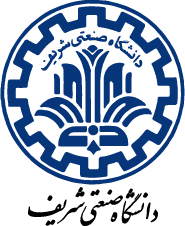
**به نام خدا**



**تمرین takehome**

**برنامه­ریزی تصادفی**

**احمد امامی**

**99207521**

**فهرست مطالب**

[**شرح مسئله** **3**](#_Toc94823535)

[**پارامترهای مدل 4**](#_Toc94823536)

[**متغیرهای تصمیم مدل 4**](#_Toc94823537)

[**تابع هدف 4**](#_Toc94823538)

[**محدودیتها 4**](#_Toc94823539)

[**حل به کمک الگوریتم کاهش سناریو (پسرو) و تولید سناریوفن 5**](#_Toc94823540)

[**پیادهسازی الگوریتم کاهش سناریو در پایتون 5**](#_Toc94823541)

[**توضیحات کد الگوریتم کاهش سناریو 7**](#_Toc94823542)

[**حل مسئلهی کشاورز به کمک سناریوهای کاهش یافته شده 8**](#_Toc94823543)

[**حل به کمک الگوریتم تجزیهی تو در تو 10**](#_Toc94823544)

[**کد پایتون الگوریتم تجزیه ی تو در تو 12**](#_Toc94823545)

[**مقایسهی نتایج دو الگوریتم و جمع بندی 19**](#_Toc94823546)

\*\* تمام کد­ها و خروجی­های مرتبط با هرکدام در فایل نوت­بوک پایتون، بدون اینکه نیاز به ران کردن مدل داشته باشید قابل مشاهده است.

**مسئله­ی تصمیم­گیری کشاورز را در حالت چند مرحله­ای در نظر بگیرید. برای حالت­های زیر مسئله را نوشته و جواب بهینه را به دست بیاورید:**

**1. تابع توزیع پیوسته برای متغیر­های تصادفی و استفاده از روش حل scenario reduction**

**2. تابع توزیع گسسته برای متغیر­های تصادفی و استفاده از روش حل تجزیه­ی تو در تو**

**جواب­های بدست آمده از دو روش را با یکدیگر مقایسه کنید. برای این منظور نیاز است تعداد مراحل در هر دو حالت مشابه و ارتباط معناداری بین توابع توزیع و متغیر­های تصادفی وجود داشته باشد.**

# شرح مسئله

همانطور که در مسئله­ی کشاورز صفحه­ی 4 کتاب توضیح داده شده است، کشاورزی میخواهد سه محصول گندم، ذرت و چغندرقند را در زمینی به مساحت 500 هکتار و در طول دو سال متمادی کاشت و برداشت نماید. هدف این کشاورز اینست که ضمن در نظر گرفتن محدودیتهای دنیای واقعی طوری زمین زیر کشت خود را به کاشت این سه محصول تخصیص دهد که هزینه­هایش کمینه شود.

برای حالت سه مرحله­ای که در صفحه­ی 18 کتاب درسی بیان شده است، ابتدا مدل مربوطه را می­نویسیم:

همان طور که در صورت سوال نیز مطرح شده است، مرحله­ی اول در این مدل به کشت محصول در سال اول اختصاص دارد. در مرحله­ی دوم تصمیم گرفته می­شود که با مقادیر برداشت شده از کشت اول چه باید کرد و چه مقدار باید فروخته و چه مقدار باید خریداری شود. در این مرحله هم­چنین مقدار کشت هر محصول برای برداشت در سال دوم نیز تعیین می­شود. در مرحله­ی سوم و پایانی نیز در ارتباط با محصولات کشت دوم تصمیم­گیری می­شود.

|  |  |  |
| --- | --- | --- |
| T=3  مرحله­ی سوم | T=2  مرحله­ی دوم | T=1  مرحله­ی اول |
| تصمیم در ارتباط با برداشت سال دوم | تصمیم در ارتباط با برداشت سال اول و کاشت محصول برای سال دوم | کاشت محصول (سال اول) |

با توجه به توضیحات مطرح شده مدل مسئله­ را می­نویسیم:

## پارامتر­های مدل

|  |  |
| --- | --- |
| میزان برداشت محصول i در سناریوی s |  |
| احتمال رخداد سناریوی s |  |
| هزینه­ی کاشت هر هکتار از محصول i |  |
| قیمت فروش محصول i (برای چغندر قند قیمت بالاتر را و قیمت پایین­تر را با نمایش می­دهیم |  |
| قیمت خرید هر تن محصول i |  |
| حداقل کشت مورد نیاز از محصول i |  |

## متغیر­های تصمیم مدل

|  |  |  |
| --- | --- | --- |
| میزان زمین اختصاص داده شده به کشت برای محصول i |  |  |
| میزان خرید گندم مرحله­ی t و تحت سناریوی s |  |  |
| میزان خرید ذرت مرحله­ی t و تحت سناریوی s |  |
| میزان فروش گندم در مرحله­ی t و تحت سناریوی s |  |  |
| میزان فروش ذرت در مرحله­ی t و تحت سناریوی s |  |
| میزان فروش چغندرقند با قیمت بالا در مرحله­ی t و تحت سناریوی s |  |
| میزان فروش چغندرقند با قیمت پایین در مرحله­ی t و تحت سناریوی s |  |

## تابع هدف

## محدودیت­ها

# حل به کمک الگوریتم کاهش سناریو (پسرو) و تولید سناریوفن

همان­طور که می­دانیم پارامتر غیرقطعی که سناریو­ها را به کمک آن تعریف می­کنیم می­باشد. ابتدا باید توزیع پیوسته­ای را برای این پارامتر در نظر بگیریم. در این مسئله فرض کردیم که این مقادیر از توزیع نرمال پیروی می­کنند. در حالت گسسته مقادیر برداشت عادی برای این سه محصول به ترتیب برابر با 2.5 ، 3 و 20 می­باشد. اگر این مقادیر را میانگین توزیع در نظر بگیریم میتوانیم توزیع نرمالی را برای هر کدام در نظر بگیریم. فرض می­کنیم که هر یک توزیعی به شکل زیر داشته باشند:

سناریو­های تولید شده توسط توابع پیوسته­ی بالا بسیار زیاد هستند در نتیجه نیاز داریم تا آن­ها را به صورت گسسته تقریب بزنیم و به کمک کاهش سناریو، سناریو­های محدودی را به منظور حل نهایی بکار ببریم. برای تشکیل سناریوفن بازه­ی را به 100 قسمت مساوی تقسیم می­کنیم. پس از تقسیم، به منظور پیش­بردن محاسبات نیاز است که احتمال هر بخش را حساب کنیم. وسط هر یک از این 100 بازه­ را به عنوان یک سناریو در نظر می­گیریم. نتایج حاصله را در یک فایل اکسل قرار داده­ایم و سپس به کمک آن الگوریتم کاهش سناریو را می­نویسیم.

# پیاده­سازی الگوریتم کاهش سناریو در پایتون

برای پیاده سازی الگوریتم از پایتون و کتابخانه­ی docplex استفاده نموده­ایم. این کتابخانه امکانات متعددی را در اختیار دارد که به راحتی می­توان بدون نیاز به نرم افزار cplex، مسائل برنامه­ریزی را در محیط پایتون کد کرده و حل نمود. کد الگوریتم را در شکل زیر مشاهده می­کنید. لازم به ذکر است که الگوریتم ذیل برای کاهش سناریو­ها به 10 عدد نوشته شده است.

# IMPORTING REQUIRED LIBRARIES

import pandas as pd

import numpy as np

from math import  inf

import matplotlib.pyplot as plt

# FIRST WE IMPORT OUR SCENARIOS INTO A VARIABLE

scenarios=pd.read\_excel('scenarios.xlsx',header=None)

# LET'S TAKE A LOOK

scenarios.head(10)

# FIRST THREE COLUMN OF OUR DATASET CONSIST OF OUR SCENARIOS

# SO WE NEED TO FILTER IT OUT FIRST AND PUT IT A NEW VARIABLE WE LIKE TO CALL MAIN\_SCENARIOS

main\_scenarios=scenarios.iloc[:,:3]

# WE ALSO NEED Probabilities OF EACH SCENARIO

probabilities=np.array(scenarios.iloc[:,5])

# NOW WE DEFINE OUR DISTANCE MATRIX

scenario\_distance\_matrix=np.zeros([100,100])

#FIRST WE DEFINE OUR minimum AND epsilon

minimum=0

epsilon=0.065

# (minimum < epsilon) makes sure that the min distance between scenarios does not surpass epsilon

while minimum<epsilon:

    n=scenario\_distance\_matrix.shape[0]      #this gives out number of rows in our distance matrix

    m=scenario\_distance\_matrix.shape[1]      #this gives out number of columns in our distance matrix

    for i in range(n):

        for j in range(m):

            if i==j:

                scenario\_distance\_matrix[i][j]=inf   # we set the distance of each scenario from itself to infinite

            else:

                a=np.array(main\_scenarios.iloc[i])   # take scenario a

                b=np.array(main\_scenarios.iloc[j])   # take scenario b

                dist=np.sqrt(np.sum((a-b)\*\*2, axis=0))  # calculate the distance between these two

                scenario\_distance\_matrix[i][j]=dist\*probabilities[i]  # update distance matrix

    minimum=scenario\_distance\_matrix.min()

    # NOW WE SEARCH TO SEE WHICH ROW AND COLUMN CONTAINS THE MINIMUMM VALUE

    # THEN WE PROCEED TO DELETE IT

    for i in range(n):

        for j in range(m):

            if scenario\_distance\_matrix[i][j]==minimum:

                row=i

                column=j

                break

    #update scenarios

    # we delete the row and column which contains the min value from distance matrix

    minimum=scenario\_distance\_matrix[row,column]

    probabilities[column]=probabilities[row]+probabilities[column]   # we add the deleted scenario probabilty to the one we kept

    probabilities=np.delete(probabilities,row,axis=0)                # we update our probability array for next loop

    # NOW WE UPDATE THE MAIN SCENARIOS DATAFRAME AND ALSO THE DISTANCE MATRIX

    main\_scenarios=main\_scenarios.drop(row).reset\_index().drop(["index"], axis=1)

    scenario\_distance\_matrix=np.delete(scenario\_distance\_matrix,row,axis=0)

    scenario\_distance\_matrix=np.delete(scenario\_distance\_matrix,column,axis=1)

    # This Line of code is optional

    # here we wanted to make sure that our dataframe is reduced to 10 scenarion, no more or less

    if len(main\_scenarios)==10:

        break

#let's see the new scenarios

print('reduced scenarios \n',main\_scenarios)

print('\nupdated probabilities \n',probabilities)

#############################################################################################

#############################################################################################

#############################################################################################

# توضیحات کد الگوریتم کاهش سناریو

ابتدا فایل اکسل سناریو­ها را در پایتو فرا می­خوانیم و آن را در متغیری ذخیره می­کنیم. سپس سه ستون اول دیتاست مربوطه را در متغیری جدا ذخیره می­کنیم. احتمالات این سناریو­ها را نیز در متغیری جداگانه ذخیره می­کنیم. پس از آن یک ماتریس صفر که از 100 سطر و ستون ( به تعداد سناریو­های تولید شده)تشکیل شده را تعریف می­کنیم. این ماتریس، **ماتریس فواصل** ما خواهد بود و مقادیر فاصله­ی هر سناریو از سناریو­های دیگر را در آن قرار خواهیم داد. پس از این قسمت شروع به تعریف الگوریتم می­کنیم.

ابتدا مقداری را برای اپسیلون در نظر می­گیریم. این مقدار برای بررسی شرطی است که اجازه ندهیم حداقل فواصل بین سناریو­ها از حد مشخصی بیشتر شود. در صورتی که حداقل فاصله بین سناریوها از این مقدار عبور کند و بیشتر شود دیگر سناریویی را حذف نخواهیم کرد. برای نوشتن الگوریتم نیاز به چند حلقه­ی تو در تو داریم. ابتدا و پس از چک کردن شرط کوچکتر بودن مقدار مینیموم فاصله­ها از اپسیلون تعریف شده، به ازای هر سطر و ستون i و j فاصله­ی سناریوی i و j از هم محاسبه شده و پس از ضرب این فاصله در احتمال سناریوی i، به سطر و ستون متناظر در ماتریس فواصل اختصاص می­یابد. هم­چنین مقادیر روی قطر اصلی را بی­نهایت می­گذاریم زیرا فاصله­ی هر سناریو از خودش همواره صفر است و نمی­خواهیم آن را در بررسی­مان دخیل کنیم. این حلقه به همین ترتیب تکرار می­شود تا تمام عناصر ماتریس فواصل تکمیل گردد. سپس مقدار مینیموم فاصله را حساب کرده و در متغیر minimum قرار می­دهیم. سپس اندیس سطر و ستون این مقدار مینیموم را در ماتریس فواصل پیدا می­کنیم و سناریوی مربوط به اندیس صف را از مجموعه سناریو­های موجود حذف می­کنیم. پس از حذف سناریوی حذف شده، احتمال آن به احتمال سناریوی باقی مانده افزوده می­شود. مجددا این حلقه تکرار شده و این بار ماتریس فواصل با ابعاد 99\*99 را خواهیم داشت و مجددا فواصل را حساب کرده و مینیموم را حذف می­کنیم تا به تعداد سناریوی مدنظرمان برسیم. با گذر مینیموم از مقدار 0.065 الگوریتم متوقف می­شود.

**\* توضیحات به صورت کامل­تر به صورت کامنت در کد قرار داده شده است.**

# حل مسئله­ی کشاورز به کمک سناریو­های کاهش یافته شده

from docplex.mp.model import Model

FARMER=Model(name='farmer problem using reduced scenario')

#parameters

p=probabilities

wheat\_yield = main\_scenarios[0]

corn\_yield = main\_scenarios[1]

sugarbeet\_yield = main\_scenarios[2]

planting\_cost=[150,230,260]

buying\_price=[238,210]

selling\_price=[170, 150, 36, 10]

#variables

x=FARMER.continuous\_var\_matrix(range(1,4),range(1,3),name='x',key\_format='%s')

y=FARMER.continuous\_var\_cube(range(1,3),range(1,11),range(2,4),name='y',key\_format='%s')

w=FARMER.continuous\_var\_cube(range(1,5),range(1,11),range(2,4),name='w',key\_format='%s')

#constraints

# PLANTING CONSTRAINT FOR WHEAT

for t in range(1,3):

    FARMER.add\_constraint\_(x[1,t]+x[2,t]+x[3,t]<=500)

# LEAST AMOUNT REQUIRED FOR WHEAT CONSTRAINTS

for t in range(1,3):

    for s in range(1,11):

        FARMER.add\_constraint\_(wheat\_yield[s-1]\*x[1,t]+y[1,s,t+1]-w[1,s,t+1]>=200)

# LEAST AMOUNT REQUIRED FOR CORN CONSTRAINTS

for t in range(1,3):

    for s in range(1,11):

        FARMER.add\_constraint\_(corn\_yield[s-1]\*x[2,t]+y[2,s,t+1]-w[2,s,t+1]>=240)

# SELLING AMOUNT SHOULD NOT EXCEED THE CULTIVATION

for t in range(1,3):

    for s in range(1,11):

        FARMER.add\_constraint\_(sugarbeet\_yield[s-1]\*x[3,t]-w[3,s,t+1]-w[4,s,t+1]>=0)

# CONSTRAINT ON SELLING AMOUNT WITH HIGHEST PRICE FOR SUGARBEET

for t in range(2,4):

    for s in range(1,11):

        FARMER.add\_constraint\_(w[3,s,t]<=6000)

# CAN'T PLANT SUGARBEET ON THE SAME FARM FOR TWO YEARS STRAIGHT

FARMER.add\_constraint\_(x[3,2]<=x[1,1]+x[2,1])

# DEFINE OBJECTIVE FUNCTION

FARMER.minimize(sum(planting\_cost[i]\*x[i+1,t] for i in range(3) for t in range(1,3))+

sum(probabilities[s-1]\*(buying\_price[i-1]\*y[i,s,t]-selling\_price[k-1]\*w[k,s,t]) for s in range(1,11)

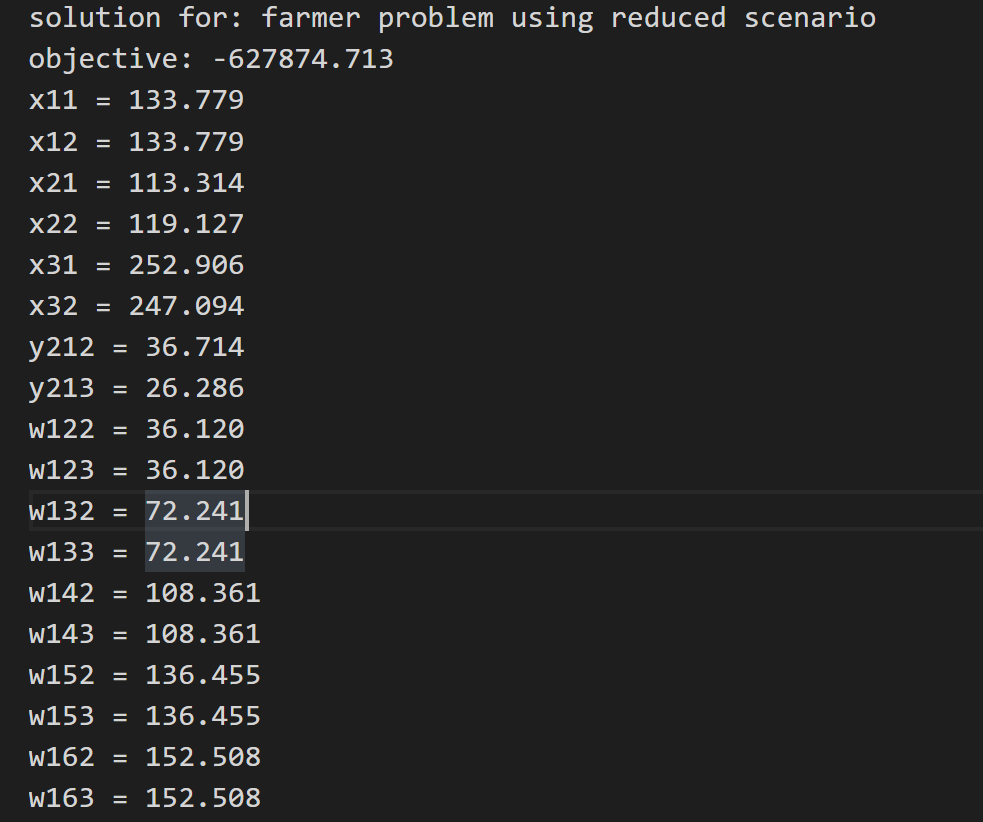
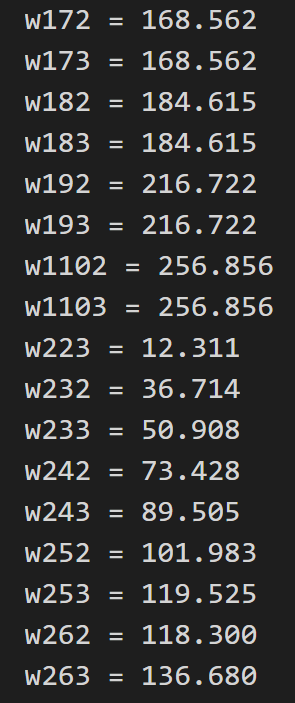
for i in range(1,3) for t in range(2,4) for k in range(1,5)))

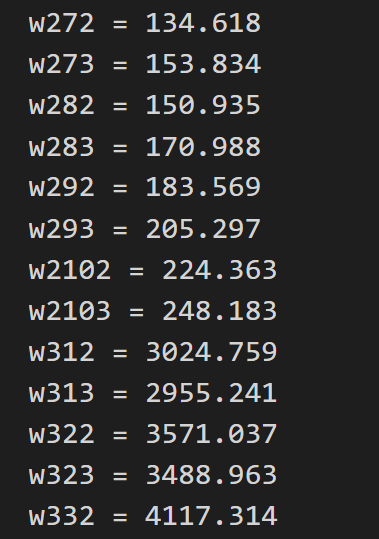
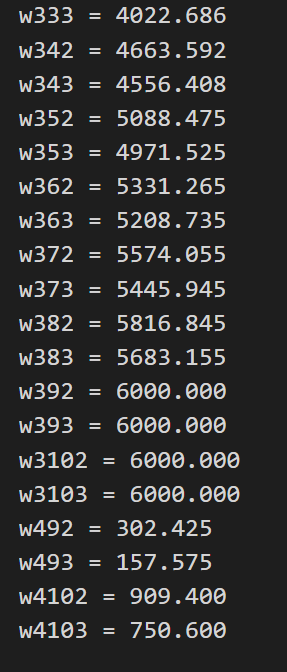
#PRINT solution

Solution=FARMER.solve()

Solution.display()

همان طور که در کد بالا مشاهده می­کنیم به کمک خروجی­های الگوریتم پیشین مسئله­ی کشاورز را مجددا فرموله می­کنیم و نتایج حاصله به ترتیب زیر می­باشد. ( اندیس ها از چپ به راست i (نوع محصول) s (سناریو) و t (پریود مورد بررسی) می­باشد)





# حل به کمک الگوریتم تجزیه­ی تو در تو

در این بخش از همان سناریو­های مطرح شده در کتاب برای حل مسئله استفاده می­کنیم. در نتیجه سه سناریوی کم­باران، متوسط و پرباران را خواهیم داشت. برای سناریوی کم­باران احتمال 0.3، متوسط 0.4 و پرباران 0.3 را اختصاص می­دهیم.

برای اینکه درک بهتری از الگوریتم کد شده در نرم­افزار بدست بیاوریم مسئله را برای گام اول و جهت Dir می­نویسیم:

برای تا نیز به همین ترتیب است و تنها اندیس s در متغیر­های خرید و فروش تغییر خواهد کرد.

# کد پایتون الگوریتم تجزیه ی تو در تو

\*\* توضیحات کد به صورت کامل و جامع به صورت کامنت در کد قرار داده شده است. در این بخش صرفا توضیح مختصری از نحوه­ی تعریف منطق مدل خواهیم داد.

from termcolor import colored

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import scipy.stats as scs

from math import  inf

from docplex.mp.model import Model

#parameters

yields={1:[3,2.5,2,3,2.5,2,3,2.5,2] , 2:[3.6,3,2.4,3.6,3,2.4,3.6,3,2.4] , 3:[24,20,16,24,20,16,24,20,16]}

planting\_cost=[150,230,260]

buying\_price=[238,210]

selling\_price=[170, 150, 36, 10]

# i is our counter which we can use to set how many loop we want our algorithm to Run

i=1

#define E21 and e21

#these variables are used to produce cuts in each iteration

E21=np.zeros([10,3])

e21=np.zeros(10)

E11=np.zeros([10,3])

e11=np.zeros(10)

while i<=10:

    print ('\n',colored('##############################################################################################','green')

           ,'\n',

           colored('###########################################Iteration{}#######################################'.format(i),'green')

           ,'\n',colored('#############################################################################################','green'),

           '\n')

    print(colored('\n DIR:FORWARD \n','red'))

    #FORWARD DIRECTION

    #FIRST WE SOLVE NLDS1 PROBLEM

    NLDS1=Model(name='NLDS1')

    x1=NLDS1.continuous\_var\_matrix(range(1,4),range(1,3),name='x',key\_format='%s')

    teta1=NLDS1.continuous\_var\_list(range(1,2),lb=-inf,name='teta1',key\_format='%s')

    NLDS1.add\_constraint\_(x1[1,1]+x1[2,1]+x1[3,1]<=500)

    # THE LINE BLOW CHECKS WHAT ITERATION WE'RE CURRENTLY AT

    #IF WE'RE IN THE FIRST ITERATION, THEN THE TETA==0 CONSTRAINT SHOULD BE INCLUDED IN THE MODEL

    #OTHERWISE WE DONT INCLUDE IT AND ADD THE OPTIMALITY CUTS TO OUR PROBLEM

    #FOR ADDING OPTIMALITY CUTS WE USE E21 AND e21

    # E21 \* X + TETA > e

    if i==1:

        NLDS1.add\_constraint\_(teta1[0]==0,'g')

    if i>1:

        for k in range(i-1):

            NLDS1.add\_constraint\_(x1[1,1] \* E11[k][0] + x1[2,1] \* E11[k][1] + x1[3,1] \* E11[k][2] + teta1[0] >= e11[k])

    # NOW WE DEFINE OUR OBJECTIVE FUNCTION

    NLDS1.minimize(sum(planting\_cost[i-1]\*x1[i,j] for i in range(1,4) for j in range(1,2))+teta1[0])

    sol\_NLDS1=NLDS1.solve()

    # PRINT SOLUTION IN THE OUTPUT

    sol\_NLDS1.display()

    # HERE WE DEFINE A DICTIONARY WHICH IS GOING TO CONTAIN SIMPLEX MULTIPLIERS OF EACH PROBLEM

    # WE WILL USE THIS DICTIONARY LATER IN THE MODEL TO FORMULATE OPTIMALITY CUTS

    Pi\_NLDS2={}

    # NOW WE SOLVE NLDS(2,1) NLDS(2,2) AND NLDS(2,3)

    for s in range(1,4):

        NLDS2=Model(name='NLDS2{}'.format(s))

        #variables

        x2=NLDS2.continuous\_var\_matrix(range(1,4),range(1,3),name='x',key\_format='%s')

        teta2=NLDS2.continuous\_var\_list(range(1,4),lb=-inf,name='teta2',key\_format='%s')

        y2=NLDS2.continuous\_var\_cube(range(1,3),range(1,4),range(2,4),name='y',key\_format='%s')

        w2=NLDS2.continuous\_var\_cube(range(1,5),range(1,4),range(2,4),name='w',key\_format='%s')

        #constraints

        NLDS2.add\_constraint\_(-x2[1,2]-x2[2,2]-x2[3,2]>=-500,'a')

        NLDS2.add\_constraint\_(sol\_NLDS1[x1[1,1]] \* yields[1][s-1] + y2[1,s,2] - w2[1,s,2] >= 200,'b')

        NLDS2.add\_constraint\_(sol\_NLDS1[x1[2,1]] \* yields[2][s-1] + y2[2,s,2] - w2[2,s,2] >= 240,'c')

        NLDS2.add\_constraint\_(sol\_NLDS1[x1[3,1]] \* yields[3][s-1] - w2[3,s,2] - w2[4,s,2] >= 0,'d')

        NLDS2.add\_constraint\_(-w2[3,s,2]>=-6000,'e')

        # sugar beets cannot be planted two successive years on the same field

        NLDS2.add\_constraint\_(sol\_NLDS1[x1[1,1]] + sol\_NLDS1[x1[2,1]] - x2[3,2]>=0,'f')

        # LIKE BEFORE IF WE ARE IN FIRST ITERATION WE ADD TETA=0 CONSTRAINT

        #OTHERWISE WE ADD OPT CUTS

        if i==1:

                NLDS2.add\_constraint\_(teta2[s-1]==0,'g')

        if i>1:

            for k in range(i-1):

                NLDS2.add\_constraint\_(E21[k][0] \* x2[1,2] + E21[k][1] \* x2[2,2] + E21[k][2] \* x2[3,2] + teta2[s-1] >= e21[k])

        #objective function

        NLDS2.minimize(150\*x2[1,2]+230\*x2[2,2]+260\*x2[3,2]+sum(buying\_price[i-1]\*y2[i,s,2] for i in range(1,3))+

                    sum(selling\_price[j-1]\*w2[j,s,2] for j in range(1,5))+teta2[s-1])

        sol\_NLDS2=NLDS2.solve()

        sol\_NLDS2.display()

        # HERE WE CALCULATE SIMPLEX MULTIPLIERS FOR EACH CONSTRAINT AND APPEND IT TO A LIST

        Pi=[]

        Pi.append(NLDS2.dual\_values(NLDS2.find\_matching\_linear\_constraints('a'))[0])

        Pi.append(NLDS2.dual\_values(NLDS2.find\_matching\_linear\_constraints('b'))[0])

        Pi.append(NLDS2.dual\_values(NLDS2.find\_matching\_linear\_constraints('c'))[0])

        Pi.append(NLDS2.dual\_values(NLDS2.find\_matching\_linear\_constraints('d'))[0])

        Pi.append(NLDS2.dual\_values(NLDS2.find\_matching\_linear\_constraints('e'))[0])

        Pi.append(NLDS2.dual\_values(NLDS2.find\_matching\_linear\_constraints('f'))[0])

        # NOW WE ADD THE RESULTING LIST TO A DICTIONARY

        # DICTIONARY KEY IS THE NAME OF THE MODEL AND WE USE THIS DICTIONARY LATER TO FORMULATE CUTS

        Pi\_NLDS2['NLDS2{}'.format(s)]=Pi

    print('Pi multipliers for NLDS2',Pi\_NLDS2)

    # NOW WE FORMULATE AND SOLVE NLDS(3,1) NLDS(3,2) ...  NLDS(3,9)

    Pi\_NLDS3={}

    for s in range(1,10):

        NLDS3=Model(name='NLDS3{}'.format(s))

        #variables

        teta3=NLDS3.continuous\_var\_list(range(1,10),lb=-inf,name='teta2',key\_format='%s')

        y3=NLDS3.continuous\_var\_cube(range(1,3),range(1,10),range(2,4),name='y',key\_format='%s')

        w3=NLDS3.continuous\_var\_cube(range(1,5),range(1,10),range(2,4),name='w',key\_format='%s')

        #constraints

        NLDS3.add\_constraint\_(sol\_NLDS2[x2[1,2]] \* yields[1][s-1] + y3[1,s,3] - w3[1,s,2] >= 200,'a')

        NLDS3.add\_constraint\_(sol\_NLDS2[x2[2,2]] \* yields[2][s-1] + y3[2,s,3] - w3[2,s,3] >= 240,'b')

        NLDS3.add\_constraint\_(sol\_NLDS2[x2[3,2]] \* yields[3][s-1] - w3[3,s,3] - w3[4,s,3] >= 0,'c')

        NLDS3.add\_constraint\_(-w3[3,s,3]>=-6000,'d')

        NLDS3.add\_constraint\_(teta3[s-1]==0)

        #objective function

        NLDS3.minimize(sum(buying\_price[i-1]\*y3[i,s,3] for i in range(1,3))+

                    sum(selling\_price[j-1]\*w3[j,s,3] for j in range(1,5))+teta3[s-1])

        sol\_NLDS3=NLDS3.solve()

        sol\_NLDS3.display()

        Pi=[]

        Pi.append(NLDS3.dual\_values(NLDS3.find\_matching\_linear\_constraints('a'))[0])

        Pi.append(NLDS3.dual\_values(NLDS3.find\_matching\_linear\_constraints('b'))[0])

        Pi.append(NLDS3.dual\_values(NLDS3.find\_matching\_linear\_constraints('c'))[0])

        Pi.append(NLDS3.dual\_values(NLDS3.find\_matching\_linear\_constraints('d'))[0])

        Pi\_NLDS3['NLDS3{}'.format(s)]=Pi

    print('Pi multipliers for NLDS3',Pi\_NLDS3)

    print(colored('\n DIR:BACKWARD \n','red'))

    # DIR CHANGES TO BACKWARD

    # HERE WE CALCULATE E21 AND e21

    #COEFFICENT OF X VARIABLES IN THE MODEL FOR THE FIRST SCENARIO (HIGH precipitation)

    T21=np.array([[3,0,0],[0,3.6,0],[0,0,24],[0,0,0]]) # COEFFICENT OF X VARIABLES IN THE MODEL FOR THE FIRST SCENARIO (HIGH precipitation)

    T22=np.array([[2.5,0,0],[0,3,0],[0,0,20],[0,0,0]]) # COEFFICENT OF X VARIABLES IN THE MODEL FOR THE SECOND SCENARIO (AVERAGE precipitation)

    T23=np.array([[2,0,0],[0,2.4,0],[0,0,16],[0,0,0]]) # COEFFICENT OF X VARIABLES IN THE MODEL FOR THE THIRD SCENARIO (LOW precipitation)

    # HERE WE EXTRACT SIMPLEX MULTIPLIER FROM THE DICTIONARY

    Pi31=np.array(Pi\_NLDS3['NLDS31']).reshape(1,4)

    Pi32=np.array(Pi\_NLDS3['NLDS32']).reshape(1,4)

    Pi33=np.array(Pi\_NLDS3['NLDS33']).reshape(1,4)

    # CALCULATE E21

    E21[i-1]=(1/3)\*np.matmul(Pi31,T21)+(1/3)\*np.matmul(Pi32,T22)+(1/3)\*np.matmul(Pi33,T23)

    print(colored('E21','red'))

    print(colored(E21[i-1],'red'))

    #E22 and E23 will be the same as E21

    # DEFINE h (RIGHT HAND SIDE OF OUR MAIN CONSTRAINTS)

    h31=np.array([200,240,0,-6000]).reshape(4,1)

    h32=h31

    h33=h31

    # CALCULATE e21

    e21[i-1]=(1/3)\*np.matmul(Pi31,h31)+(1/3)\*np.matmul(Pi32,h32)+(1/3)\*np.matmul(Pi33,h33)

    print(colored('e21','red'))

    print(colored(e21[i-1],'red'))

    # NOW WE NEED TO CHECK IF WE SHOULD ADD THE CUT OR NOT

    # FOR INSTANCE, IF e21 - E21 \* X21 >=TETA2 THEN WE SHOULD ADD THE CUT TO NLDS(2,1)

    x21=np.array([sol\_NLDS2[x2[1,2]] ,sol\_NLDS2[x2[2,2]] ,

    sol\_NLDS2[x2[3,2]]]).reshape(3,1)

    if e21[i-1]-np.matmul(E21[i-1],x21)>=sol\_NLDS2[teta2[0]]:

        print(colored('condition does not hold - we add a cut to NLDS21 problem','red'))

    else:

        print(colore('no cut needed go to t-1','red'))

    # WE CONTINUE MOVING BACKWARD

    # WE ADD THE CUT TO NLDS(2,1) NLDS(2,2) NLDS(2,3) AND FIND THE SOLUTION

    for s in range(1,4):

        NLDS2=Model(name='NLDS2{}'.format(s))

        #variables

        x2=NLDS2.continuous\_var\_matrix(range(1,4),range(1,3),name='x',key\_format='%s')

        teta2=NLDS2.continuous\_var\_list(range(1,4),lb=-inf,name='teta2',key\_format='%s')

        y2=NLDS2.continuous\_var\_cube(range(1,3),range(1,4),range(2,4),name='y',key\_format='%s')

        w2=NLDS2.continuous\_var\_cube(range(1,5),range(1,4),range(2,4),name='w',key\_format='%s')

        #constraints

        NLDS2.add\_constraint\_(-x2[1,2]-x2[2,2]-x2[3,2]>=-500,'a')

        NLDS2.add\_constraint\_(sol\_NLDS1[x1[1,1]] \* yields[1][s-1] + y2[1,s,2] - w2[1,s,2] >= 200,'b')

        NLDS2.add\_constraint\_(sol\_NLDS1[x1[2,1]] \* yields[2][s-1] + y2[2,s,2] - w2[2,s,2] >= 240,'c')

        NLDS2.add\_constraint\_(sol\_NLDS1[x1[3,1]] \* yields[3][s-1] - w2[3,s,2] - w2[4,s,2] >= 0,'d')

        NLDS2.add\_constraint\_(-w2[3,s,2]>=-6000,'e')

        NLDS2.add\_constraint\_(sol\_NLDS1[x1[1,1]] + sol\_NLDS1[x1[2,1]] - x2[3,2]>=0,'f')

        # THIS FOR LOOP ADDS OPT CUTS TO THE PROBLEM

        for x in range(i):

                NLDS2.add\_constraint\_(E21[x][0] \* x2[1,2] + E21[x][1] \* x2[2,2] + E21[x][2] \* x2[3,2] + teta2[s-1] >= e21[x])

        #objective function

        NLDS2.minimize(150\*x2[1,2]+230\*x2[2,2]+260\*x2[3,2]+sum(buying\_price[i-1]\*y2[i,s,2] for i in range(1,3))+

                    sum(selling\_price[j-1]\*w2[j,s,2] for j in range(1,5))+teta2[s-1])

        sol\_NLDS2=NLDS2.solve()

        sol\_NLDS2.display()

        # CALCULATE SIMPLEX MULTIPLIERS

        Pi=[]

        Pi.append(NLDS2.dual\_values(NLDS2.find\_matching\_linear\_constraints('a'))[0])

        Pi.append(NLDS2.dual\_values(NLDS2.find\_matching\_linear\_constraints('b'))[0])

        Pi.append(NLDS2.dual\_values(NLDS2.find\_matching\_linear\_constraints('c'))[0])

        Pi.append(NLDS2.dual\_values(NLDS2.find\_matching\_linear\_constraints('d'))[0])

        Pi.append(NLDS2.dual\_values(NLDS2.find\_matching\_linear\_constraints('e'))[0])

        Pi.append(NLDS2.dual\_values(NLDS2.find\_matching\_linear\_constraints('f'))[0])

        Pi\_NLDS2['NLDS2{}'.format(s)]=Pi

    #t-1=1

    # LIKE BEFORE WE CALCULATE E11 AND e11 AND USE THESE VALUES TO FORMULATE NLDS(1) CUT

    T11=np.array([[3,0,0],[0,3.6,0],[0,0,24],[0,0,0],[0,0,0],[0,0,0]])

    T12=np.array([[2.5,0,0],[0,3,0],[0,0,20],[0,0,0],[0,0,0],[0,0,0]])

    T13=np.array([[2,0,0],[0,2.4,0],[0,0,16],[0,0,0],[0,0,0],[0,0,0]])

    Pi21=np.array(Pi\_NLDS2['NLDS21']).reshape(1,6)

    Pi22=np.array(Pi\_NLDS2['NLDS22']).reshape(1,6)

    Pi23=np.array(Pi\_NLDS2['NLDS23']).reshape(1,6)

    # CALCULATE E11

    E11[i-1]=(1/3)\*np.matmul(Pi21,T11)+(1/3)\*np.matmul(Pi22,T12)+(1/3)\*np.matmul(Pi23,T13)

    print(colored('E11','red'))

    print(colored(E11[i-1],'red'))

    #E22 and E23 will be the same as E21

    h21=np.array([-500,200,240,0,-6000,0]).reshape(6,1)

    h22=h21

    h23=h21

    # CALCULATE e11

    e11[i-1]=(1/3)\*np.matmul(Pi21,h21)+(1/3)\*np.matmul(Pi22,h22)+(1/3)\*np.matmul(Pi23,h23)

    print(colored('e11','red'))

    print(colored(e11[i-1],'red'))

    x11=np.array([sol\_NLDS1[x1[1,1]] ,sol\_NLDS1[x1[2,1]] ,

    sol\_NLDS1[x1[3,1]]]).reshape(3,1)

    # CHECK IF IT'S NEEDED TO ADD THE CUT OR NOT!

    if e21[i-1]-np.matmul(E11[i-1],x11)>=sol\_NLDS1.get\_value\_list(teta1)[0]:

        print(colored('condition does not hold - we add a cut to NLDS1 problem','red'))

    else:

        print(colored('no cut needed - optimal solution achieved!','red'))

        break

    # IF THE CONDITION ABOVE HOLDS, THEN WE ADD THE OPT CUT TO THE NLDS(1) PROBLEM

    # CUT = E11 \* X + TETA1 >= e11

    NLDS1.add\_constraint\_(x1[1,1] \* E11[i-1][0] + x1[2,1] \* E11[i-1][1] + x1[3,1] \* E11[i-1][2] + teta1[0] >= e11[i-1])

    NLDS1.remove\_constraint('g')

    sol\_NLDS1=NLDS1.solve()

    sol\_NLDS1.display()

    # ADD COUNTER

    i+=1

print(colored('\n OPTIMAL SOLUTION','yellow'))

print(colored('x11 (wheat planted in first period)','green'))

print(sol\_NLDS1[x1[1,1]])

print(colored('x21 (corn planted in first period','green'))

print(sol\_NLDS1[x1[2,1]])

print(colored('x31 (sugarbeet planted in first period','green'))

print(sol\_NLDS1[x1[3,1]])

print(colored('x12 (wheat planted in second period)','green'))

print(sol\_NLDS2[x2[1,2]])

print(colored('x22 (corn planted in second period)','green'))

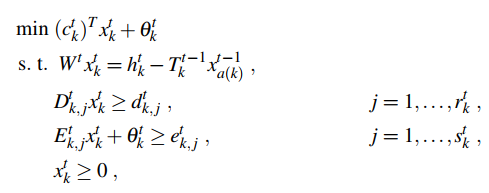
print(sol\_NLDS2[x2[2,2]])

print(colored('x32 (sugarbeet planted in second period)','green'))

print(sol\_NLDS2[x2[3,2]])

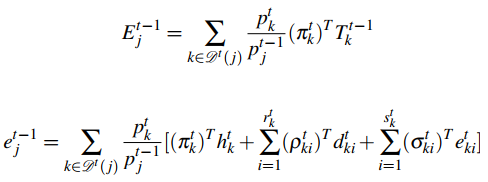
پس از تعریف پارامتر­های مدل و فراخوانی کتابخانه­های مدنظر برای مدل­سازی وارد الگوریتم می­شویم. لازم به ذکر است که به منظور مدلسازی این بخش نیز مجددا از کتابخانه­ی docplex استفاده نموده­ایم.

وارد الگوریتم می­شویم. جهت حرکت در شروع الگوریتم Forward است و ابتدا باید مسئله­ی را حل بنماییم. در ادامه مسائل و را نیز حل کرده و جواب هرکدام را در خروجی مسئله چاپ می­کنیم.



پس از هر مسئله نیز مقادیر ضرایب سیمپلکس متناظر با هر مسئله را نیز در متغیر قرار می­دهیم. این مقادیر در مسیر برگشت الگوریتم برای محاسبه­ی برش­های بهینگی مورد استفاده قرار می­گیرند.

در حرکت BACK=DIR ،ابتدا مقادیر E و e را برای هر زیرمسئله محاسبه کرده و پس از بررسی شرط اعمال برش، در صورت نیاز، به برش مدنظر به مسئله­ی مدنظر اضافه می­شود. با توجه به جنس متغیرها و مدل، میتوان نشان داد که هیچگاه نیازی به برشهای شدنی در این مسئله نخواهیم داشت) زیرا همواره میتوان کمبود یا مازاد را خریداری کرد یا به فروش رساند.



**\*\* توضیحات کد به صورت جامع و کامل در فایل نرم­افزار قرار داده شده است و برای جلوگیری از شلوغی متن، از توضیح مجدد کد صرف نظر می­کنیم. خروجی جواب هر دو الگوریتم نیز در فایل نرم افزار قابل مشاهده است.**

# مقایسه­ی نتایج دو الگوریتم و جمع بندی

بر خلاف انتظار جواب دو مسئله یکسان نشد. علت این موضوع می­تواند خطاهای محاسباتی و یا خطاهایی در بخش محاسبه­ی برش­های کد باشد. الگوریتم تا دور سوم به درستی و صحیح اجرا می­شود اما در دور چهارم به نظر می­رسد که برای تولید برش با مشکل مواجه می­شود. به طور کلی الگوریتم تجزیه­ی تو در تو دارای شروط پیچیده بسیاری است که با تغییر یک پارامتر یا المان در مسئله ممکن است به جواب متفاوتی دست پیدا کنیم. در این پروژه سعی کردیم تا جای ممکن و با وجود کمبود وقت شدید در نظر گرفته شده، الگوریتم تو در تو و نحوه­ی تولید جواب آن را تا حد ممکن با دقت بالا مدل کنیم. هر چند ممکن است در بعضی از قسمت­ها از مقادیر کوچکی صرف نظر کرده باشیم.