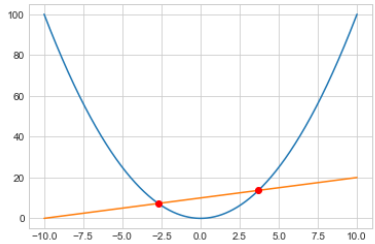
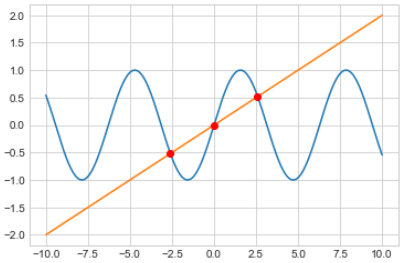
CVXPY

הספרייה cvxpy היא ספריית קוד פתוח בשפת פייתון עבור בעיות אופטימיזציה קמורה.
אופטימיזציה קמורה היא תת תחום של אופטימזציה מתמטית, המטפלת במקרים בהם פונקציית המטרה קמורה והאילוצים מגדירים מרחב שהוא קבוצה קמורה.
אופטימיזציה קמורה היא תחום רחב עם יישומים בעיקר בעיבוד אותות, תקשורת ורשתות, ניתוח נתונים, כלכלה, סטטיסטיקה, בניה ועוד מלא.
אז מה זה בכלל פונקציה קמורה? פונקציה קמורה היא פונקציה שעבור כל קו שנמתם בין שני נקודות שלה הקו יהיה מעל לגרף.
למשל הפונקציה $f(x) = x^2$ היא פונקציה קמורה כי כל שתי נקודות שנמתח בניהן קו הקו יהיה מעל לפונקציה:



לעומת זאת אם הפונקציה $f(x) = \sin(x)$ היא לא קמורה כי יש מקומות שבהם הקו יהיה מתחת לגרף:



אבל בתחום מסויים היא כן נחשבת קמורה למשל בין $x = \frac{\pi}{2}$ ל- $x = \frac{\pi}{2} + 2\pi$ (בין נקודת קיצון חיובית אחת לפעם הבאה שתהיה נקודת קיצון חיובית).

בדרך כלל בבעיות אופטימיזציה קמורות ננסה לענות על שאלות בסגנון של מה הערך של המקסימלי או המינימלי של איקס(או כמה משתנים) בפונקציה $f(x)$ בתחום $[a, b]$.
דוגמא מתורת המשחקים: נניח יש לנו שלוש עוגות ושני אנשים שרוצים לחלוק את העוגות, ולכל אחד יש העדפות שונות:

עוגה / אנשים	א'	ב'	ג'
אבי	2	3	4
בני	8	7	6

אבי מעריך את עוגה א' כ-2, את עוגה ב' כ-3 ועוגה ג' כ-4
בני מעריך את עוגה א' כ-8, את עוגה ב' כ-7 ועוגה ג' כ-6
אנחנו רוצים למצוא חלוקה הוגנת של העוגות כך שהחלוקה שהתקבלה לא ניתנת לשיפור- לא ניתן למצוא חלוקה חדשה שבה מישהו יקבל יותר ממה שהוא קיבל כבר בחלוקה הראשונה (יעריך יותר את החלוקה החדשה) , והשני יישאר עם אותם ערכים שהיו לו בחלוקה הקודמת.

אנחנו מחפשים פונקציה שתמקסם לנו ערכים x y ו- z שמייצגים כמה אחוז אבי לקח מהעוגות ו- (1-x), (1-y) ו- (1-z)

האחוזים שבני לקח מהעוגות.
מבלי להרחיב בנושא הפונקציה הלוגריתמית ידועה כמתאימה ביותר לאופטימיזציה של בעיות חלוקה בסגנון הזה (היא שומרת שהחלוקה תהיה גם ללא קנאה), אז נשתמש בה ונקבל את הפונקציה הבאה:

$$\log(2x + 3y + 4z) + \log(8(1 - x) + 7(1 - y) + 6(1 - z))$$

ועכשיו שאנחנו יודעים כי זאת בעיה מקסימום מקומי בתחום $0 \le x, y, z \le 1$, ובאמת הנגזרת של פונקציה לוגריתמית היא קמורה נוכל להשתמש בספרייה למציאת פתרון מקסימלי.

הכרות בסיסית עם הספרייה-

אובייקט Problem- לספרייה יש אובייקט מיוחד שמגדיר בעיה , ונקרא Problem.
Problem מקבל כפרמטר סוג בעיה , למשל בעיית מקסימום או מינימום, ואילוצים למשל שהמשתנים יהיו בטווח מסיום.
גם סוגי הבעיות הן אובייקט של cvxpy ומקבלות כפרמטר פונקציית המטרה, הפונקצייה יכולה להכיל כמה נעלמים, וצריך להגדיר מראש שאותם משתנים שהצגנו בפוקציה הם מטיפוס Variable שהם עוד אובייקט שייך לספרייה.
האובייקט Variable יכול להיות מספר בודד (סקאלר), וקטור או מטריצה:

```
import cvxpy as cp
from cvxpy import log

# A scalar variable.
a = cp.Variable()

# Vector variable with shape (5,).
x = cp.Variable(5)

# Matrix variable with shape (5, 1).
x = cp.Variable((5, 1))

# Matrix variable with shape (4, 7).
A = cp.Variable((4, 7))
```

האובייקט Problem מכיל בתוכו את המתודה ()solve שפותרת את הבעיה אותה קיבל האובייקט עם האילוצים, ומעדכנת את התוצאה בשדה של המחלקה בשם value, את מצב הפתרון בשדה בשם status, ואת התוצאות של הנעלמים במחלקה שלהם, כך שניתן לראות את הערכים שהם קיבלו מהמתודה בשדה value של המשתנים.
האילוצים צריכים להיות בצורה של ביטוי בוליאני המשתמש רק באחד מהאופרטורים הבאים: == , <= ו- >= .
אפשר להגדיר גם כמה אילוצים ברשימה, אבל יש לשים לב שאילוץ חייב להיות ביטוי בודד למשל לא יכול להיות ביטוי בסגנון של *0<=x<=1* וגם הביטויים חייבים להיות משוערים, כלומר קטנים\ גדולים שווים, ולא קטנים\גדולים ממש, למעט שוויון שהוא היחיד שיכל להיות ממשי:

```
x = cp.Variable()
y = cp.Variable()

# Create two constraints.
constraints = [x + y == 1,
               x - y >= 1]

# Form objective.
obj = cp.Minimize((x - y)**2)

# Form and solve problem.
prob = cp.Problem(obj, constraints)
prob.solve()  # Returns the optimal value.
optimal_x = round(float(x.value),5)
optimal_y = round(float(y.value),5)
print("status:", prob.status)
print("optimal value", prob.value)
print(f"optimal var: x = {optimal_x} , y = {optimal_y} ")
```

```
print(f"optimal var precise : x = {x.value} , y = {y.value} ")
```

```
status: optimal
optimal value 1.0
optimal var: x = 1.0 , y = 0.0
optimal var precise : x = 1.0 , y = 1.570086213240983e-22
```

נחזור לשאלה שהצגנו מקודם עם העוגה ונראה כיצד נוכל לפתור אותה עם הספרייה:

```
print("\nPROBLEM: ")
print("Three cakes have to be divided among 2 people with values:")
print("2 3 4")
print("8 7 6")

# Define x,y,z = the fraction of each region given to player 1.
x = cvxpy.Variable()
y = cvxpy.Variable()
z = cvxpy.Variable()

print("\nMaximize the sum of logs:")
prob = cvxpy.Problem(
    objective   =  cvxpy.Maximize(log(2*x + 3*y + 4*z) + log(8*(1-x)+7*(1-y)+6*(1-z))),
    constraints = [0 <= x, x <= 1, 0<= y , y <= 1, 0 <= z, z <= 1])
prob.solve()
print("status:", prob.status)
print("optimal value", prob.value)
print("optimal product", math.exp(prob.value))
print("optimal x", round(float(x.value),5))
print("optimal y", round(float(y.value),5))
print("optimal z", round(float(z.value),5))
```

```
PROBLEM:
Three cakes have to be divided among 2 people with values:
2 3 4
8 7 6

Maximize the sum of logs:
status: optimal
optimal value 4.150102096431418
optimal product 63.44047701380348
optimal x 0.0
optimal y 0.40476
optimal z 1.0
```

אז החלוקה אופטימלית תהיה לתת לבני את כל עוגה א' , ולאבי את כל עוגה ג', ובעוגה ב' הם יתחלקו בערך *0.4* אחוז מהעוגה ילך לאבי והשאר לבני.

הערה: לא תמיד הסטטוס של הבעיה יהיה אופטימלי, ישנן שאלות שבהן הערך שחזר הוא קירות, או בעיות בהן לא ניתן להגיע לתוצאה בכלל למשל במקרה הבא:

```
x = cp.Variable()

# An infeasible problem.
prob = cp.Problem(cp.Minimize(x), [x >= 1, x <= 0])
prob.solve()
print("status:", prob.status)
print("optimal value", prob.value)
```

```
# An unbounded problem.
prob = cp.Problem(cp.Minimize(x))
prob.solve()
print("status:", prob.status)
print("optimal value", prob.value)
```

```
status: infeasible
optimal value inf
status: unbounded
optimal value -inf
```

הסטטוס יכול לחזור בצורות הבאות:

`OPTIMAL`

`INFEASIBLE`

`UNBOUNDED`

`OPTIMAL_INACCURATE`

`INFEASIBLE_INACCURATE`

`UNBOUNDED_INACCURATE`

כך נוכל לבחון האם התוצאה שקיבלנו אכן אופטימלית, או בקירוב וכו'.

יש עוד מלא דברים שאפשר לעשות עם הספרייה ומומלץ להסתכל באתר הרשמי שלה לעוד אינפורמציה.