به نام خدا

تمرین takehome

برنامه‌ریزی تصادفی

احمد امامی

۹۹۲۰۷۵۲۱

فهرست مطالب

** 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:

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

۲. 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 ۴ کتاب توضیح داده شده است، کشاورزی میخواهد سه محصول گندم، ذرت و چغندرقند را در زمینی به مساحت ۵۰۰ 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 ۱۸ 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=1 مرحله‌ی اول	T=2 مرحله‌ی دوم	T=3 مرحله‌ی سوم
کاشت محصول (سال اول)	تصمیم در ارتباط با برداشت سال اول و کاشت محصول برای سال دوم	تصمیم در ارتباط با برداشت سال دوم

با توجه به توضیحات مطرح شده مدل مسئله را می‌نویسیم:

پارامترهای مدل

میزان برداشت محصول i در سناریوی s	$Yield_{is}$
احتمال رخداد سناریوی s	P_s
هزینه‌ی کاشت هر هکتار از محصول i	$Plant_i$
قیمت فروش محصول i (برای چغندر قند قیمت بالاتر را $sell_3$ و قیمت پایین‌تر را با $sell_4$ نمایش می‌دهیم	$Sell_i$
قیمت خرید هر تن محصول i	Buy_i
حداقل کشت مورد نیاز از محصول i	$Required_i$

متغیرهای تصمیم مدل

میزان زمین اختصاص داده شده به کشت برای محصول i		X_{it}
میزان خرید گندم مرحله‌ی t و تحت سناریوی s	Y_{1st}	Y_{ist}
میزان خرید ذرت مرحله‌ی t و تحت سناریوی s	Y_{2st}	
میزان فروش گندم در مرحله‌ی t و تحت سناریوی s	W_{1st}	W_{ist}
میزان فروش ذرت در مرحله‌ی t و تحت سناریوی s	W_{2st}	
میزان فروش چغندرقند با قیمت بالا در مرحله‌ی t و تحت سناریوی s	W_{3st}	
میزان فروش چغندرقند با قیمت پایین در مرحله‌ی t و تحت سناریوی s	W_{4st}	

تابع هدف

$$Min\ z = \sum_{i=1}^{3}\sum_{t=1}^{2} plant_i * X_{it} + \sum_{s}\sum_{t=2}^{3} P_s * (\sum_{i=1}^{2} Buy_i * Y_{ist} - \sum_{i=1}^{4} Sell_i * W_{ist})$$

محدودیت‌ها

$$\sum_{i=1}^{3} X_{it} \leq 500$$

$Yield_{is} * X_{it} + Y_{is(t+1)} - W_{is(t+1)} \geq Required_i$

$Yield_{3s} * X_{3t} - W_{3s(t+1)} - W_{4s(t+1)} \geq 0$

$W_{3st} \leq 6000$

$X_{32} \leq X_{11} + X_{21}$

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

همان‌طور که می‌دانیم پارامتر غیرقطعی که سناریوها را به کمک آن تعریف می‌کنیم $Yield_{is}$ 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 ۲.۵ ، ۳ و ۲۰ 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:

$$Yield_{is} \sim N(\mu, (0.2\mu)^2\,)$$

$$Yield_{1s} \sim N(2.5\,, 0.5^2)$$

$$Yield_{2s} \sim N(3\,, 0.6^2)$$

$$Yield_{3s} \sim N(20, 4^2)$$

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 $\mu \pm 3\sigma$ را به ۱۰۰ 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 ۱۰۰ 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 ۱۰ عدد نوشته شده است.

```python
# 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 ۱۰۰ 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 ۹۹*۹۹ 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 ۰.۰۶۵ الگوریتم متوقف می‌شود.

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

حل مسئله‌ی کشاورز به کمک سناریوهای کاهش یافته

```
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_forma
t='%s')
w=FARMER.continuous_var_cube(range(1,5),range(1,11),range(2,4),name='w',key_forma
t='%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)
```

```python
# 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)

```
solution for: farmer problem using reduced scenario
objective: -627874.713
x11 = 133.779
x12 = 133.779
x21 = 113.314
x22 = 119.127
x31 = 252.906
x32 = 247.094
y212 = 36.714
y213 = 26.286
w122 = 36.120
w123 = 36.120
w132 = 72.241
w133 = 72.241
w142 = 108.361
w143 = 108.361
w152 = 136.455
w153 = 136.455
w162 = 152.508
w163 = 152.508
w172 = 168.562
w173 = 168.562
w182 = 184.615
w183 = 184.615
w192 = 216.722
w193 = 216.722
w1102 = 256.856
w1103 = 256.856
w223 = 12.311
w232 = 36.714
w233 = 50.908
w242 = 73.428
w243 = 89.505
w252 = 101.983
w253 = 119.525
w262 = 118.300
w263 = 136.680
```

```
w272 = 134.618
w273 = 153.834
w282 = 150.935
w283 = 170.988
w292 = 183.569
w293 = 205.297
w2102 = 224.363
w2103 = 248.183
w312 = 3024.759
w313 = 2955.241
w322 = 3571.037
w323 = 3488.963
w332 = 4117.314
w333 = 4022.686
w342 = 4663.592
w343 = 4556.408
w352 = 5088.475
w353 = 4971.525
w362 = 5331.265
w363 = 5208.735
w372 = 5574.055
w373 = 5445.945
w382 = 5816.845
w383 = 5683.155
w392 = 6000.000
w393 = 6000.000
w3102 = 6000.000
w3103 = 6000.000
w492 = 302.425
w493 = 157.575
w4102 = 909.400
w4103 = 750.600
```

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 ۰.۳، متوسط ۰.۴ و پرباران ۰.۳ را اختصاص می‌دهیم.

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

$$NLDS(1)$$

$$\min 150x_{11} + 230x_{21} + 260x_{31} + \theta^1$$

$$x_{11} + x_{21} + x_{31} \le 500$$

$$\theta^1 = 0$$

$$NLDS(2,1)$$

$$\min 150x_{12} + 230x_{22} + 260x_{32} + \sum_{i=1}^{2} Buy_i * Y_{i12} + \sum_{i=1}^{4} Sell_i * W_{i12} + \theta_1^2$$

$$x_{12} + x_{22} + x_{32} \le 500$$

$$0 + Y_{112} - W_{112} \ge 200$$

$$0 + Y_{212} - W_{212} \ge 240$$

$$W_{312} + W_{412} \le 0$$

$$W_{312} \leq 6000$$

$$x_{32} \leq x_{11} + x_{21}$$

$$\theta_1^2 = 0$$

$NLDS(2,2)$

$$\min 150x_{12} + 230x_{22} + 260x_{32} + \sum_{i=1}^{2} Buy_i * Y_{i22} + \sum_{i=1}^{4} Sell * W_{i22} + \theta_2^2$$

$$x_{12} + x_{22} + x_{32} \leq 500$$

$$0 + Y_{122} - W_{122} \geq 200$$

$$0 + Y_{222} - W_{222} \geq 240$$

$$W_{322} + W_{422} \leq 0$$

$$W_{322} \leq 6000$$

$$x_{32} \leq x_{11} + x_{21}$$

$$\theta_2^2 = 0$$

$NLDS(2,3)$

$$\min 150x_{12} + 230x_{22} + 260x_{32} + \sum_{i=1}^{2} Buy_i * Y_{i32} + \sum_{i=1}^{4} Sell * W_{i32} + \theta_3^2$$

$$x_{12} + x_{22} + x_{32} \leq 500$$

$$0 + Y_{132} - W_{132} \geq 200$$

$$0 + Y_{232} - W_{232} \geq 240$$

$$W_{332} + W_{432} \leq 0$$

$$W_{332} \leq 6000$$

$$x_{32} \leq x_{11} + x_{21}$$

$$\theta_3^2 = 0$$

$NLDS(3,1)$

$$\min \sum_{i=1}^{2} Buy_i * Y_{i13} + \sum_{i=1}^{4} Sell_i * W_{i13} + \theta_1^3$$

$$0 + Y_{113} - W_{113} \geq 200$$

$$0 + Y_{213} - W_{213} \geq 240$$

$$W_{313} + W_{413} \leq 0$$

$$W_{313} \leq 6000$$

$$\theta_1^3 = 0$$

برای $NLDS(3,2)$ تا $LDS(3,9)$ 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_fo
rmat='%s')
        w2=NLDS2.continuous_var_cube(range(1,5),range(1,4),range(2,4),name='w',key_fo
rmat='%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_f
ormat='%s')
        w3=NLDS3.continuous_var_cube(range(1,5),range(1,10),range(2,4),name='w',key_f
ormat='%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_fo
rmat='%s')
        w2=NLDS2.continuous_var_cube(range(1,5),range(1,4),range(2,4),name='w',key_fo
rmat='%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 $NLDS(1)$ را حل بنماییم. در ادامه مسائل $NLDS(2,s)$ و $NLDS(3,s)$ را نیز حل کرده و جواب هرکدام را در خروجی مسئله چاپ می‌کنیم.

$$
\begin{aligned}
\min\ & (c_k^t)^T x_k^t + \theta_k^t \\
\text{s. t. }\ & W^t x_k^t = h_k^t - T_k^{t-1} x_{a(k)}^{t-1}\ , \\
& D_{k,j}^t x_k^t \geq d_{k,j}^t\ , && j = 1,\ldots,r_k^t\ , \\
& E_{k,j}^t x_k^t + \theta_k^t \geq e_{k,j}^t\ , && j = 1,\ldots,s_k^t\ , \\
& x_k^t \geq 0\ ,
\end{aligned}
$$

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.

$$E_j^{t-1} = \sum_{k \in \mathscr{D}^t(j)} \frac{p_k^t}{p_j^{t-1}} (\pi_k^t)^T T_k^{t-1}$$

$$e_j^{t-1} = \sum_{k \in \mathscr{D}^t(j)} \frac{p_k^t}{p_j^{t-1}} [(\pi_k^t)^T h_k^t + \sum_{i=1}^{r_k^t} (\rho_{ki}^t)^T d_{ki}^t + \sum_{i=1}^{s_k^t} (\sigma_{ki}^t)^T e_{ki}^t]$$

**** 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.