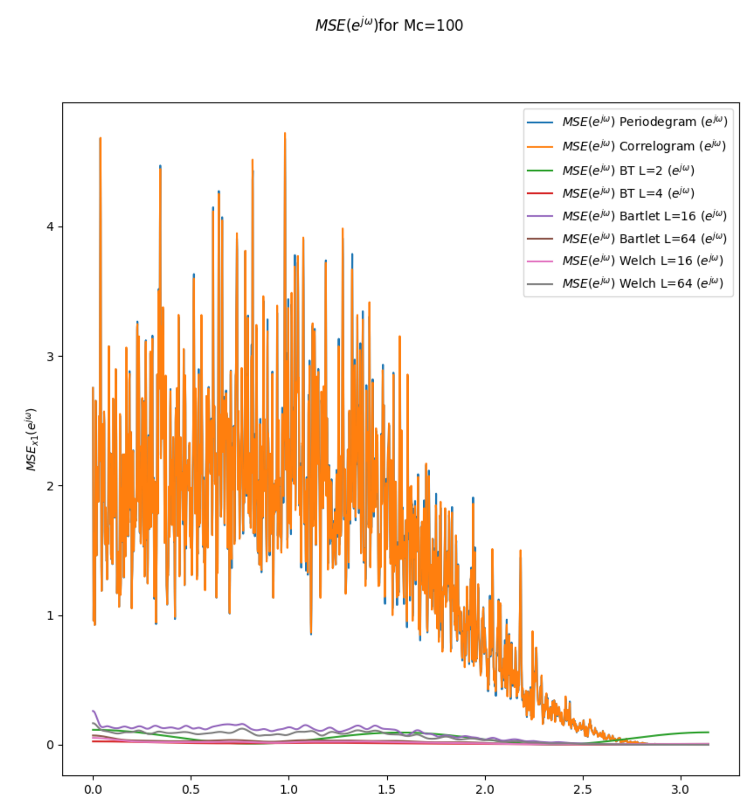
על שלושת השורות האחרונות של הפונקציה נסביר בהמשך.

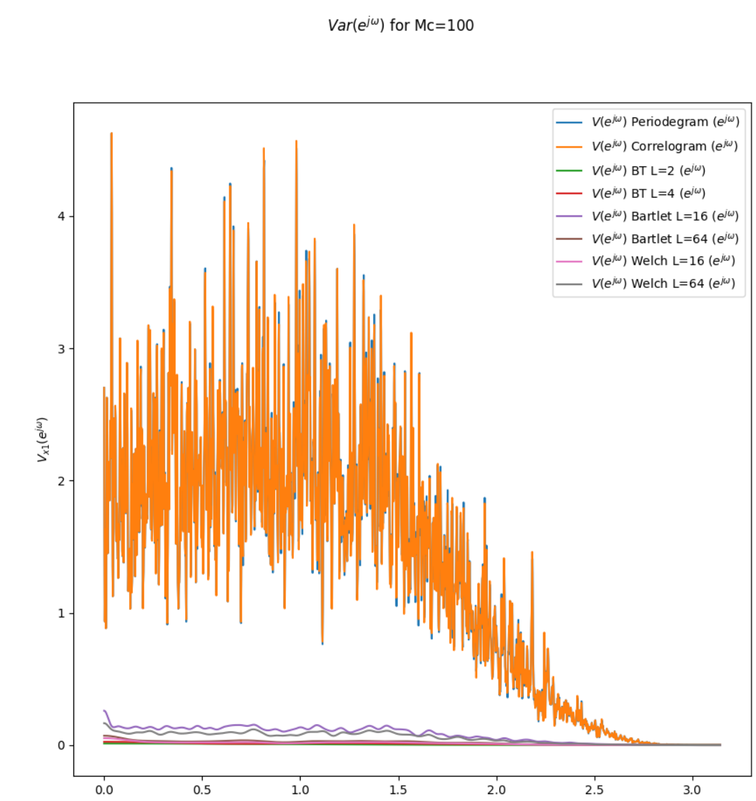
להלן כל הגרפים משאלה זו:

$$x_{1[n]}$$

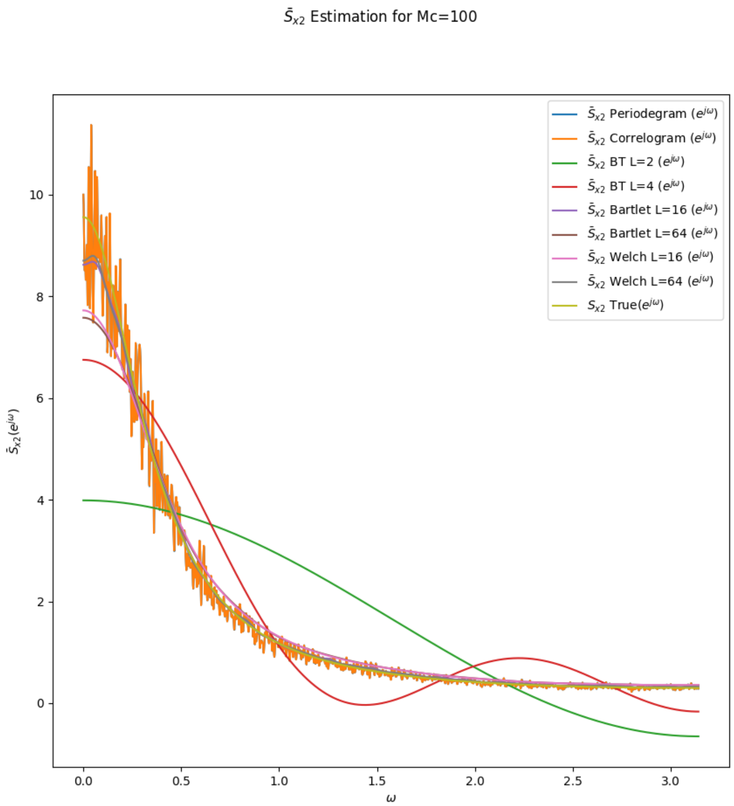


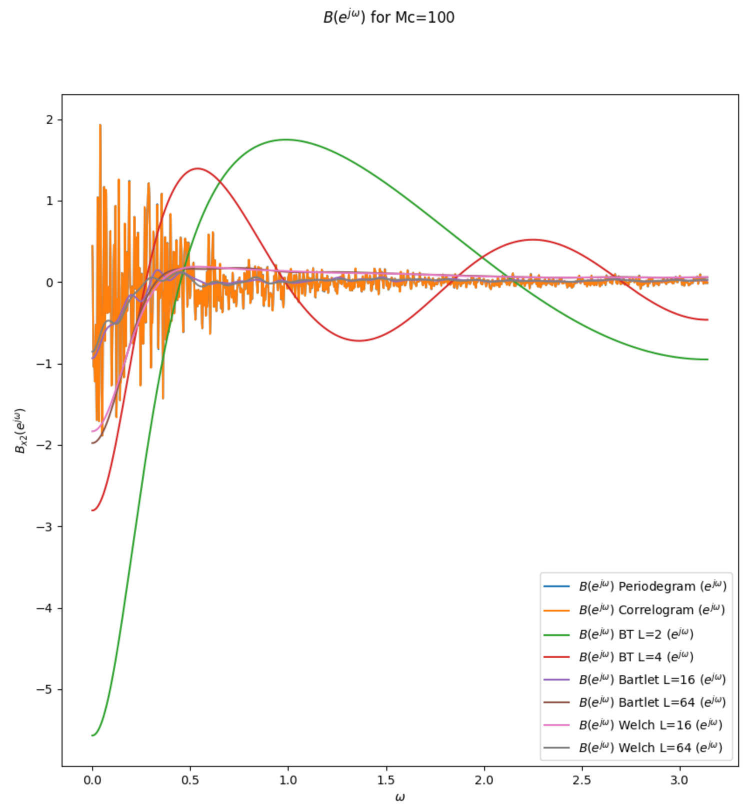


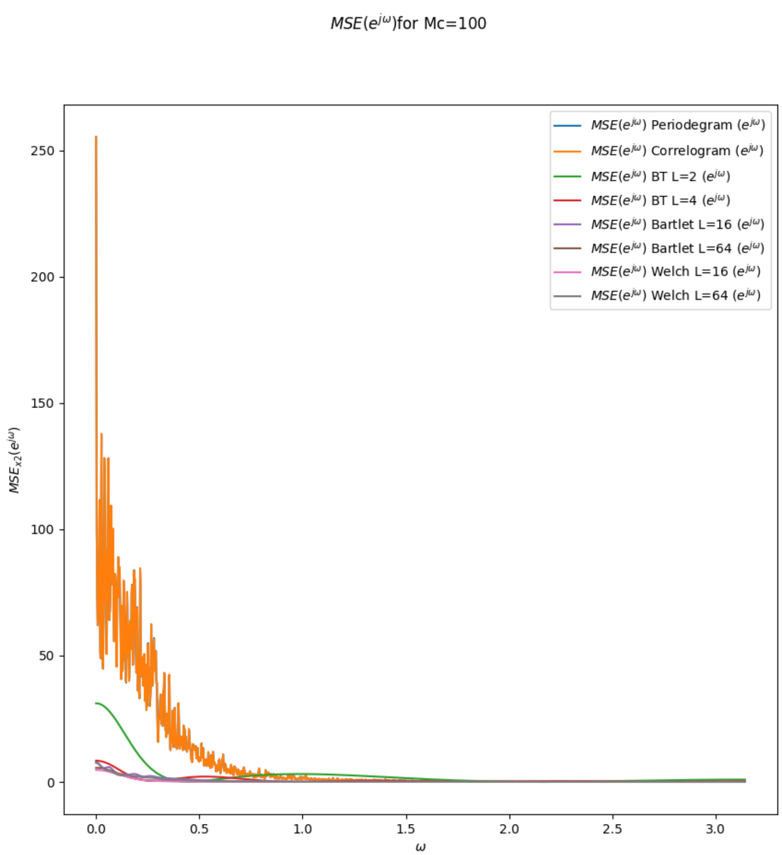


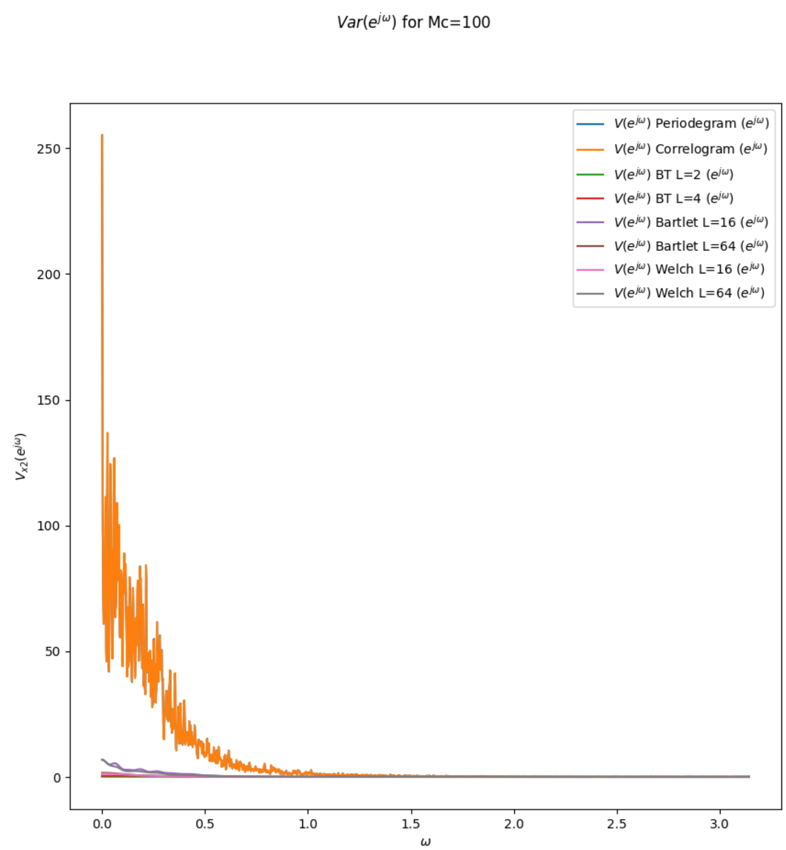


$x_{2[n]}$


$\bar{S}_{x2}$ Estimation for Mc=100
$\bar{S}_{x2}$ Periodegram $(e^{j\omega})$
$\bar{S}_{x2}$ Correlogram $(e^{j\omega})$
$\bar{S}_{x2}$ BT L=2 $(e^{j\omega})$
$\bar{S}_{x2}$ BT L=4 $(e^{j\omega})$
$\bar{S}_{x2}$ Bartlet L=16 $(e^{j\omega})$
$\bar{S}_{x2}$ Bartlet L=64 $(e^{j\omega})$
$\bar{S}_{x2}$ Welch L=16 $(e^{j\omega})$
$\bar{S}_{x2}$ Welch L=64 $(e^{j\omega})$
S_{x2} True$(e^{j\omega})$
$\bar{S}_{x2}(e^{j\omega})$
ω


$B(e^{j\omega})$ for Mc=100
$B(e^{j\omega})$ Periodegram $(e^{j\omega})$
$B(e^{j\omega})$ Correlogram $(e^{j\omega})$
$B(e^{j\omega})$ BT L=2 $(e^{j\omega})$
$B(e^{j\omega})$ BT L=4 $(e^{j\omega})$
$B(e^{j\omega})$ Bartlet L=16 $(e^{j\omega})$
$B(e^{j\omega})$ Bartlet L=64 $(e^{j\omega})$
$B(e^{j\omega})$ Welch L=16 $(e^{j\omega})$
$B(e^{j\omega})$ Welch L=64 $(e^{j\omega})$
$B_{x2}(e^{j\omega})$
ω


$MSE(e^{j\omega})$for Mc=100
$MSE(e^{j\omega})$ Periodegram $(e^{j\omega})$
$MSE(e^{j\omega})$ Correlogram $(e^{j\omega})$
$MSE(e^{j\omega})$ BT L=2 $(e^{j\omega})$
$MSE(e^{j\omega})$ BT L=4 $(e^{j\omega})$
$MSE(e^{j\omega})$ Bartlet L=16 $(e^{j\omega})$
$MSE(e^{j\omega})$ Bartlet L=64 $(e^{j\omega})$
$MSE(e^{j\omega})$ Welch L=16 $(e^{j\omega})$
$MSE(e^{j\omega})$ Welch L=64 $(e^{j\omega})$
$MSE_{x2}(e^{j\omega})$
ω


$Var(e^{j\omega})$ for Mc=100
$V(e^{j\omega})$ Periodegram $(e^{j\omega})$
$V(e^{j\omega})$ Correlogram $(e^{j\omega})$
$V(e^{j\omega})$ BT L=2 $(e^{j\omega})$
$V(e^{j\omega})$ BT L=4 $(e^{j\omega})$
$V(e^{j\omega})$ Bartlet L=16 $(e^{j\omega})$
$V(e^{j\omega})$ Bartlet L=64 $(e^{j\omega})$
$V(e^{j\omega})$ Welch L=16 $(e^{j\omega})$
$V(e^{j\omega})$ Welch L=64 $(e^{j\omega})$
$V_{x2}(e^{j\omega})$
ω

שאלה 4 – דירוג ביצועים:

בשאלה זו נדרשנו לחשב פרמטרים המעידים על טיב השערוך:

הטיה ריבועית ממוצעת כוללת	$\langle B^2 \rangle = \frac{1}{2\pi}\int_{-\pi}^{\pi} B^2(\omega)d\omega \approx \frac{1}{M}\sum_{m=0}^{M-1} B^2(2\pi jm/M)$
שונות ממוצעת כוללת	$\langle V \rangle = \frac{1}{2\pi}\int_{-\pi}^{\pi} V(\omega)d\omega \approx \frac{1}{M}\sum_{m=0}^{M-1} V(2\pi jm/M)$
שגיאה ריבועית ממוצעת כוללת	$\langle MSE \rangle = \frac{1}{2\pi}\int_{-\pi}^{\pi} MSE(\omega)d\omega \approx \frac{1}{M}\sum_{m=0}^{M-1} MSE(2\pi jm/M)$

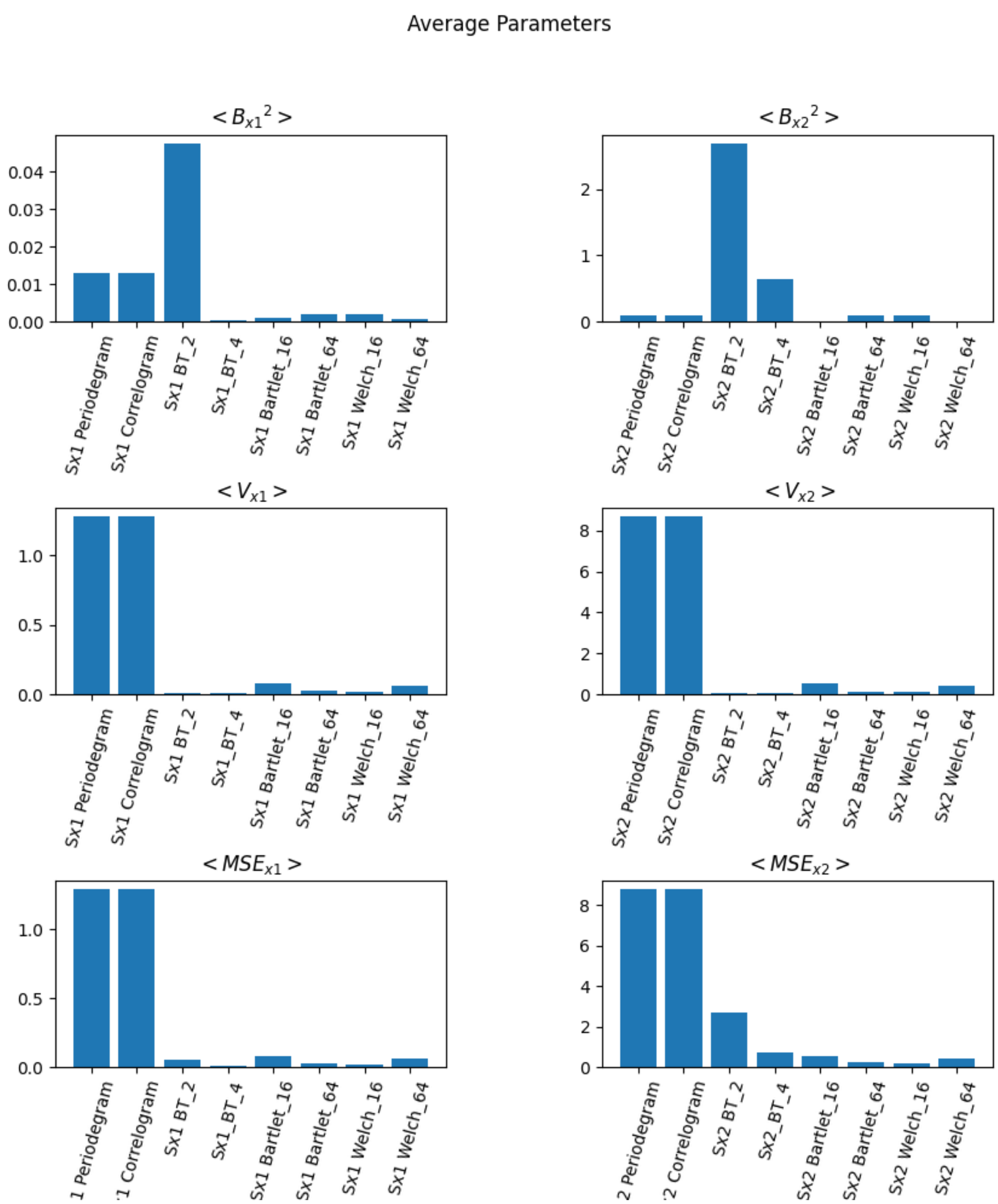
מימשנו את שלושתם בפונקציות הבאות (וקראנו להם בסוף ניסוי הMC):

```
def total_bias(B):
    return np.sum(B**2)/len(B)


def total_var(var):
    return np.sum(var)/len(var)


def total_mse(mse):
    return np.sum(mse)/len(mse)
```

את כל הנתונים הכנסנו לגרף על מנת להשוות בין המשערכים:



א. ניתן לראות שהמשערכים שלא מסיקים דבר על האות יש שונות גדולה ולכן גם השגיאה הריבועית הממוצעת גבוהה. שאר המשערכים בעלי שונות נמוכה יחסית ולכן גם עדיפים על משערכי פריודגרמה וקורלוגרמה.

ב. מהתוצאות ניתן לראות כי עבור משערכי הקורלוגרמה והפריודגרמה השונות גדולה (מאוד) ואילו עבוד שאר השיטות היא קטנה משמעותית – מתאים לערכים שקיבלנו בכיתה!

ועבור ההטיה, קיימת הטיה (אמנם קטנה) בכל המשערכים (משום שN אינו שואף לאינסוף).