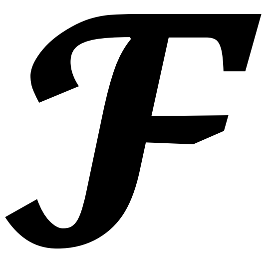
אותות ומערכות – עבודה 1 להגשה

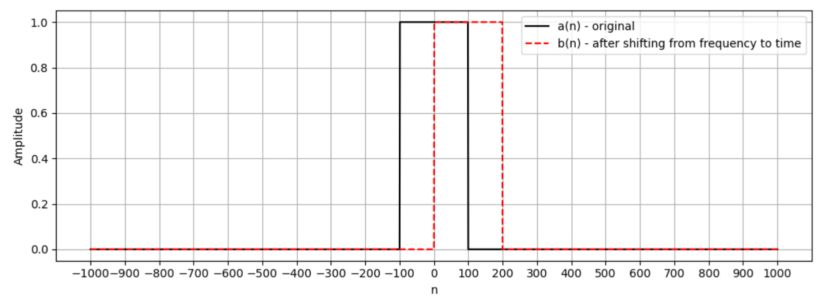
מגישה: רותם סילם – קיבלתי אישור מאופק להגיש באיחור קל

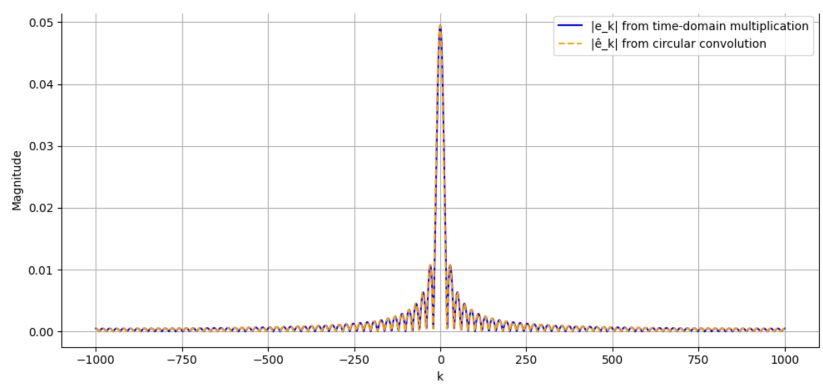
ת.ז: 206663437

מרצה: ד"ר עופר שוורץ

<u>תוכן עניינים:</u>







סעיף א':

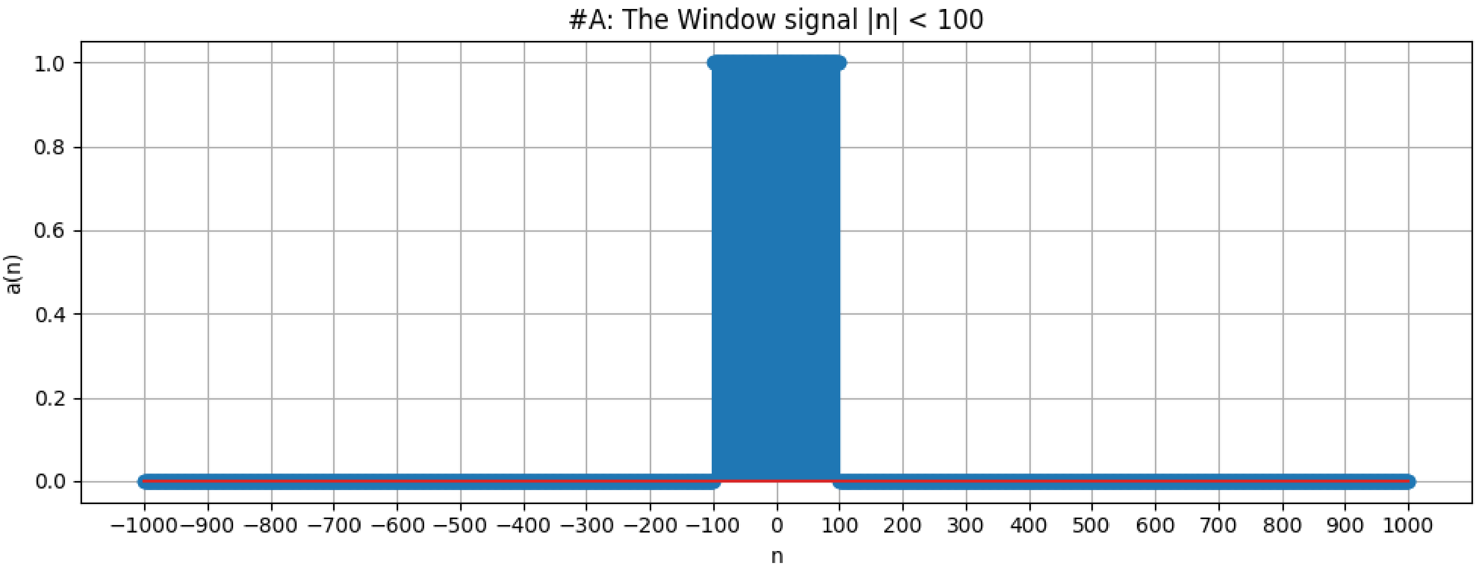
- הגדרת אות החלון, והדפסת גרף שלו.

```python
import numpy as np
import matplotlib.pyplot as plt
import cmath
import math

# Part A:
# Time vector - Total 2001 points
n = np.arange(-1000, 1001)

# Generate the signal according to the condition |n| < 100
a = np.where(np.abs(n) < 100, 1, 0)

# Window signal graph
plt.figure(figsize=(10, 4))
plt.stem(n, a, use_line_collection=True)
plt.title("#A: The Window signal |n| < 100")
plt.xlabel("n")
plt.xticks(np.arange(min(n), max(n)+1, 100))
plt.ylabel("a(n)")
plt.grid(True)
plt.tight_layout()
plt.show()
```

n - וקטור זמן של מספרים שלמים בין -1000 ל-1000 קפיצות של 1 (סה"כ 2001 ערכים)

a - פונקציית חלון: מחזירה 1 אם |n|<100, אחרת 0. מקבלים אות חלון עם 1 ב-199 המקומות שבין -99 ל-99

הדפסת הגרף – מימדי הגרף, ערכי x וy, כותרת הגרף, שם ציר x, שנתות ציר x, כותרת ציר y, קווי אורך ורוחב, סידור שהכל יסתדר יפה בתצוגה, הדפסה של הגרף.



סעיף ב':

- הצגה של מקדמי פורייה של האות (אמפליטודה ופאזה)
- הצגה שהאות ממשי וסימטרי

```python
# Part B:
N = 2001  # Cycle length
k_vals = np.arange(-1000, 1001)
a_k = []

for k in k_vals:
    sum_val = 0
    for idx, n_val in enumerate(n):
        angle = -2 * np.pi * k * n_val / N
        sum_val += a[idx] * np.exp(1j * angle)
    a_k.append(sum_val / N)
a_k = np.array(a_k)  # Convert to numpy array after filling

# Test 1: Are all a_k real?
imag_part = np.max(np.abs(np.imag(a_k)))
if imag_part < 1e-12:
    print("#B: Fourier coefficients are real (imag -> 0)")
else:
    print("#B: There are imaginary parts and therefore not real")

# Define threshold for negligible imaginary parts - define the imaginary part as zero if it is negligible
threshold = 1e-12
a_k_real = np.array([complex(x.real, 0) if abs(x.imag) < threshold else x for x in a_k])
```

```python
# Real part plot
plt.figure(figsize=(10, 4))
plt.plot(k_vals, a_k_real.real, label='a(k) - original', color='black')
plt.title("#B: real part of a_k")
plt.xlabel("k")
plt.xticks(np.arange(min(k_vals), max(k_vals)+1, 100))
plt.ylabel("a_k real part")
plt.grid(True)
plt.tight_layout()
plt.show()

#image part of a_k:
plt.figure(figsize=(10, 4))
plt.plot(k_vals, a_k_real.imag, label='a(k)-image part', color='black')
plt.title("#B: image part of a_k")
plt.xlabel("k")
plt.xticks(np.arange(min(k_vals), max(k_vals)+1, 100))
plt.ylabel("a_k Image part")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```python
# Phase graph of the coefficients a_k
plt.figure(figsize=(10, 4))
plt.stem(k_vals, np.angle(a_k_real, deg=True), use_line_collection=True)
plt.title("#B: Phase of Fourier coefficients ∠a_k of a(n) (in degrees)")
plt.xlabel("k")
plt.ylabel("angle(a_k) [degrees]")
plt.yticks([-180, -90, 0, 90, 180])
plt.grid(True)
plt.tight_layout()
plt.show()

# frequency symmetry: Does a_k = a_{-k} hold?
symmetric = True
N_k = len(a_k_real)
for i in range(1, N_k // 2):
    idx_pos = N_k // 2 + i
    idx_neg = N_k // 2 - i
    diff = np.abs(a_k_real[idx_pos] - a_k_real[idx_neg])
    if diff > 1e-12:
        symmetric = False
if symmetric:
    print("#B: a_k = a_{-k} for all k => symetric")
```

- **<u>k_vals</u> -** מאפשר לקבל את ערכי האינדקס k של מקדמי סדרת פורייה בטווח [-1000,1000] באופן סימטרי סביב 0.
- **a_k -** מערך ריק שניתן יהיה למלא אותו במקדמים מרוכבים של סדרת פורייה שנמצא.
- **<u>לולאת for:</u>**

הלולאה מחשבת את הערך של כל מקדם פורייה לפי הנוסחה הנתונה, בהתאם לתדירות שלו, ועוברת על כולם באמצעות האינדקס שעוקב באיזה מיקום נמצאים במערך. $ak = \frac{1}{N}\sum_n a(n)\cdot e^{-j2\pi kn/N}$

k – עובר על כל ערך בתחום [-1000, 1000] (התדר בו אנחנו נמצאים)

n_val – גם רץ על כל ערך בתחום [-1000, 1000]

idx - הוא פשוט המספר הסידורי של n במערך (לדוגמה עבור n=-1000, נשים במערך במקום ה index=0)

נחשב זווית, נחשב את המכפלה $a(n)\cdot e^{-j2\pi kn/N}$, מוסיפים לסכום הטור בהתאם לטור פורייה (sum_val), מוסיפים את הערך שקיבלנו (מנורמל) למיקום k המתאים במערך a_k. נשנה את מערך a_k לnumpy כדי שנוכל לעשות גרפים מנתוניו.

<u># Test 1: Are all a_k real?:</u>

לוקח את האיבר עם החלק המדומה הכי גדול (בערך מוחלט), ובודק אם הוא זניח.

ההדפסה שקיבלתי – מראה שכל המקדמים ממשיים:

```
#B: Fourier coefficients are real (imag -> 0)
```

 # treshold:

בלולאה זו, במידה והחלק המדומה זניח (קטן מtreshold שבחרתי שהוא קטן מאוד), נאפס לגמרי את החלק המדומה של אותו מקדם. עובר על כל המקדמים ושומר את ערכם המעודכן במערך חדש a_k_real.

 # real part plot: גרף שמראה את החלק הממשי של מקדמי הפורייה

 # imag part plot: גרף שמראה שהחלק המדומה של כל מקדמי פורייה הוא 0.

 # Phase graph of the coefficients a_k: מראה שהפאזה של כל המקדמים היא או 0 מעלות (ממשי חיובי) או 180 מעלות (ממשי שלילי).

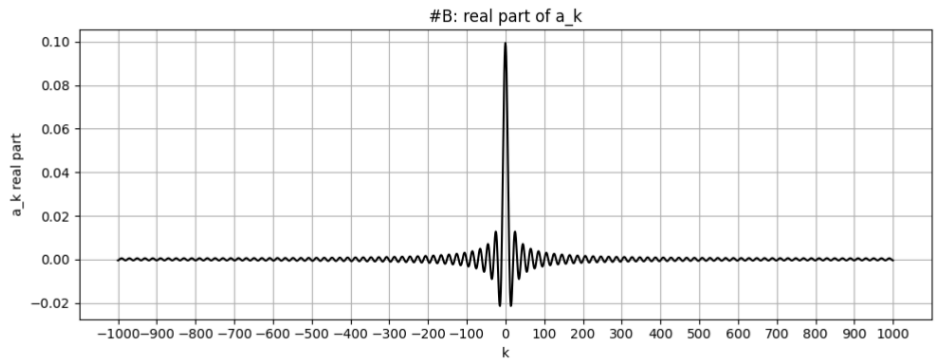
 # frequency symmetry: Does a_k = a_{-k} hold?: הלולאה בודקת האם מקדמי פורייה ak הם סימטריים סביב k=0 (חלוקת אורך המערך N ב2 ואז מעבר אינדקסים אחד ימינה ואחד שמאלה כל פעם).

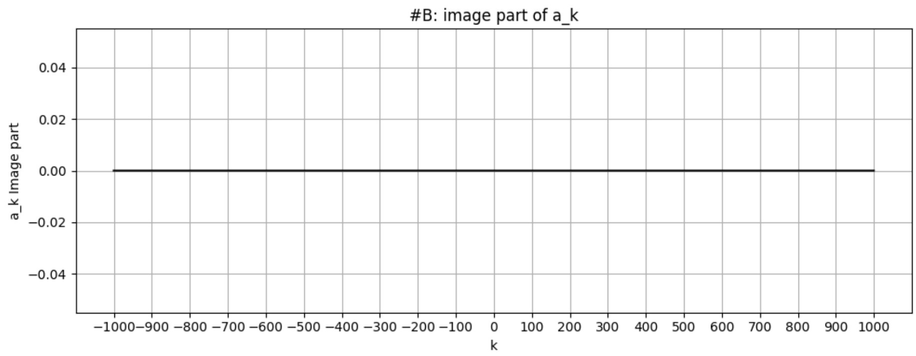
בבדיקת הסימטריה נבדוק האם ak=a−k מתקיים עבור כולם $k\neq0$ (מעבר לסף שגיאה זניח).

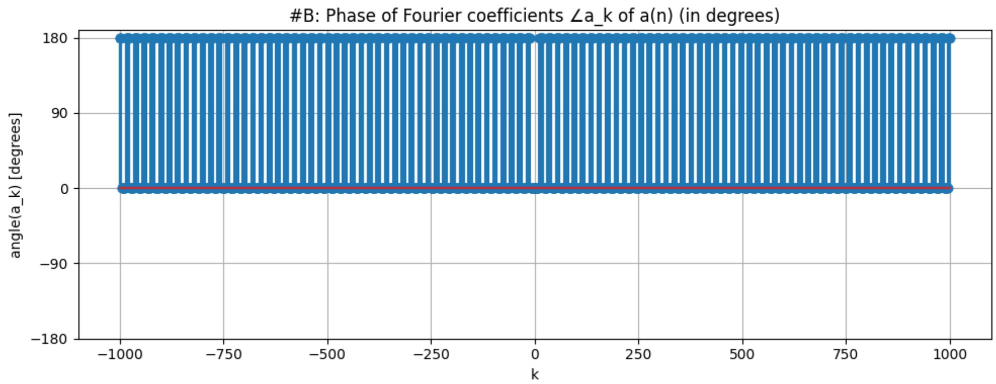
בודק שוויון בעזרת diff בו מחסרים בינהם. לאחר מכן נבדוק אם diff זניח, ואם לא אז נעדכן שלא סימטרי.

ההדפסה שקיבלתי – כלומר סימטרי:

```
#B: a_k = a_{-k} for all k => symetric
```







סעיף ג':

נרצה להראות שהזזה בזמן = פאזה בתדר.

ניקח את a_k_real שחישבנו, נעבור איבר איבר ונכפיל אותו ב $e^{-j\frac{2\pi}{N}k\cdot 100}$ ונשמור את הערכים שנקבל במערך b_k.

נעשה np על b_k על מנת שנוכל הדפיס גרפים שלו.

```python
# Part C:
n0 = 100  # shift amount
b_k = []

for k_idx, k in enumerate(k_vals):
    shift = np.exp(-1j * 2 * np.pi * k * n0 / N)
    b_k.append(a_k_real[k_idx] * shift)

b_k = np.array(b_k)
```

כעת נרצה לעבור איבר איבר על מערך b_k ולהעביר את המקדמים למישור הזמן.

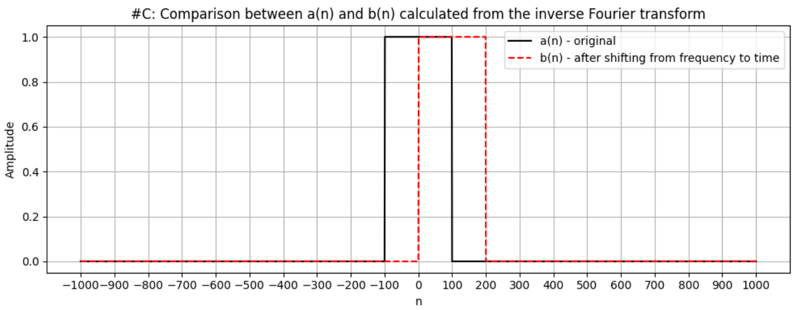
נגדיר מערך b_n בו נשמור את המקדמים שנחשב.

לוקחים כל איבר b_k ומכפילים אותו ב $e^{j\frac{2\pi}{N}k\cdot n\_val}$ ומקבלים את b_n עבור ערך n_val מסויים.

נעשה np על b_n על מנת שנוכל הדפיס גרפים שלו.

נדפיס את הגרפים a (בזמן) וb_n (מוזז, בזמן), כך שנראה שקיבלנו את אות החלון מוזז ב100 ימינה.

```python
b_n = []
for n_val in n:
    sum_val = 0
    for k_idx, k in enumerate(k_vals):
        angle = 2 * np.pi * k * n_val / N
        sum_val += b_k[k_idx] * np.exp(1j * angle)
    b_n.append(sum_val)  # Keep only real part

b_n = np.array(b_n)

# a(n) vs b(n):
plt.figure(figsize=(10, 4))
plt.plot(n, a, label='a(n) - original', color='black')
plt.plot(n, b_n, '--', label='b(n) - after shifting from frequency to time', color='red')
plt.title("Comparison between a(n) and b(n) calculated from the inverse Fourier transform")
plt.xlabel("n")
plt.xticks(np.arange(min(k_vals), max(k_vals)+1, 100))
plt.ylabel("Amplitude")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

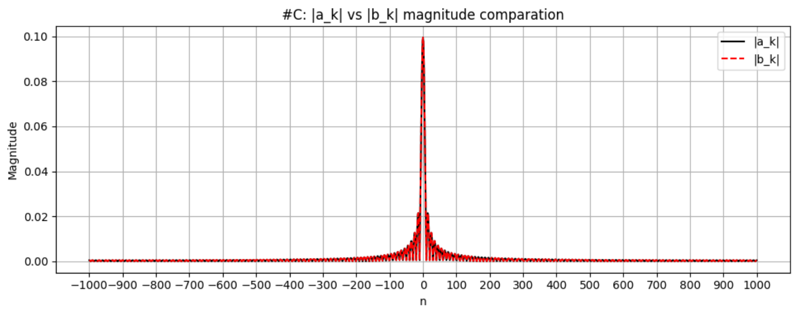


נדפיס את המגניטודה של כל מקדמי פורייה של a_k וb_k שזה בעצם לקחת ערך מוחלט שלהם.

נדפיס את הגרפים ונראה שהם מתלכדים, כלומר הפאזה בתדר לא משפיעה על המגניטודה (שמייצגת אנרגיה, כלומר האנרגיה לא משתנה).

ניתן לראות שקיבלנו sinc שזוהי התמרה המתאימה לאות החלון שלנו כפי שלמדנו.

```python
#a_k & b_k magnitude:
a_k_magnitud = [abs(x) for x in a_k_real]
b_k_magnitud = [abs(x) for x in b_k]

plt.figure(figsize=(10, 4))
plt.plot(n, a_k_magnitud, label='|a_k|', color='black')
plt.plot(n, b_k_magnitud, '--', label='|b_k|', color='red')
plt.title("#C: |a_k| vs |b_k| magnitude comparation")
plt.xlabel("n")
plt.xticks(np.arange(min(k_vals), max(k_vals)+1, 100))
plt.ylabel("Magnitude")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



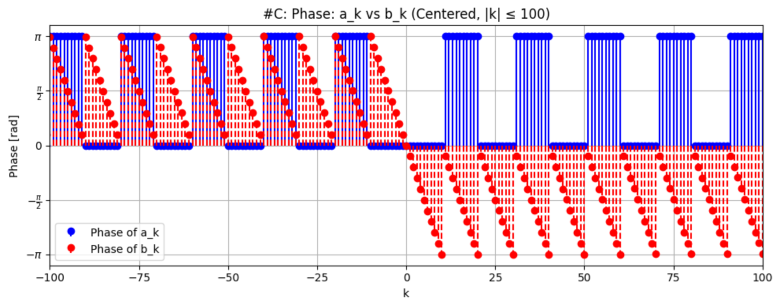
נרצה לחשב את הפאזה של כל מקדם על מנת לראות איך ההזזה השפיעה עליה.

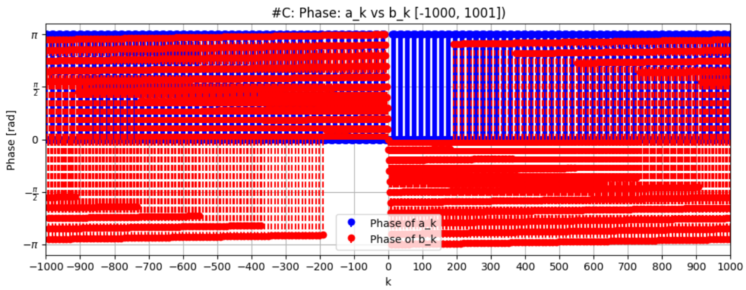
נעשה לולאת for שעוברת מקדם מקדם, מחלצת את הפאזה שלו, ושומרת אותו באינדקס המתאים במערך a_k_phase.

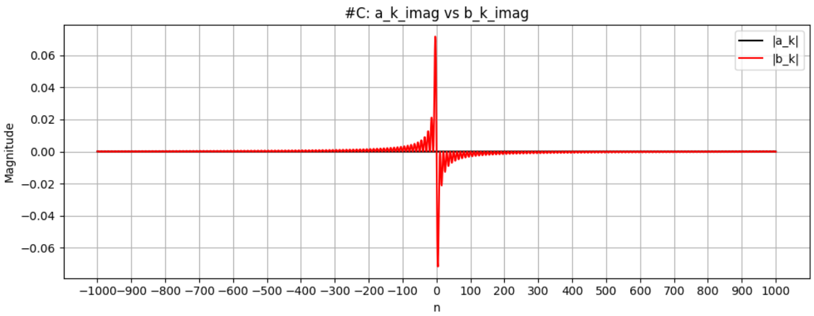
אותו דבר גם עבור b_k.

נראה בגרף שיש שוני בין הפאזות, כלומר הזזה בזמן גורמת לשינוי פאזה.

```
a_k_phase = [cmath.phase(x) for x in a_k_real]
b_k_phase = [cmath.phase(x) for x in b_k]

a_k_phase = np.array(a_k_phase)
b_k_phase = np.array(b_k_phase)

plt.figure(figsize=(10, 4))
plt.stem(n, a_k_phase, linefmt='b-', markerfmt='bo', basefmt=" ", label='Phase of a_k')
plt.stem(n, b_k_phase, linefmt='r--', markerfmt='ro', basefmt=" ", label='Phase of b_k')
plt.title("#C: Phase: a_k vs b_k (Centered, |k| ≤ 100)")
plt.xlabel("k")
plt.ylabel("Phase [rad]")
pi = np.pi
plt.yticks([-pi, -pi/2, 0, pi/2, pi], [r"$-\pi$", r"$-\frac{\pi}{2}$", r"$0$", r"$\frac{\pi}{2}$", r"$\pi$"])
plt.xlim(-100, 100)
plt.xticks(range(-100, 101, 25))
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```








```
#a_k vs b_k imag part
plt.figure(figsize=(10, 4))
plt.plot(n, a_k_real.imag, label='|a_k|', color='black')
plt.plot(n, b_k.imag, label='|b_k|', color='red')
plt.title("#C: a_k_imag vs b_k_imag")
plt.xlabel("n")
plt.xticks(np.arange(min(k_vals), max(k_vals)+1, 100))
plt.ylabel("Magnitude")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

 # השוואה בין הפאזות:

הזזה בזמן של a_k ב n צעדים ייתן מקדמי פורייה $b_k = a_k e^{-j\frac{2\pi}{N}kn}$

לכן מבחינת הפאזה של b_k נקבל $\angle b_k = \angle a_k - \frac{2\pi}{N}kn$ כלומר קיבלנו ביטוי ליניארי בפאזה החדשה. כלומר הזזה בזמן נותנת שיפוע קבוע בגרף הפאזה של מקדמי פורייה. בנוסף למדנו כי סימטריה במישור הזמן = ממשיות במישור התדר. הפאזה של a_k היא 0 או π.

נוסף על כך נשים לב כי באמת מתקיימת מחזוריות של 2π בהתאם להתנהגות הפאזה הידועה של מקדמי פורייה.

```
#b10
k_target = 10
index_k10 = np.where(k_vals == k_target)[0][0]  # find index of k=10
b10 = b_k[index_k10]

# Compute b_10(n) = b10 * e^{j*2π*10*n/N}
b10_n = b10 * np.exp(2j * np.pi * k_target * n / N)

# Plot b10(n)
plt.figure(figsize=(10, 4))
plt.plot(n, np.real(b10_n), label='Re{b₁₀(n)}', color='red', linewidth=2)
plt.plot(n, np.imag(b10_n), label='image{b₁₀(n)}', color='black', linewidth=2)
plt.title("#C: Time-domain signal of b₁₀(n) = b₁₀·e^(j·2π·10·n/N)")
plt.xlabel("n")
plt.xticks(np.arange(min(n), max(n)+1, 100))
plt.ylabel("Value")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

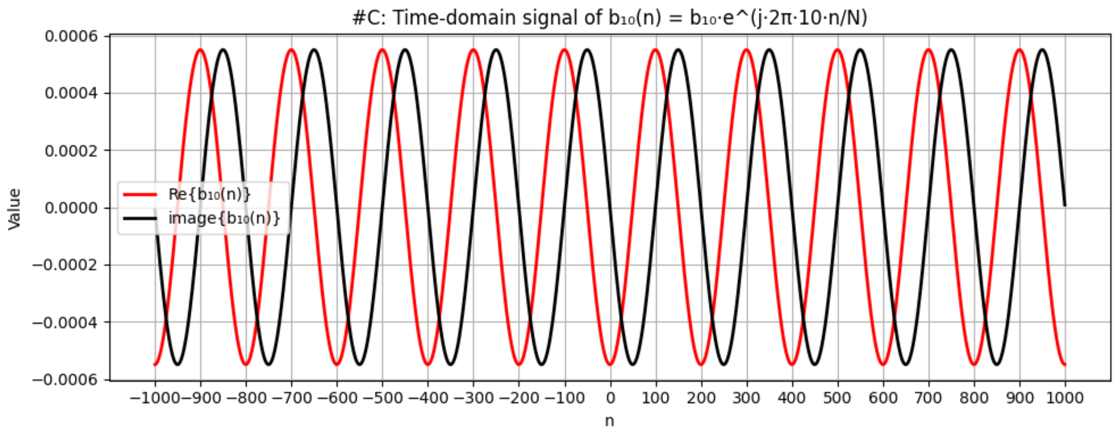
 # חישוב b10:

נחפש איפה במערך k_vals יש את הערך 10 ונשמור את האינדקס הזה בindex_k10 .

שומרים את הערך של b_k באינדקס שמצאנו בתוך b10.
זהו מקדם פורייה המתאים לb10.

עושים התמרת פורייה הפוכה על מנת לקבל את b10 במרחב הזמן.

מדפיסים את הערכים של החלק המדומה (בשחור) ואת של החלק הממשי (באדום), ונראה את השינוי לפי הזמן.



ביטוי מתמטי b10:

המקדמים בתחום התדר: $b_k = a_k e^{-j\frac{2\pi}{N}kn}$

המרה לתחום הזמן: $b_n = b_k * e^{\frac{j2\pi kn}{N}}$

המקדם b_{10} בזמן: $b_{10}(n) = b_{10} * e^{\frac{j2\pi 10n}{N}}$

העברה לאמפליטודה ופאזה: $b_{10}(n) = |b_{10}| * e\hat{}(j\left(\frac{2\pi 10n}{N} + \angle b_{10}\right))$

החלק הממשי: $Re(b_{10}(n) = |b_{10}| * cos\,(\frac{2\pi 10n}{N} + \angle b_{10})$

החלק המדומה: $Im(b_{10}(n) = |b_{10}| * sin\,(\frac{2\pi 10n}{N} + \angle b_{10})$

סעיף ד:

- בחינת הזהות: גזירה בזמן = הכפלה בa k בתדר

```python
#Part D:

# Calculate c_k
ck = a_k * (1 - np.exp(-2j * np.pi * k_vals / N))

# Calculate c(n) using the inverse Fourier transform
cn = []
n_vals = np.arange(-N//2, N//2 + 1)

for n_ in n_vals:
    sum_val = 0
    for k in range(N):
        angle = 2 * np.pi * (k_vals[k] * n_ / N)
        sum_val += ck[k] * np.exp(1j * angle)
    cn.append(sum_val.real)  # Only the real part
```

נגדיר ck לפי הנתון

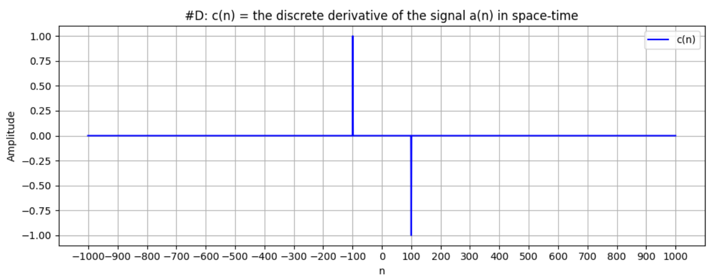
נגדיר רשימה ריקה cn

n_vals - וקטור ערכי זמן המתאים לערכי האות המקורי, למען הגרף

לולאת for: חישוב כל מקדם פורייה לפי הנוסחה $C_k * e^{nk\frac{2\pi}{N}j}$ והוספה לסכום הטור.

לוקחים רק את החלק הממשי של התוצאה (כי החלק המדומה הוא שגיאת עיגול זניחה) ושומרים אותה בcn.

```python
#Amplitude graph cn
plt.figure(figsize=(10, 4))
plt.plot(n_vals, cn, label='c(n)', color='b')
plt.title("#D: c(n) = the discrete derivative of the signal a(n) in space-time")
plt.xlabel("n")
plt.xticks(np.arange(min(n), max(n)+1, 100))
plt.ylabel("Amplitude")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



הסיגנל שלנו בתדר: $a\_k = (1/N) * \sum a(n) * e^{-j\cdot 2\pi kn/N}$

ההתמרה ההפוכה של *a_k*: $a(n) = \sum a\_k * e^{j\cdot 2\pi kn/N}$

נתון: $c\_k = a\_k * (1 - e^{-j\cdot 2\pi k/N})$

ההתמרה ההפוכה של *c_k*: $c(n) = \sum c\_k * e^{j\cdot 2\pi kn/N}$

נציב את *c_k* הנתון: $\sum a\_k * (1 - e^{-j\cdot\frac{2\pi kn}{N}}) * e^{j\cdot\frac{2\pi kn}{N}} = \sum a\_k * e^{j\cdot\frac{2\pi kn}{N}} - \sum a\_k * e^{(j\cdot 2\pi k(n-1)/N)}$

כלומר קיבלנו c(n) = a(n) - a(n-1)

הוכחנו שכאשר כופלים את מקדמי פורייה *ak* בפונקציה $(1 - e^{-j\cdot 2\pi k/N})$, מתקבל אות חדש c(n) שהינו ההפרש a(n) - a(n-1) שזוהי בדיוק הנגזרת הבדידה בזמן (מודדת את קצב השינוי של האות בין שתי דגימות סמוכות).

מבחינת הגרף:

הגרף ממחיש שהאות *c(n)* הוא כמו נגזרת — הוא מאפס כל מקום פרט לנקודות השינוי של פונקציית החלון שלנו *a(n)*:

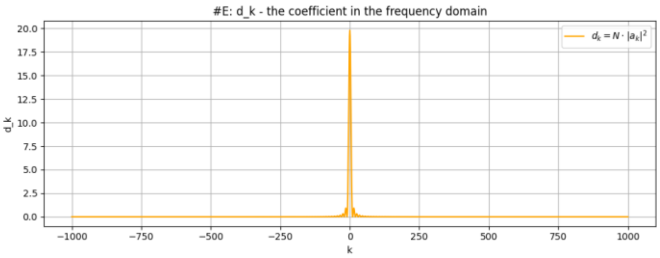
בתחילת החלון יש קפיצה מ־0 ל־1, ולכן c(n)=1. בסוף החלון יש ירידה חדה מ־1 ל־0, ולכן c(n)=-1.

בכל שאר הנקודות הערך קבוע ולכן אין שינוי c(n)=0.

סעיף ה':

- קונוולוציה בזמן = הכפלה בתדר
- צריך לחשב: $d_k = Na_k^2$:

```python
# Part E:

# Calculate d_k = N * |a_k|^2
d_k = N * (np.abs(a_k) ** 2)

# Plotting d_k
plt.figure(figsize=(10, 4))
plt.plot(k_vals, d_k, label=r'$d_k = N \cdot |a_k|^2$', color='orange')
plt.title("#E: d_k - the coefficient in the frequency domain")
plt.xlabel("k")
plt.ylabel("d_k")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



חישוב וגרף של d(n) – ההתמרה ההופכית של d_k (מעבר ממרחב התדר למרחב הזמן):

נגדיר d_n מערך חדש בגודל n מאותחל באפסים, בו נשמור כל מקדם שנחשב לפי האינדקס שלו.

לולאת for: בפייתון אי אפשר לעבור על אינדקסים שליליים, לכן נשתמש בו כאינדקס עולה עד שנעבור על כל האיברים. בנוסף נשתמש בn_val בתור הערך של n שבו נחשב את הפונקציה (הוא יכול להיות שלילי).

אנחנו מגדירים בלולאה אקספוננט לפי ההגדרה של התמרה הפוכה, ועושים כפל איבר איבר עם dk (אפשר לחלק בN לנרמול). את התוצאה אנחנו שומרים במערך d_n אותו אתחלנו באפסים, ולכן כשנוסיף את הסכום במקום הi, נקבל את ערך האיבר שחישבנו.

ואז יש הדפסה של הגרף.

```python
# Initialize time-domain vector
d_n = np.zeros_like(n, dtype=complex)

# Inverse DFT using manually coded sum
for i, n_val in enumerate(n):
    exponent = np.exp(2j * np.pi * k_vals * n_val / N)
    d_n[i] = np.sum(d_k * exponent)

# Take the real part
d_n_real = np.real(d_n)/N

# Plot d(n)
plt.figure(figsize=(10, 4))
plt.plot(n, d_n_real, label='d(n) - inverse transform of d_k', color='orange')
plt.title("#E: d(n) as the inverse Fourier transform of d_k")
plt.xlabel("n")
plt.xticks(np.arange(min(n), max(n)+1, 100))
plt.ylabel("Amplitude")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

נאשר אנליטית שקיבלנו קונלוולוציה בזמן:

לפי קונוולוציה – אם $X(k) \overset{F}{\leftrightarrow} x(n)$ וגם $Y(k) \overset{F}{\leftrightarrow} y(n)$ אז $X(k)Y(k) \overset{F}{\leftrightarrow} x(n) * y(n)$:

$a_n \overset{\mathcal{F}}{\leftrightarrow} a_k$

$\overline{a_n} \overset{\mathcal{F}}{\leftrightarrow} \overline{a_k}$

$\overset{1}{\Leftarrow}$ $a_k\overline{a_k} = |a_k|^2$ $\overset{2}{\Leftarrow}$ $|a_k|^2 \overset{\mathcal{F}}{\leftrightarrow} a_n * \overline{a_{-n}}$ $\overset{3}{\Leftarrow}$

אבל מצאנו במקרה שלנו ש a_n ממשי וסימטרי ולכן $a_n = \overline{a_{-n}}$

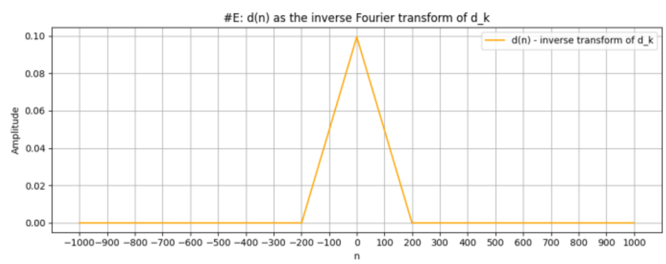
כלומר נוכל להגיד: $|a_k|^2 \overset{\mathcal{F}}{\leftrightarrow} a_n * a_n$

נחזור לנתון שלנו: $d_k = N|a_k|^2$

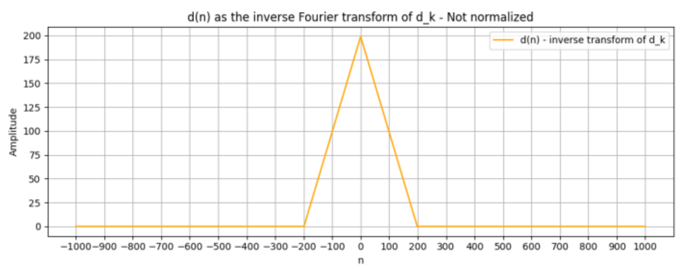
כלומר בהתמרת פורייה הפוכה באמת נקבל $d(n) = a_n * a_n$ (פקטור N נעלם מהגדרת ההתמרה, הוא מצטמצם).

כלומר הראנו כי נקבל שהכפלה בתדר שווה לקונוולוציה בזמן – ובמקרה בו האות ממשי וסימטרי, נקבל קונוולוציה של האות עם עצמו.

- d(n) מנורמל:



- d(n) לא מנורמל:



לא מנורמל: כפי שאמרנו, קונוולוציה בזמן שווה כפל בתדר ובנוסף במקרה שלנו מדובר על קונוולוציה של אות מלבן עם עצמו. נקבל את המקסימום כאשר תהיה חפיפה מלאה בין המלבנים (המלבנים שלנו הם מ-99- עד 99, כלומר האמצע הוא באמת אינדקס 0). אזי נקבל בחפיפה מלאה $d(0) = \sum_{-99}^{99} |a_n|^2$, וכל ערך $a_n = 1$ ולכן נקבל 199 (בפלט מהפונקציה מקבלים 199.00049 ולא בול 199 בגלל שגיאות נומריות זניחות של המחשב).

```
#E: The maximum value of d(n): 199.00000
#E: It occurs for n = 0
```

מנורמל: במקסימום אנחנו מקבלים 199 חלקי גורם הנרמול: $\frac{199}{2001} = 0.09945$.

```
#E: The maximum value of d(n): 0.09945
#E: It occurs for n = 0
```

סעיף ו':

parseval_time - הגדרת פרמטר של חישוב הנוסחה הנתונה לפי הזמן.

Parseval_freq - הגדרת פרמטר של חישוב הנוסחה הנתונה לפי התדר.

הדפסת ערכי הטורים.

check closeness - בדיקה שערכי הטורים שווים עד כדי שגיאה נומרית זניחה.

מוצגת ההדפסה שקיבלתי:

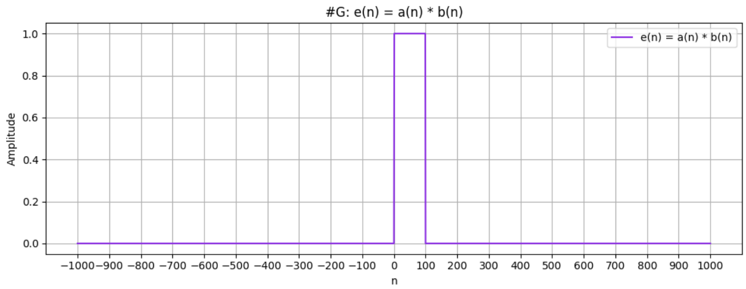
```
# Part F:

# Parseval's theorem check

# Left side: (1/N) * sum |d(n)|^2
parseval_time = np.sum(np.abs(d_n) ** 2)/N

# Right side: sum |d_k|^2
parseval_freq = np.sum(np.abs(d_k) ** 2)

# Print results
print(f"#F: Parseval - Time domain: {parseval_time:.6f}")
print(f"#F: Parseval - Frequency: {parseval_freq:.6f}")

# Check closeness
if np.isclose(parseval_time, parseval_freq):
    print("#F: Parseval's theorem holds (values are equal within numerical accuracy)")
else:
    print("#F: Parseval's theorem does not hold (unexpected discrepancy)")
```

```
#F: Parseval - Time domain: 2625.586707
#F: Parseval - Frequency: 2625.586707
#F: Parseval's theorem holds (values are equal within numerical accuracy)
```

סעיף ז':

- עבור e(n) = a(n) b(n) נקבל את הגרף הבא:
- התשובה שקיבלנו הגיונית לפי הגרפים שקיבלנו בסעיף ג' עבור a(n), b(n).

```python
# Part G:

# Compute e(n) = a(n) * b(n)
e_n = a * b_n  # A term-by-term multiplication of the two signals

# Plot e(n)
plt.figure(figsize=(10, 4))
plt.plot(n, e_n, label='e(n) = a(n) * b(n)', color='blueviolet')
plt.title("#G: e(n) = a(n) * b(n)")
plt.xlabel("n")
plt.xticks(np.arange(min(n), max(n)+1, 100))
plt.ylabel("Amplitude")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



e_k בתדר לאחר התמרה הפוכה של e_n:

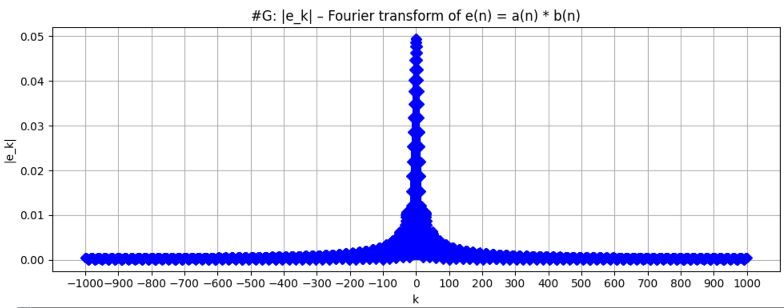
e_k - הגדרת מערך מאותחל באפסים.

לולאת for: מחשבים לפי הגדרה של התמרה הפוכה עבור כל k מתאים (index בשביל לעקוב שעוברים על כל k_vals), שמים באינדקס המתאים במערך e_k שיצרנו את הערך שחושב (מנורמל).

ציור גרפים.

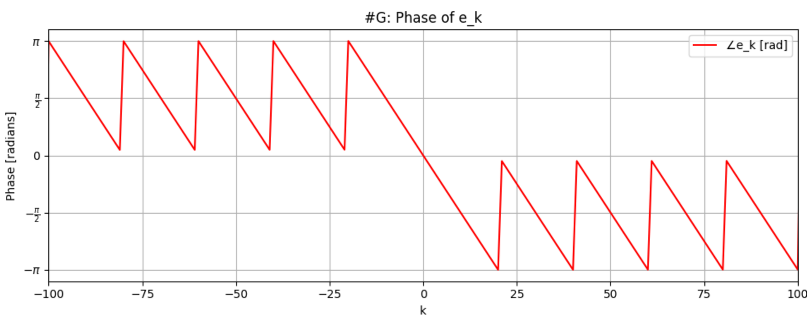
```python
# Initialize e_k array
e_k = np.zeros(len(k_vals), dtype=complex)

# Compute DFT manually
for index, k in enumerate(k_vals):
    exponent = np.exp(-2j * np.pi * k * n / N)
    e_k[index] = np.sum(e_n * exponent) / N

# Plot the magnitude of e_k
plt.figure(figsize=(10, 4))
plt.stem(k_vals, np.abs(e_k), linefmt='.b', markerfmt='bD', basefmt=" ", use_line_collection=True)
plt.title("#G: |e_k| - Fourier transform of e(n) = a(n) * b(n)")
plt.xlabel("k")
plt.xticks(np.arange(min(n), max(n)+1, 100))
plt.ylabel("|e_k|")
plt.grid(True)
plt.tight_layout()
plt.show()
```

מגניטודה של e_k:



פאזה של e_k:

```python
# Phase of e_k
plt.figure(figsize=(10, 4))
plt.plot(k_vals, np.angle(e_k), label="∠e_k [rad]", color='red')
plt.title("#G: Phase of e_k")
plt.xlabel("k")
plt.xticks(range(-100, 101, 25))
plt.xlim(-100, 100)
plt.ylabel("Phase [radians]")
plt.yticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi], [r"$-\pi$", r"$-\frac{\pi}{2}$", r"$0$", r"$\frac{\pi}{2}$", r"$\pi$"])
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



חלק ציקלי:

```
349 #Cyclic convolution:
350 # Replacement functions for ifftshift and fftshift (manual, allowed for use)
351 def manual_fftshift(x):
352     N = len(x)
353     return np.concatenate((x[N//2:], x[:N//2]))
354
355 def manual_ifftshift(x):
356     N = len(x)
357     return np.concatenate((x[-(N//2):], x[:-(N//2)]))
358
359 # Manual cyclic arrangement
360 a_k_shifted = manual_ifftshift(a_k)
361 b_k_shifted = manual_ifftshift(b_k)
362
363 e_k_circular = np.zeros_like(a_k, dtype=complex)
364 N_k = len(a_k)
365 for k in range(N_k):
366     total = 0
367     for l in range(N_k):
368         index = (k - l) % N_k
369         total += a_k_shifted[l] * b_k_shifted[index]
370     e_k_circular[k] = total
371 e_k_circular = manual_fftshift(e_k_circular)
```

```
373 # plot graph compared to the previous graph
374 plt.figure(figsize=(10, 5))
375 plt.plot(k_vals, np.abs(e_k), label="|e_k| from time-domain multiplication", color='blue')
376 plt.plot(k_vals, np.abs(e_k_circular), '--', label="|ê_k| from circular convolution", color='orange')
377 plt.title("#G: Comparison between |e_k| and |ê_k| (circular convolution)")
378 plt.xlabel("k")
379 plt.ylabel("Magnitude")
380 plt.legend()
381 plt.grid(True)
382 plt.tight_layout()
383 plt.show()
```

שורות 351-353: מחלקת את המערך לשני חצאים. שמה את החצי שבאינדקסים הגדולים יותר ,באינדקסים שליליים ,כך שנוצרת סימטריה במספר האיברים סביב 0 דוגמה: 1,2,3,4,5,6 הופך אחרי הפעל הפונקציה הזאת ל: 4,5,6,1,2,3

355-357: עושה את הפעולה ההפוכה ל - manual_fftshift כלומר מחזירה את הסדר לאיך שהיה בהתחלה. דוגמה: לוקח 4,5,6,1,2,3 ומחזיר 1,2,3,4,5,6.

360-361: סידור ציקלי ידני של a_k ו-b_k : בגלל k_vals הם היו מ -1000 ל1000 ואנחנו רוצים שהם יהיו מ0 עד 2001.

363: מכין מערך ריק (מלא באפסים) באותו גודל כמו a_k, שיכיל את התוצאה הסופית $\hat{e_k}$

364: קובע את גודל הסדרה – מספר התדרים (N=2001)

365-370: לולאת for מבצעת קונוולוציה ציקלית באופן ידני:

-מאתחלים סכום אפס כדי לצבור את סכום הקונבולוציה

-עוברים על כל ערך מ*0* עד N-1 לפי הנוסחה הנתונה בשאלה

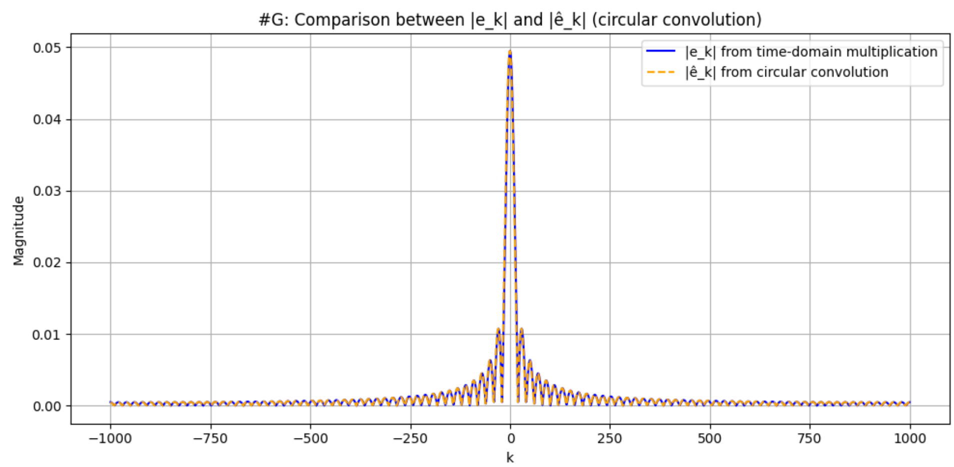
-במקום לגשת לb_{k-l} אנחנו עושים מודולו כדי לוודא שהאינדקס נשאר בטווח [0,N-1] כך שאם *k-l* שלילי, נחזיר אותו לטווח איתו אנחנו עובדים.

-מוסיפים לסכום הטור את $a_k b_{k-l} \% N$ לסכום.

-שומרים את התוצאה של הקונבולוציה עבור k.

371: החזרת התוצאה למבנה התדרים [N/2,...,N/2-] בעזרת הפונקציה שבניתי.

הדפסת הגרף שמשווה בין e_k ל $\hat{e_k}$:



סעיף ח':

388: הגדרת g_n הנתונה

394-398: חישוב כל מקדם לפי פורייה $g_k = \frac{1}{N}\sum_n g(n)\, e^{-j\frac{2\pi}{N}kn}$

401-402: חישוב הפיקים של $|g_k|$ ע"י חיפוש כל הערכים הקרובים לערך המקסימלי, וזיהוי האינדקסים שלהם.

405-406: לולאת for שרצה על האינדקסים שבהם נמצאו הערכים המקסימלים (או הקרובים להם), ומדפיסה את ערכי k בהם מתקבל פיק, ואת הערך שיש באינדקסים האלו (איפה שמתקבלת האנרגיה הכי גבוהה).

הדפסת הגרפים.

```python
386 #Part H:
387
388 g_n = a * np.cos(2 * np.pi * 500 * n / N)
389
390 # Allocate output array for g_k
391 g_k = np.zeros(len(k_vals), dtype=complex)
392
393 # Compute Fourier coefficients g_k
394 for i, k in enumerate(k_vals):
395     sum_val = 0
396     for j, n_ in enumerate(n):
397         sum_val += g_n[j] * np.exp(-1j * 2 * np.pi * k * n_ / N)
398     g_k[i] = sum_val / N  # normalization
399
400 # Find all indices where |g_k| equals the maximum value (within tolerance)
401 max_val = np.max(np.abs(g_k))
402 Indexes = np.where(np.isclose(np.abs(g_k), max_val, atol=1e-10))[0]
403
404 # Print all k values with maximum magnitude
405 for idx in Indexes:
406     print(f"Max |g_k| = {k_vals[idx]}, magnitude = {np.abs(g_k[idx]):.4f}")
```

```python
# Plot - real part of g_k
plt.figure(figsize=(10, 4))
plt.plot(k_vals, np.real(g_k), label="|g_k|", color='blue')
plt.title("#H: real part of g_k")
plt.xlabel("k")
plt.ylabel("g_k real")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

```python
# Plot - imag part of g_k
plt.figure(figsize=(10, 4))
plt.plot(k_vals, np.imag(g_k), label="|g_k|", color='blue')
plt.title("#H: imag part of g_k")
plt.xlabel("k")
plt.ylabel("g_k image")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

```python
# Plot phase - real part of g_k
plt.figure(figsize=(10, 4))
plt.stem(k_vals, np.angle(g_k.real), label="∠g_k", basefmt=" ")
plt.title("#H: Phase of the real part of g_k")
plt.xlabel("k")
plt.ylabel("Phase [radians]")
plt.yticks([-np.pi, 0, np.pi], [r"$-\pi$", r"$0$", r"$\pi$"])
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Cos מורכב משני אקספוננטים $\cos = \frac{e^{j\frac{2\pi}{w}n}+e^{-j\frac{2\pi}{w}n}}{2}$

נקבל את האנרגיה המקסימלית של הגל עבור ± 500 כי שם יש את הרכיבי התדר שבנו את הקוסינוס, ולכן שם נראה את הפיקים של האנרגיה. החלון a(n) רק חותך את הקוסינוס בזמן — אבל לא משנה את התדרים עצמם.

בגרף - קיבלנו פיקים ב500 וב-500:

```
Max |g_k| = -500, magnitude = 0.0495
Max |g_k| = 500, magnitude = 0.0495
```

החלון a(n) רק חותך את הקוסינוס בזמן — אבל לא משנה את התדרים עצמם.

הצדקה אנליטית:

$$g(n) = a(n)\cdot cos\left(\frac{2\pi\cdot 500\cdot n}{N}\right) = a(n)\cdot\frac{1}{2}\left(e^{j\cdot\frac{2\pi\cdot 500\cdot n}{N}} + e^{-j\cdot\frac{2\pi\cdot 500\cdot n}{N}}\right) = \frac{1}{2}a(n)e^{j\cdot\frac{2\pi\cdot 500\cdot n}{N}} + \frac{1}{2}a(n)e^{-j\cdot\frac{2\pi\cdot 500\cdot n}{N}}$$

אם נתמיר את מה שקיבלנו לתדר:

$$a(n) \to a_k$$

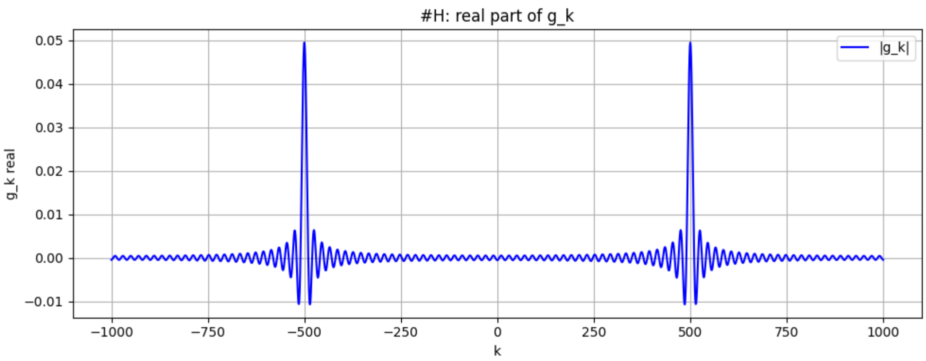
$$e^{j\cdot\frac{2\pi\cdot 500\cdot n}{N}} \to \delta(k-500)$$

$$e^{-j\cdot\frac{2\pi\cdot 500\cdot n}{N}} \to \delta(k+500)$$

לכן נקבל: $g(n) \to G_k = \frac{1}{2}a_{k-500} + \frac{1}{2}a_{k+500}$

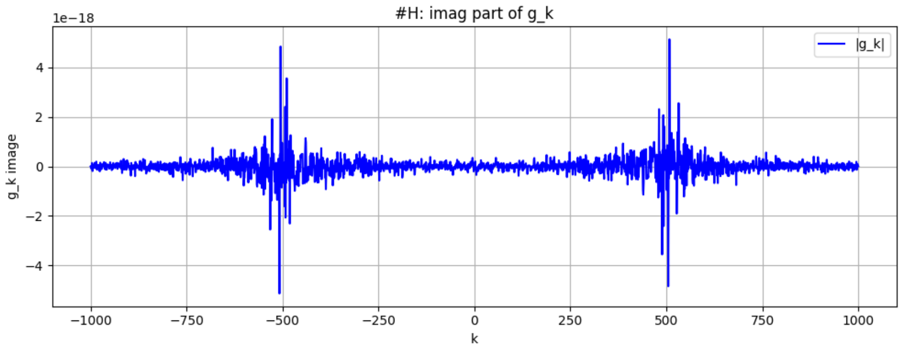
ומשום שהאות מחזורי, נקבל עבור $a_{k-(N-500)} = a_{k+500}$ ובאמת קיבלנו פיקים ב500, וב-500.

החלק הממשי של g_k:



החלק המדומה של g_k:

ניתן לראות שהוא זניח.



פאזה של חלק ממשי:

(לא נראה את המדומה כי הוא זניח).