



PROJECT REPORT

OPERATION RESEARCH 1
(LINEAR PROGRAMMING)

Jasmeet Singh

#V00843228

Table of Contents

Introduction	3
Code Snapshots (Primal and Dual Method).....	4
Code Snapshots (Latex File Generator, Python Code).....	13
Code Snapshots (Criss Cross Method)	18
Code Snapshots (Test file with output on terminal)	20
Latex PDF.....	24

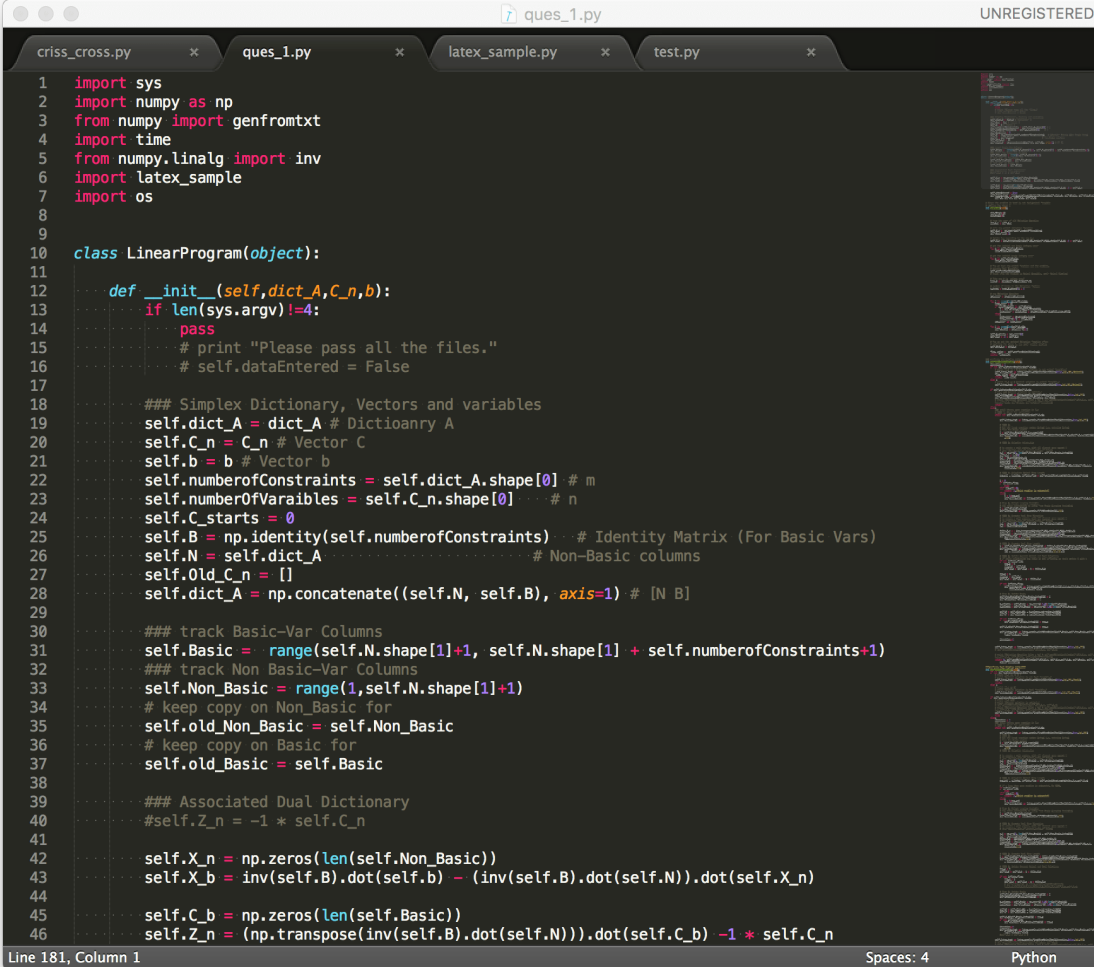
Introduction

This report contains the code snapshots of the implementation of the Primal Simplex, Dual Simplex method, Criss Cross Method for linear programming. It also includes the snapshot for the test file and the output file snapshot corresponding to the test. All the implementation has been done in python programming language. I have used numpy framework for storing my vectors, matrices, computing inverse of matrices and other basic mathematic computations.

Since it was not clear from the project assignment pdf, what to include in the project report, I have included the screenshots corresponding to the code I have written. I have uploaded my code on the github along with the matrices csv that I have used for testing and development purposes. The code represents my work only. ***I have also uploaded the PDF file generated by Latex code at the end of this report i.e. you will find the latex PDF attached at the end of this report.*** Since, it was not clear in the description, whether we have to make the latex file for Primal or Dual method. I have created tex file and pdf files for both the solvers. The code generates the latex file depending upon the input to the solvers.

All the latex files, Code files, test files, csv files are on the github :
https://github.com/jasmeet17/lp_project

Code Snapshots (Primal and Dual Method)



The screenshot shows a code editor window titled 'ques_1.py' with a tab bar containing 'criss_cross.py', 'ques_1.py', 'latex_sample.py', and 'test.py'. The status bar at the bottom indicates 'Line 181, Column 1', 'Spaces: 4', and 'Python'. The code defines a class 'LinearProgram' with an '__init__' method that initializes various attributes for a linear programming problem, including dictionaries for constraints and variables, and vectors for the objective function and right-hand side.

```
1 import sys
2 import numpy as np
3 from numpy import genfromtxt
4 import time
5 from numpy.linalg import inv
6 import latex_sample
7 import os
8
9
10 class LinearProgram(object):
11
12     def __init__(self, dict_A, C_n, b):
13         if len(sys.argv) != 4:
14             pass
15         # print "Please pass all the files."
16         # self.dataEntered = False
17
18         ### Simplex Dictionary, Vectors and variables
19         self.dict_A = dict_A # Dictioanry A
20         self.C_n = C_n # Vector C
21         self.b = b # Vector b
22         self.numberOfConstraints = self.dict_A.shape[0] # m
23         self.numberOfVariables = self.C_n.shape[0] # n
24         self.C_starts = 0
25         self.B = np.identity(self.numberOfConstraints) # Identity Matrix (For Basic Vars)
26         self.N = self.dict_A # Non-Basic columns
27         self.Old_C_n = []
28         self.dict_A = np.concatenate((self.N, self.B), axis=1) # [N B]
29
30         ### track Basic-Var Columns
31         self.Basic = range(self.N.shape[1]+1, self.N.shape[1] + self.numberOfConstraints+1)
32         ### track Non Basic-Var Columns
33         self.Non_Basic = range(1, self.N.shape[1]+1)
34         # keep copy on Non_Basic for
35         self.old_Non_Basic = self.Non_Basic
36         # keep copy on Basic for
37         self.old_Basic = self.Basic
38
39         ### Associated Dual Dictionary
40         self.Z_n = -1 * self.C_n
41
42         self.X_n = np.zeros(len(self.Non_Basic))
43         self.X_b = inv(self.B).dot(self.b) - (inv(self.B).dot(self.N)).dot(self.X_n)
44
45         self.C_b = np.zeros(len(self.Basic))
46         self.Z_n = (np.transpose(inv(self.B).dot(self.N))).dot(self.C_b) -1 * self.C_n
```

ques_1.pyUNREGISTERED

criss_cross.pyques_1.pylatex_sample.pytest.py

```
47
48     self.dataEntered = True
49     self.latex_text = latex_sample.getInitialMatrices(self.dict_A, self.Basic, self.Non_Basic,
50                                                       self.B, self.N, self.X_b, self.Z_n)
51
52     # Since the problem in hand is not dual/primal feasible
53     # applies the phase 1
54     def phaseOne(self):
55         old_NBasic=[]
56         old_Basic=[]
57
58         # keep the copy of old Objective Fuction
59         old_C_n = self.C_n
60
61         # make new primal objective function
62         self.C_n = np.empty(self.numberofVariables)
63         self.C_n.fill(-1)
64
65         # Update Z_n according to the new C_n
66         self.Z_n = (np.transpose(inv(self.B).dot(self.N))).dot(self.C_b) -1 * self.C_n
67
68         # get the initial non basic indexes array
69         for x in self.old_Non_Basic:
70             old_NBasic.append(x)
71
72         # get the initial basic indexes array
73         for x in self.old_Basic:
74             old_Basic.append(x)
75
76         # Now we have new primal function and the problem,
77         # becomes Dual Feasible.
78         self.preformDualSimplex()
79         # print 'Now the problem is Primal Feasible, apply Primal Simplex'
80
81         # "the new A is values are:"
82         temp_A = -(inv(self.B)).dot(self.N)
83
84         # number of terms in the Objective fuction
85         n_terms = temp_A.shape[1] + 1
86
87         # new Objective Fuction
88         sum_array = np.zeros(n_terms)
89
90         for i in range(len(old_NBasic)):
91             temp_array = []
```

Line 181, Column 1Spaces: 4Python

ques_1.pyUNREGISTERED

criss_cross.pyques_1.pylatex_sample.pytest.py

```
91         temp_array = []
92         if old_NBasic[i] in self.Basic:
93             t = self.Basic.index(old_NBasic[i])
94             temp_array = old_C_n[i] * np.append([self.X_b[t]], temp_A[t])
95         else:
96             temp_array = np.zeros(n_terms)
97             temp_array[i+1] = old_C_n[i]
98             sum_array += temp_array
99
100     for i in range(len(self.C_n)):
101         self.C_n[i] = sum_array[i+1]
102
103     self.C_starts = sum_array[0]
104     self.Z_n = -1 * self.C_n
105
106     # Now we got the updated Objective function after
107     # applying the phase 1, now apply primal simplex
108     self.Old_C_n = old_C_n
109
110     flag, value = self.preformPrimalSimplex()
111     return flag, value
112
113     ### performs the Simplex method
114     def preformPrimalSimplex(self):
115         iteration = 1
116         if not self.isVectorPositive(self.X_b):
117             # print "X_b is < 0 Initial solution is not primal feasible."
118             self.latex_text += latex_sample.getInitialCondition(False, 'x', 'B', 'Primal')
119             flag, value = self.phaseOne()
120             return flag, value
121         else:
122             # print 'X_b >= 0 Initial solution is primal feasible.'
123             self.latex_text += latex_sample.getInitialCondition(True, 'x', 'B', 'Primal')
124
125         if self.isVectorPositive(self.Z_n):
126             # 'Z_n >= 0 Current solution is optimal.'
127             self.latex_text += latex_sample.firstStepPrimalDual(iteration, False, 'z', 'N')
128             self.printObjectiveFunction(self.Old_C_n, self.C_n)
129             # print "Objective Function Value : %s" % self.getObjectiveValue(self.Old_C_n, self.
130             C_n, self.X_b, self.Basic, self.numberofVariables)
131             return
132         else:
133             ### until theres some negative in Z_n
134             # STEP 1: Check for optimality
135             while not self.isVectorPositive(self.Z_n):
```

Line 181, Column 1Spaces: 4Python

ques_1.pyUNREGISTERED

criss_cross.pyques_1.pylatex_sample.pytest.py

```
136 self.latex_text += latex_sample.firstStepPrimalDual(iteration,True,'z','N')
137
138 # STEP 2:
139 # Get the least negative number Index( i.e. entering Index)
140 # from Non Basic vector
141 j = self.Non_Basic[self.Z_n.argmin()]
142 self.latex_text += latex_sample.secondStepPrimalDual(self.Z_n[self.Z_n.argmin()],j,'z',
    'j')
143
144 # STEP 3: Calculate delata_X_b
145
146 # to create a unit vector, with all element zero except 1
147 # np.eye(value,size_of_vector,index of Value)
148 e_j = np.eye(1, len(self.Non_Basic) , self.Non_Basic.index(j))
149 self.X_n = e_j[0]
150 e_j = np.transpose(e_j)
151 delta_X_b = (inv(self.B).dot(self.N)).dot(e_j)
152 delta_X_b = np.reshape(delta_X_b,(delta_X_b.shape[0],))
153 self.latex_text += latex_sample.thirdStepPrimal(3,inv(self.B).dot(self.N),e_j,
    delta_X_b,'j')
154
155 # STEP 4: Calculate Primal Step Length
156 max_val , t_index, infinte_flag = self.primalStepLength(delta_X_b,self.X_b)
157
158 t = 0
159 if infinte_flag:
160     pass
161 elif max_val<=0:
162     return -1,'Print problem is unbounded'
163 else:
164     t = 1/max_val
165 self.latex_text += latex_sample.fourthStepPrimalDual(delta_X_b,self.X_b,t,'t')
166
167 # Step 5: Select Leaving Variable
168 # max ratio corresponds to index from Basic (Leaving Variable)
169 i = self.Basic[t_index]
170 self.latex_text += latex_sample.fifthStepPrimal(i,'i')
171
172 # STEP 6: Compute Dual Step Direction
173 # to create a unit vector, with all element zero except 1
174 # np.eye(value,size_of_vector,index of Value)
175 e_i = np.eye(1, len(self.Basic) , self.Basic.index(i))
176 e_i = np.transpose(e_i)
177
178 delta_Z_n = - (np.transpose((inv(self.B)).dot(self.N))).dot(e_i)
179 delta_Z_n = np.reshape(delta_Z_n,(delta_Z_n.shape[0],))
```

Line 181, Column 1Spaces: 4Python

ques_1.pyUNREGISTERED

criss_cross.pyques_1.pylatex_sample.pytest.py

```
179 delta_Z_n = np.reshape(delta_Z_n, (delta_Z_n.shape[0],))
180 self.latex_text += latex_sample.sixthStepPrimal(6, np.transpose((inv(self.B)).dot(self.
181 N)), e_i, delta_Z_n, 'i')
182
183 # STEP 7: Compute Dual Step Length
184 s = self.Z_n[self.Non_Basic.index(j)] / delta_Z_n[self.Non_Basic.index(j)]
185 self.latex_text += latex_sample.seventhStepPrimalDual(s, self.Z_n[self.Non_Basic.index(
186 j)], delta_Z_n[self.Non_Basic.index(j)], 's', 'z', j)
187
188 # STEP 8: Update Current Primal and Dual Solutions
189 # if while calculating max ratio we get infinite; we don't update X with t
190 if not infinte_flag:
191     new_x = t
192     old_X_b = self.X_b
193     self.X_b = self.X_b - t * delta_X_b
194
195     new_z = s
196     old_Z_n = self.Z_n
197     self.Z_n = self.Z_n - s * delta_Z_n
198
199     if not infinte_flag:
200         self.latex_text += latex_sample.eightStepPrimal(j, i, t, s, old_X_b, old_Z_n, delta_X_b,
201         delta_Z_n, self.X_b, self.Z_n)
202
203 # Step 9: Update Basis
204 self.Non_Basic[self.Non_Basic.index(j)] = i
205 self.Basic[self.Basic.index(i)] = j
206
207 b_columns = self.Basic + np.array([-1.0]*len(self.Basic))
208 n_columns = self.Non_Basic + np.array([-1.0]*len(self.Non_Basic))
209
210 self.B = self.dict_A[:, b_columns.astype(np.int64)]
211 self.N = self.dict_A[:, n_columns.astype(np.int64)]
212
213 if not infinte_flag:
214     self.X_b[self.Basic.index(j)] = new_x
215
216 self.Z_n[self.Non_Basic.index(i)] = new_z
217
218 self.latex_text += latex_sample.ninthStepPrimal(self.Basic, self.Non_Basic, self.B, self.
219 N, self.X_b, self.Z_n)
220
221 iteration+=1
222
223 self.latex_text += latex_sample.firstStepPrimalDual(iteration, False, 'z', 'N')
```

Line 181, Column 1Spaces: 4Python

ques_1.pyUNREGISTERED

criss_cross.pyques_1.pylatex_sample.pytest.py

```
220         self.latex_text += latex_sample.firstStepPrimalDual(iteration,False,'z','N')
221         # self.printObjectiveFunction(self.Old_C_n, self.C_n)
222
223         # print "Objective Function Value : %s" % self.getObjectiveValue(self.Old_C_n, self.
224         C_n,self.X_b, self.Basic, self.numberofVariables)
225         return 0, self.getObjectiveValue(self.Old_C_n, self.C_n,self.X_b, self.Basic, self.n
226         umberofVariables)
227
228         """performs Dual Simplex method"""
229         def preformDualSimplex(self):
230             if not self.isVectorPositive(self.Z_n):
231                 # print "Z_n is <= 0 "
232                 # print "Initial solution is not Dual feasible."
233                 self.latex_text += latex_sample.getInitialCondition(False,'z','N','Dual')
234                 return
235             else :
236                 # print 'Z_n >= 0'
237                 # print "Initial solution is Dual feasible."
238                 self.latex_text += latex_sample.getInitialCondition(True,'z','N','Dual')
239
240             if self.isVectorPositive(self.X_b):
241                 # print 'X_b >=0'
242                 # print 'Current solution is optimal.'
243                 # self.printObjectiveFunction(self.Old_C_n, self.C_n)
244                 # print "Objective Function Value : %s" % self.getObjectiveValue(self.Old_C_n, self.
245                 C_n,self.X_b, self.Basic, self.numberofVariables)
246                 self.latex_text += latex_sample.firstStepPrimalDual(iteration,False,'x','B')
247                 pass
248             else:
249                 iteration = 1
250                 ### until theres some negative in Z_n
251                 # STEP 1: Check for optimality
252                 while not self.isVectorPositive(self.X_b):
253                     self.latex_text += latex_sample.firstStepPrimalDual(iteration,True,'x','B')
254                     # STEP 2:
255                     # Get the least negative number Index( i.e. entering Index)
256                     # from Non Basic vector
257                     i = self.Basic[self.X_b.argmin()]
258                     self.latex_text += latex_sample.secondStepPrimalDual(self.X_b[self.X_b.argmin()],i,'x',
259                     'i')
260                     # STEP 3: Calculte delata_Z_n
261                     # to create a unit vector, with all element zero except 1
262                     # np.eye(value,size_of_vector,index_of Value)
263                     e_i = np.eye(1, len(self.Basic) , self.Basic.index(i))
```

Line 181, Column 1Spaces: 4Python

ques_1.pyUNREGISTERED

criss_cross.pyques_1.pylatex_sample.pytest.py

```
261 ..... e_i = np.eye(1, len(self.Basic) , self.Basic.index(i))
262 ..... e_i = np.transpose(e_i)
263 ..... delta_Z_n = - (np.transpose(inv(self.B).dot(self.N))).dot(e_i)
264 ..... delta_Z_n = np.reshape(delta_Z_n,(delta_Z_n.shape[0],))
265 ..... self.latex_text += latex_sample.sixthStepPrimal(3,np.transpose((inv(self.B)).dot(self.
N)),e_i,delta_Z_n,'i')

266 .....
267 ..... # STEP 4: Calculate Primal Step Length
268 ..... max_val , s_index, infinte_flag = self.primalStepLength(delta_Z_n,self.Z_n)
269 .....
270 ..... # if s less than zero problem is unbounded, So STOP.
271 ..... if infinte_flag:
272 .....     pass
273 .....     elif max_val<=0:
274 .....         return -1,'Print problem is unbounded'
275 .....     else:
276 .....         s = 1/max_val
277 .....         self.latex_text += latex_sample.fourthStepPrimalDual(delta_Z_n,self.Z_n,s,'s')
278 .....
279 ..... # Step 5: Select Leaving Variable
280 ..... # max ratio corresponds to index from Basic (Leaving Variable)
281 ..... j = self.Non_Basic[s_index]
282 ..... self.latex_text += latex_sample.fifthStepPrimal(j,'j')
283 .....
284 .....
285 ..... # STEP 6: Compute Dual Step Direction
286 ..... # to create a unit vector, with all element zero except 1
287 ..... # np.eye(value,size_of_vector,index_of Value)
288 .....
289 ..... e_j = np.eye(1, len(self.Non_Basic) , self.Non_Basic.index(j))
290 ..... e_j = np.transpose(e_j)
291 ..... delta_X_b = ((inv(self.B)).dot(self.N)).dot(e_j)
292 ..... delta_X_b = np.reshape(delta_X_b,(delta_X_b.shape[0],))
293 ..... self.latex_text += latex_sample.thirdStepPrimal(6,inv(self.B).dot(self.N),e_j,
delta_X_b,'j')

294 .....
295 .....
296 ..... # STEP 7: Compute Dual Step Length
297 ..... t = self.X_b[self.Basic.index(i)] / delta_X_b[self.Basic.index(i)]
298 ..... self.latex_text += latex_sample.seventhStepPrimalDual(t,self.X_b[self.Basic.index(i)]
, delta_X_b[self.Basic.index(i)],'t','x',j)

299 .....
300 ..... # STEP 8: Update Current Primal and Dual Solutions
301 ..... new_x = t
302 ..... self.X_b = self.X_b - t * delta_X_b
303 .....
```

Line 181, Column 1Spaces: 4Python

ques_1.pyUNREGISTERED

criss_cross.pyques_1.pylatex_sample.pytest.py

```
306 self.Z_n = self.Z_n - s * delta_Z_n
307 # self.latex_text += latex_sample.eightStepPrimal(
308     j,i,t,s,old_X_b,old_Z_n,delta_X_b,delta_Z_n,self.X_b,self.Z_n)
309
310 # Step 9: Update Basis
311 self.Non_Basic[self.Non_Basic.index(j)] = i
312 self.Basic[self.Basic.index(i)] = j
313
314 b_columns = self.Basic + np.array([-1.0]*len(self.Basic))
315 n_columns = self.Non_Basic + np.array([-1.0]*len(self.Non_Basic))
316
317 self.B = self.dict_A[:, b_columns.astype(np.int64)]
318 self.N = self.dict_A[:, n_columns.astype(np.int64)]
319
320 self.X_b[self.Basic.index(j)] = new_x
321 if not infinte_flag:
322     self.Z_n[self.Non_Basic.index(i)] = new_z
323
324 iteration+=1
325 self.latex_text += latex_sample.ninthStepPrimal(self.Basic,self.Non_Basic,self.B,self.
326     N,self.X_b,self.Z_n)
327 #self.Z_n = self.Z_n * 0
328 self.latex_text += latex_sample.firstStepPrimalDual(iteration,False,'x','B')
329 # self.printObjectiveFunction(self.Old_C_n, self.C_n)
330 # print "Objective Function Value : %s" % self.getObjectiveValue(self.Old_C_n, self.
331     C_n,self.X_b, self.Basic, self.numberofVariables)
332
333 return 0,self.getObjectiveValue(self.Old_C_n, self.C_n,self.X_b, self.Basic, self.n
334     umberofVariables)
335
336 ### Calculate Primal Step Length
337 ### Divide element by element (also consider 0/0 as 0)
338 ### takes the max of the resulted list and return inverse and
339 ### index corresponding to max
340 def primalStepLength(self,delta_x,delta_x_i):
341     temp_list=[]
342     infinte_index = -1
343     for i in range(delta_x_i.shape[0]):
344         if delta_x_i[i]==0:
345             if delta_x[i]==0:
346                 temp_list.append(0)
347             elif delta_x[i]<0:
348                 temp_list.append(0)
```

Line 181, Column 1Spaces: 4Python

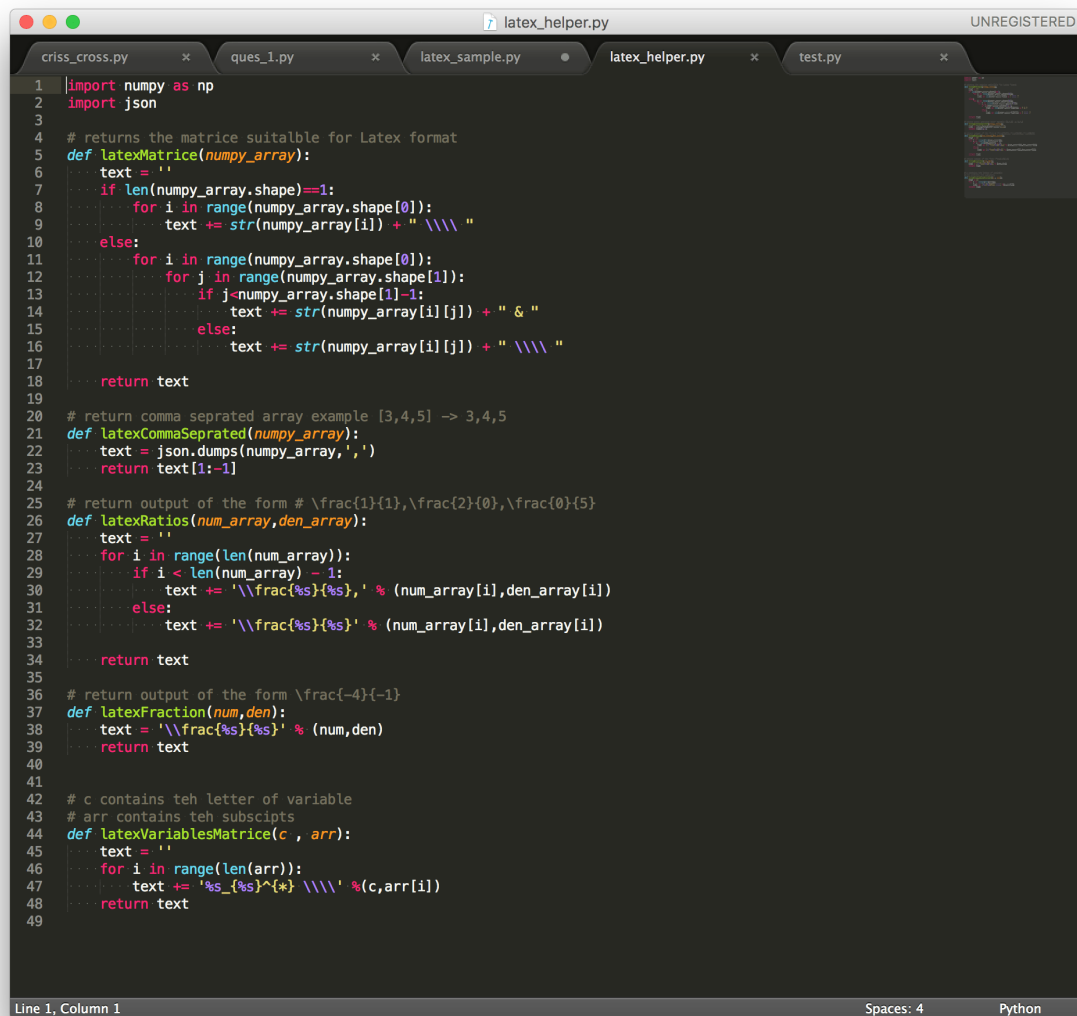
ques_1.pyUNREGISTERED

criss_cross.py x ques_1.py x latex_sample.py x test.py x

```
346         elif delta_x[i]<0:
347             temp_list.append(0)
348         elif delta_x[i]>0:
349             infinte_index = i
350         else:
351             temp_list.append(delta_x[i]/delta_x_i[i])
352
353     if infinte_index!=-1:
354         return 0.0, infinte_index , True
355
356     max_val = max(temp_list)
357
358     return float(max_val), temp_list.index(max_val), False
359
360
361     ### To check all elements in the vectors are positive
362     ### checks Greater than or equal to zero and returns True If so
363     def isVectorPositive(self,vec):
364
365     # vec_c -> objective function
366     # vec_x -> Soutlion
367     # basic -> index of basic vars
368     # n -> number of variables in objective function, initially
369     # returns the objective Function value
370     def getObjectivValue(self,old_c,vec_c,vec_x,basic_vec,n):
371
372     # Prints the Objective function in the Standard form
373     # and the Value obtained by the Solution
374     def printObjectiveFunction(self, old_c,vec_c):
375
376     def printAllVariables(self):
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
dict_A = genfromtxt(sys.argv[1], delimiter=',') # Dictioanry A
C_n = genfromtxt(sys.argv[2], delimiter=',') # Vector C
b = genfromtxt(sys.argv[3], delimiter=',') # Vector b
### create an object of Linear_Prog class
simplex = LinearProgram(dict_A,C_n,b)
# simplex.printAllVariables()
simplex.preformPrimalSimplex()
# simplex.preformDualSimplex()
```

Line 181, Column 1Spaces: 4Python

Code Snapshots (Latex File Generator, Python Code)



```
1 import numpy as np
2 import json
3
4 # returns the matrix suitable for Latex format
5 def latexMatrix(numpy_array):
6     text = ''
7     if len(numpy_array.shape)==1:
8         for i in range(numpy_array.shape[0]):
9             text += str(numpy_array[i]) + " \\\\"
10    else:
11        for i in range(numpy_array.shape[0]):
12            for j in range(numpy_array.shape[1]):
13                if j<numpy_array.shape[1]-1:
14                    text += str(numpy_array[i][j]) + " & "
15                else:
16                    text += str(numpy_array[i][j]) + " \\\\"
17    return text
18
19 # return comma separated array example [3,4,5] -> 3,4,5
20 def latexCommaSeparated(numpy_array):
21     text = json.dumps(numpy_array, ',')
22     return text[1:-1]
23
24 # return output of the form # \frac{1}{1},\frac{2}{0},\frac{0}{5}
25 def latexRatios(num_array,den_array):
26     text = ''
27     for i in range(len(num_array)):
28         if i < len(num_array) - 1:
29             text += '\\frac{%s}{%s}, ' % (num_array[i],den_array[i])
30         else:
31             text += '\\frac{%s}{%s}' % (num_array[i],den_array[i])
32     return text
33
34 # return output of the form \frac{-4}{-1}
35 def latexFraction(num,den):
36     text = '\\frac{%s}{%s}' % (num,den)
37     return text
38
39 # c contains teh letter of variable
40 # arr contains teh subscripts
41 def latexVariablesMatrice(c , arr):
42     text = ''
43     for i in range(len(arr)):
44         text += '%s_{}^{} \\\\" % (c,arr[i])
45     return text
46
47
48
49
```

Line 1, Column 1 Spaces: 4 Python

latex_sample.pyUNREGISTERED

criss_cross.pyques_1.pylatex_sample.pytest.py

```
118
119 from latex_helper import latexMatrice
120 from latex_helper import latexCommaSeprated
121 from latex_helper import latexRatios
122 from latex_helper import latexFraction
123 from latex_helper import latexVariablesMatrice
124
125 def getInitialMatrices(matrice_A, b_indices, non_b_indices, matrice_B, matrice_N, matrice_X, matrice_Z):
126     matrice_A = latexMatrice(matrice_A)
127     b_indices = latexCommaSeprated(b_indices)
128     non_b_indices = latexCommaSeprated(non_b_indices)
129     matrice_B = latexMatrice(matrice_B)
130     matrice_N = latexMatrice(matrice_N)
131     matrice_X = latexMatrice(matrice_X)
132     matrice_Z = latexMatrice(matrice_Z)
133
134     text = (initial_matrices % {'matrice_A':matrice_A, 'b_indices':b_indices, 'non_b_indices':non_b_indices,
135     return text
136
137 # return string based on whether intial condition met (i.e. true else false)
138 def getInitialCondition(bool_value,main_ch,subscript_ch,solver_type):
139     if bool_value:
140         return (initial_primal_condition_true % {'main_ch':main_ch,'subscript_ch':subscript_ch,'solver_type':
141     else:
142         return (initial_primal_condition_false % {'main_ch':main_ch,'subscript_ch':subscript_ch, 'solver_type'
143
144 # bool_value is false in case z_n has some negative; else true and algo stops
145 # iteration_no is the iteration number fo the algo
146 def firstStepPrimalDual(iteration_no,bool_value,main_ch,subscript_ch):
147     text = ''
148     if bool_value:
149         text = (step1_primal_dual_condition_true % {'iteration_no':iteration_no, 'main_ch':main_ch,'subscript
150     else:
151         text = (step1_primal_dual_condition_false % {'iteration_no':iteration_no, 'main_ch':main_ch,'subscrip
152     return text
153
154 # second step of primal
155 # argument 1 is the most negative number
156 # index is the index of the most negative in the nonbasic vector
157 def secondStepPrimalDual(negative_no, index,main_ch, var_ch):
158     text = (step2_primal_dual % {'negative_no':negative_no,'index':index, 'main_ch': main_ch, 'var_ch':var_ch
159     return text
160
161 # returns the latex string for the third Step of primal method
162 def thirdStepPrimal(step,matrice_BN, matrice_EJ,matrice_Result,subscript_ch):
163     matrice_BN = latexMatrice(matrice_BN)
164     matrice_EJ = latexMatrice(matrice_EJ)
165     matrice_Result = latexMatrice(matrice_Result)
166
167     text = (step3_primal % {'step':step,'matrice_BN':matrice_BN, 'matrice_EJ':matrice_EJ,'matrice_Result':mat
```

Line 328, Column 40Spaces: 4Python

latex_sample.pyUNREGISTERED

criss_cross.pyques_1.pylatex_sample.pytest.py

```
165 matrice_BN = latexMatrice(matrice_BN)
166 matrice_EJ = latexMatrice(matrice_EJ)
167 matrice_Result = latexMatrice(matrice_Result)
168
169 text = (step3_primal % {'step':step, 'matrice_BN':matrice_BN, 'matrice_EJ':matrice_EJ, 'matrice_Result':matrice_Result})
170 return text
171
172 step4_primal = '''\subsection{Step 4.}
173 \[
174 t = \text{Bigg}
175 \max \left\{ \frac{\text{ratio}}{s} \right\}
176 \text{Bigg}^{-1} \} = \text{value}
177 \]
178 '''
179 def fourthStepPrimalDual(num_array, den_array, value, var_ch):
180     ratio = latexRatios(num_array, den_array)
181     text = (step4_primal % {'ratio':ratio, 'value':value, 'var_ch':var_ch})
182     return text
183
184
185 step5_primal = '''\subsection{Step 5.}
186 \[
187 \text{In step 4, the ratio corresponds to basic index } \text{index}
188 \]
189 \[
190 \frac{\text{var\_ch}}{s} = \text{index}
191 \]
192 '''
193
194 def fifthStepPrimal(index, var_ch):
195     text = (step5_primal % {'index':index, 'var_ch':var_ch})
196     return text
197
198 step6_primal = '''\subsection{Step 6.}
199 \[
200 \Delta z_{\text{mathcal N}} = -(B^{-1} N)^T e_{\text{subscript\_ch}} = -\text{value}
201 \]
202 \begin{bmatrix}
203 \text{matrice\_BN} \\
204 \text{matrice\_EI} \\
205 \text{matrice\_Result}
206 \end{bmatrix}
207 = \begin{bmatrix}
208 \text{matrice\_BN} \\
209 \text{matrice\_EI} \\
210 \text{matrice\_Result}
211 \end{bmatrix}
212 '''
213
214 def sixthStepPrimal(step, matrice_BN, matrice_EI, matrice_Result, subscript_ch):
215     matrice_BN = latexMatrice(matrice_BN)
216     matrice_EI = latexMatrice(matrice_EI)
217     matrice_Result = latexMatrice(matrice_Result)
```

Line 344, Column 15Spaces: 4Python

latex_sample.pyUNREGISTERED

criss_cross.pyques_1.pylatex_sample.pytest.py

```
213 def sixthStepPrimal(step,matrice_BN,matrice_EI,matrice_Result,subscript_ch):
214     matrice_BN = latexMatrice(matrice_BN)
215     matrice_EI = latexMatrice(matrice_EI)
216     matrice_Result = latexMatrice(matrice_Result)
217
218     text = (step6_primal %{'step':step,'matrice_BN':matrice_BN,'matrice_EI':matrice_EI,'matrice_Result':matrice_Result})
219     return text
220
221 step7_primal_dual = '''\subsection{Step 7.}
222 \[
223 %(\text{var\_ch})s \ = \ \frac{1}{\Delta} \frac{\partial}{\partial (\text{main\_ch})s} \{ \Delta (\text{main\_ch})s \} \ = \ \frac{\partial}{\partial (\text{main\_ch})s} \{ \Delta (\text{main\_ch})s \}
224 \]
225 '''
226 def seventhStepPrimalDual(value,num,den,var_ch,main_ch,subscript_ch):
227     text = (step7_primal_dual %{'ratio':latexFraction(num,den),'value':value,'var_ch':var_ch,'main_ch':main_ch})
228     return text
229
230
231
232 step8_primal = '''\subsection{Step 8.}
233 \[
234 x_{(i\_index)s}^{(*)} \ = \ (i\_value)s, \quad x_{(i\_index)s} \ = \ (i\_value)s
235 \begin{bmatrix}
236 (matrice\_x\_old)s \\
237 \end{bmatrix} \ = \ (i\_value)s
238 \begin{bmatrix}
239 (matrice\_x\_delta)s \\
240 \end{bmatrix} \ = \ (i\_value)s
241 \begin{bmatrix}
242 (matrice\_x\_new)s \\
243 \end{bmatrix} \ = \ (i\_value)s
244 \]
245 \[
246 z_{(j\_index)s}^{(*)} \ = \ (j\_value)s, \quad z_{(j\_index)s} \ = \ (j\_value)s
247 \begin{bmatrix}
248 (matrice\_z\_old)s \\
249 \end{bmatrix} \ = \ (j\_value)s
250 \begin{bmatrix}
251 (matrice\_z\_delta)s \\
252 \end{bmatrix} \ = \ (j\_value)s
253 \begin{bmatrix}
254 (matrice\_z\_new)s \\
255 \end{bmatrix} \ = \ (j\_value)s
256 \]
257 \[
258 \]
259 '''
260
261 # Following parameters passed
262 # j,i,old_X_b,old_Z_n,delta_X_b,delta_Z_n,self.X_b,self.Z_n
263 def eightStepPrimal(j,i,t,s,old_X_b,old_Z_n,delta_X_b,delta_Z_n,new_X_b,new_Z_n):
264     matrice_x_old = latexMatrice(old_X_b)
```

Line 344, Column 15Spaces: 4Python

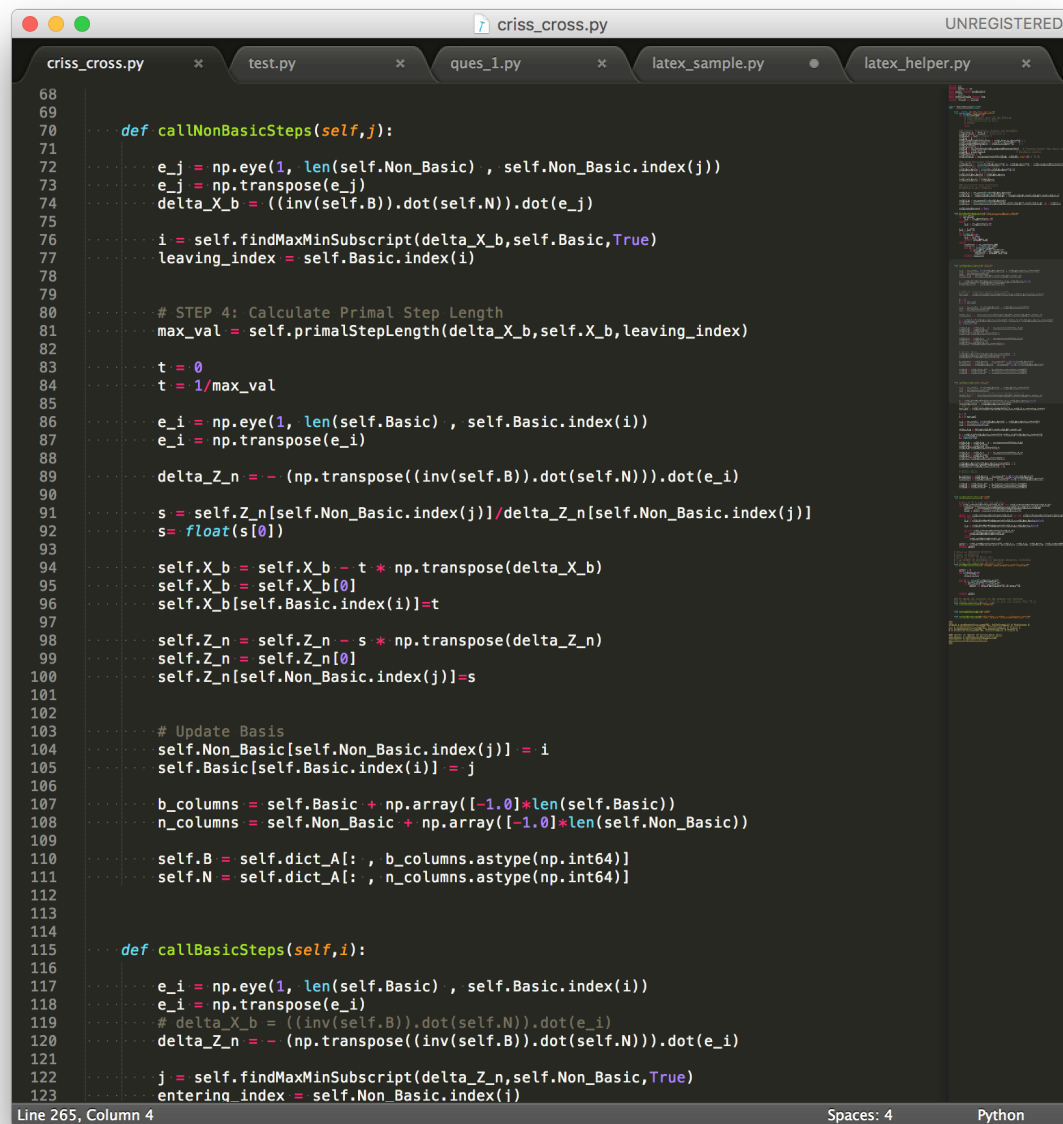
latex_sample.pyUNREGISTERED

criss_cross.pyques_1.pylatex_sample.pytest.py

```
320 matrice_var_X = latexVariablesMatrice('x',basic)
321 matrice_var_Z = latexVariablesMatrice('z',nonbasic)
322
323 basic = latexCommaSeprated(basic)
324 nonbasic = latexCommaSeprated(nonbasic)
325 matrice_B = latexMatrice(matrice_B)
326 matrice_N = latexMatrice(matrice_N)
327
328 matrice_X = latexMatrice(matrice_X)
329 matrice_Z = latexMatrice(matrice_Z)
330
331 text = (step9_primal % {'basic':basic,'nonbasic':nonbasic,'matrice_B':matrice_B,'matrice_N':matrice_N,'ma
332 return text
333
334
335 final_doc = '''\documentclass [12pt] {article}
336 \usepackage{amsmath}
337 \makeatletter
338 \renewcommand{\@secntformat}[1]{\}
339 \makeatother
340 \usepackage{url}
341 \usepackage[margin=0.8in]{geometry}
342 \pagestyle{plain}
343 \begin{document}
344 \section*{Dual Simplex Method Initial Matrices and Vector} %(latex_tex)s \end{document}
345 '''
346
347 def getWholeLatex(latex_text):
348 text = (final_doc % {'latex_tex':latex_text})
349 return text
350
351
352 objectiveFuntion = '''
353 \[
354 \zeta^{*} = %(equation)s
355 \]
356 '''
357 def latexObjectiveFuntion(vec_c,value):
358 equation = ''
359 for i in range(len(vec_c)):
360 if i != len(vec_c)-1:
361 equation += "%sx_{%s}^{*}\\ + " % (str(vec_c[i]),str(i+1))
362 else:
363 # equation += "%sx_{%s}^{*}\\ + " % (str(vec_c[i]),str(i+1))
364 equation += "%sx_{%s}^{*}\\ = " % (str(vec_c[i]),str(i+1),str(value))
365
366 text = (objectiveFuntion % {'equation':equation})
367 return text
368
369
370
371
372
```

Line 344, Column 15Spaces: 4Python

Code Snapshots (Criss Cross Method)



The image shows a screenshot of a code editor window titled "criss_cross.py" with a "UNREGISTERED" label in the top right corner. The editor has several tabs open: "criss_cross.py", "test.py", "ques_1.py", "latex_sample.py", and "latex_helper.py". The main window displays Python code for the Criss Cross Method, with line numbers 68 through 123 visible on the left. The code is as follows:

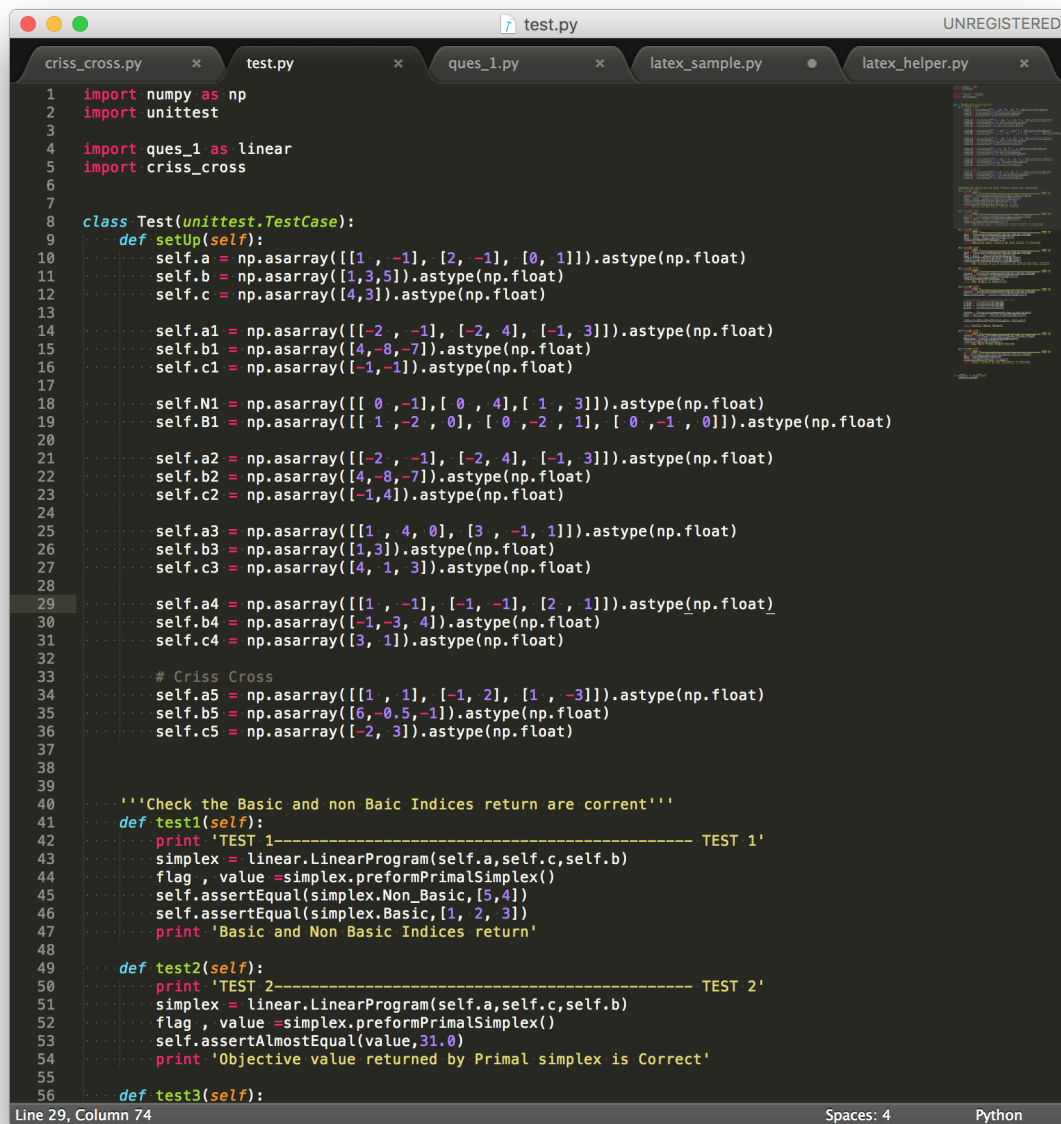
```
68
69
70 def callNonBasicSteps(self,j):
71
72     e_j = np.eye(1, len(self.Non_Basic) , self.Non_Basic.index(j))
73     e_j = np.transpose(e_j)
74     delta_X_b = ((inv(self.B)).dot(self.N)).dot(e_j)
75
76     i = self.findMaxMinSubscript(delta_X_b,self.Basic,True)
77     leaving_index = self.Basic.index(i)
78
79
80     # STEP 4: Calculate Primal Step Length
81     max_val = self.primalStepLength(delta_X_b,self.X_b,leaving_index)
82
83     t = 0
84     t = 1/max_val
85
86     e_i = np.eye(1, len(self.Basic) , self.Basic.index(i))
87     e_i = np.transpose(e_i)
88
89     delta_Z_n = - (np.transpose((inv(self.B)).dot(self.N))).dot(e_i)
90
91     s = self.Z_n[self.Non_Basic.index(j)]/delta_Z_n[self.Non_Basic.index(j)]
92     s = float(s[0])
93
94     self.X_b = self.X_b - t * np.transpose(delta_X_b)
95     self.X_b = self.X_b[0]
96     self.X_b[self.Basic.index(i)]=t
97
98     self.Z_n = self.Z_n - s * np.transpose(delta_Z_n)
99     self.Z_n = self.Z_n[0]
100     self.Z_n[self.Non_Basic.index(j)]=s
101
102
103     # Update Basis
104     self.Non_Basic[self.Non_Basic.index(j)] = i
105     self.Basic[self.Basic.index(i)] = j
106
107     b_columns = self.Basic + np.array([-1.0]*len(self.Basic))
108     n_columns = self.Non_Basic + np.array([-1.0]*len(self.Non_Basic))
109
110     self.B = self.dict_A[:, b_columns.astype(np.int64)]
111     self.N = self.dict_A[:, n_columns.astype(np.int64)]
112
113
114
115 def callBasicSteps(self,i):
116
117     e_i = np.eye(1, len(self.Basic) , self.Basic.index(i))
118     e_i = np.transpose(e_i)
119     # delta_X_b = ((inv(self.B)).dot(self.N)).dot(e_i)
120     delta_Z_n = - (np.transpose((inv(self.B)).dot(self.N))).dot(e_i)
121
122     j = self.findMaxMinSubscript(delta_Z_n,self.Non_Basic,True)
123     entering_index = self.Non_Basic.index(j)
```

The status bar at the bottom indicates "Line 265, Column 4", "Spaces: 4", and "Python".

```
criss_cross.py UNREGISTERED
criss_cross.py x test.py x ques_1.py x latex_sample.py latex_helper.py x
122         j = self.findMaxMinSubscript(delta_Z_n,self.Non_Basic,True)
123         entering_index = self.Non_Basic.index(j)
124         # STEP 4: Calculate Primal Step Length
125         max_val = self.primalStepLength(delta_Z_n,self.Z_n,entering_index)
126
127         s = 0
128         s = 1/max_val
129
130         e_j = np.eye(1, len(self.Non_Basic) , self.Non_Basic.index(j))
131         e_j = np.transpose(e_j)
132
133         delta_X_b = ((inv(self.B)).dot(self.N)).dot(e_j)
134
135         t = self.X_b[self.Basic.index(i)]/delta_X_b[self.Basic.index(i)]
136         t = float(t[0])
137
138         self.X_b = self.X_b - t * np.transpose(delta_X_b)
139         self.X_b = self.X_b[0]
140         self.X_b[self.Basic.index(i)] = t
141
142         self.Z_n = self.Z_n - s * np.transpose(delta_Z_n)
143         self.Z_n = self.Z_n[0]
144         self.Z_n[self.Non_Basic.index(j)] = s
145
146         self.Non_Basic[self.Non_Basic.index(j)] = i
147         self.Basic[self.Basic.index(i)] = j
148
149         # Update Basis
150
151         b_columns = self.Basic + np.array([-1.0]*len(self.Basic))
152         n_columns = self.Non_Basic + np.array([-1.0]*len(self.Non_Basic))
153
154         self.B = self.dict_A[:, b_columns.astype(np.int64)]
155         self.N = self.dict_A[:, n_columns.astype(np.int64)]
156
157
158
159     def preformCrissCross(self):
160
161         # Step 1 If Z_n >= 0 and X_b >= 0 Stop
162         if self.isVectorPositive(self.Z_n) and self.isVectorPositive(self.X_b):
163             simplex = linear.LinearProgram(self.dict_A,self.C_n,self.b)
164             flag , value = simplex.preformPrimalSimplex()
165
166         while not self.isVectorPositive(self.Z_n) or not self.isVectorPositive(self.X_b):
167             # step 2
168             j_z = self.findMaxMinSubscript(self.Z_n,self.Non_Basic,False)
169             i_x = self.findMaxMinSubscript(self.X_b,self.Basic,False)
170
171             if not self.isVectorPositive(self.Z_n):
172                 self.callNonBasicSteps(j_z)
173             else:
174                 self.callBasicSteps(i_x)
175
176         value = self.getObjectiveValue([],self.C_n, self.X_b, self.Basic, self.numberofVariables)
177
```

Line 265, Column 4 Spaces: 4 Python

Code Snapshots (Test file with output on terminal)



```
1 import numpy as np
2 import unittest
3
4 import ques_1 as linear
5 import criss_cross
6
7
8 class Test(unittest.TestCase):
9     def setUp(self):
10         self.a = np.asarray([[1, -1], [2, -1], [0, 1]]).astype(np.float)
11         self.b = np.asarray([1, 3, 5]).astype(np.float)
12         self.c = np.asarray([4, 3]).astype(np.float)
13
14         self.a1 = np.asarray([[2, -1], [-2, 4], [-1, 3]]).astype(np.float)
15         self.b1 = np.asarray([4, -8, -7]).astype(np.float)
16         self.c1 = np.asarray([-1, -1]).astype(np.float)
17
18         self.N1 = np.asarray([[0, -1], [0, 4], [1, 3]]).astype(np.float)
19         self.B1 = np.asarray([[1, -2, 0], [0, -2, 1], [0, -1, 0]]).astype(np.float)
20
21         self.a2 = np.asarray([[2, -1], [-2, 4], [-1, 3]]).astype(np.float)
22         self.b2 = np.asarray([4, -8, -7]).astype(np.float)
23         self.c2 = np.asarray([-1, 4]).astype(np.float)
24
25         self.a3 = np.asarray([[1, 4, 0], [3, -1, 1]]).astype(np.float)
26         self.b3 = np.asarray([1, 3]).astype(np.float)
27         self.c3 = np.asarray([4, 1, 3]).astype(np.float)
28
29         self.a4 = np.asarray([[1, -1], [-1, -1], [2, 1]]).astype(np.float)
30         self.b4 = np.asarray([-1, -3, 4]).astype(np.float)
31         self.c4 = np.asarray([3, 1]).astype(np.float)
32
33         # Criss Cross
34         self.a5 = np.asarray([[1, 1], [-1, 2], [1, -3]]).astype(np.float)
35         self.b5 = np.asarray([6, -0.5, -1]).astype(np.float)
36         self.c5 = np.asarray([-2, 3]).astype(np.float)
37
38
39
40     '''Check the Basic and non Basic Indices return are correct'''
41     def test1(self):
42         print 'TEST 1----- TEST 1'
43         simplex = linear.LinearProgram(self.a, self.c, self.b)
44         flag, value = simplex.preformPrimalSimplex()
45         self.assertEqual(simplex.Non_Basic, [5, 4])
46         self.assertEqual(simplex.Basic, [1, 2, 3])
47         print 'Basic and Non Basic Indices return'
48
49     def test2(self):
50         print 'TEST 2----- TEST 2'
51         simplex = linear.LinearProgram(self.a, self.c, self.b)
52         flag, value = simplex.preformPrimalSimplex()
53         self.assertAlmostEqual(value, 31.0)
54         print 'Objective value returned by Primal simplex is Correct'
55
56     def test3(self):
```

Line 29, Column 74 Spaces: 4 Python

test.pyUNREGISTERED

criss_cross.pytest.pyques_1.py latex_sample.py latex_helper.py

```
58 dual = linear.LinearProgram(self.a1,self.c1,self.b1)
59 flag , value =dual.preformDualSimplex()
60 self.assertEqual(value,-7)
61 print 'Objective value returned by Daul simplex is Correct'
62
63 def test4(self):
64 print 'TEST 4----- TEST 4'
65 dual = linear.LinearProgram(self.a1,self.c1,self.b1)
66 flag , value = dual.preformDualSimplex()
67 self.assertEqual(dual.N.all(),self.N1.all())
68 self.assertEqual(dual.B.all(),self.B1.all())
69 print 'The matrices B and N returned are corrent for Dual simplex'
70
71 def test5(self):
72 print 'TEST 5----- TEST 5'
73 simplex = linear.LinearProgram(self.a2,self.c2,self.b2)
74 flag,value =simplex.preformPrimalSimplex()
75 self.assertEqual(flag,-1)
76 print 'The Problem is UnBounded.'
77
78 def test6(self):
79 print 'TEST 6----- TEST 6'
80 simplex = linear.LinearProgram(self.a3,self.c3,self.b3)
81 flag,primal_value =simplex.preformPrimalSimplex()
82
83 # new Matrices and vectors for the Dual
84 a_dual = -np.transpose(self.a3)
85 c_dual = -np.transpose(self.b3)
86 b_dual = -np.transpose(self.c3)
87
88 simplex = linear.LinearProgram(a_dual,c_dual,b_dual)
89 flag , dual_value = simplex.preformDualSimplex()
90
91 self.assertEqual(primal_value,-dual_value)
92
93 print 'Duality Thoery Checked'
94
95 def test7(self):
96 print 'TEST 7----- TEST 7'
97 simplex = linear.LinearProgram(self.a4,self.c4,self.b4)
98 flag,value =simplex.preformPrimalSimplex()
99 self.assertEqual(flag,0)
100 print 'Two Phase Method Problem Passed'
101
102 def test8(self):
103 print 'TEST 8----- TEST 8'
104 cc = criss_cross.CrissCross(self.a5,self.c5,self.b5)
105 value =cc.preformCrissCross()
106 self.assertEqual(-2.5,value)
107 print 'Value returned by the CrissCross is Correct'
108
109
110
111
112 if __name__ == '__main__':
113 unittest.main()
```

Line 29, Column 74Spaces: 4Python

```
ques_1 — -bash — 76x24
OK
abhi ques_1 $ python test.py
TEST 1----- TEST 1
Basic and Non Basic Indices return
.TEST 2----- TEST 2
Objective value returned by Primal simplex is Correct
.TEST 3----- TEST 3
Objective value returned by Daul simplex is Correct
.TEST 4----- TEST 4
The matrices B and N returned are corrent for Dual simplex
.TEST 5----- TEST 5
The Problem is UnBounded.
.TEST 6----- TEST 6
Duality Thoery Checked
.TEST 7----- TEST 7
Two Phase Method Problem Passed
.TEST 8----- TEST 8
Value returned by the CrissCross is Correct
.
-----
Ran 8 tests in 0.017s

OK
abhi ques_1 $
```

Primal Simplex Method Initial Matrices and Vector

$$A = \begin{bmatrix} 1.0 & -1.0 & 1.0 & 0.0 & 0.0 \\ 2.0 & -1.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Initial set of basic and nonbasic indices

$$\beta = \{3, 4, 5\} \quad \text{and} \quad \mathcal{N} = \{1, 2\}$$

Submatrice of A

$$B = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad \text{and} \quad N = \begin{bmatrix} 1.0 & -1.0 \\ 2.0 & -1.0 \\ 0.0 & 1.0 \end{bmatrix}$$

Initial values of the basic variables are given by

$$x_B^* = b = \begin{bmatrix} 1.0 \\ 3.0 \\ 5.0 \end{bmatrix}$$

Initial values of the nonbasic dual variables are given by

$$z_N^* = -c_N = \begin{bmatrix} -4.0 \\ -3.0 \end{bmatrix}$$

Since $x_B^ \geq 0$, the initial solution is primal feasible.*

Iteration No 1

Step 1.

Since z_N^ has some negative components, the current solution is not optimal.*

Step 2.

Since $z_1^ = -4.0$ and this is the most negative dual variables,*

we see that the entering index is $j = 1$

Step 3.

$$\Delta x_B = B^{-1} N e_j = \begin{bmatrix} 1.0 & -1.0 \\ 2.0 & -1.0 \\ 0.0 & 1.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 2.0 \\ 0.0 \end{bmatrix}$$

Step 4.

$$t = \left(\max \left\{ \frac{1.0}{1.0}, \frac{2.0}{3.0}, \frac{0.0}{5.0} \right\} \right)^{-1} = 1.0$$

Step 5.

In step 4, the ratio corresponds to basic index 3

$$i = 3$$

Step 6.

$$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i = - \begin{bmatrix} 1.0 & 2.0 & 0.0 \\ -1.0 & -1.0 & 1.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} -1.0 \\ 1.0 \end{bmatrix}$$

Step 7.

$$s = \frac{z_1^*}{\Delta z_1} = \frac{-4.0}{-1.0} = 4.0$$

Step 8.

$$x_1^* = 1.0, \quad x_{\mathcal{B}}^* = \begin{bmatrix} 1.0 \\ 3.0 \\ 5.0 \end{bmatrix} - 1.0 \begin{bmatrix} 1.0 \\ 2.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} 0.0 \\ 1.0 \\ 5.0 \end{bmatrix},$$

$$z_3^* = 4.0, \quad z_{\mathcal{N}}^* = \begin{bmatrix} -4.0 \\ -3.0 \end{bmatrix} - 4.0 \begin{bmatrix} -1.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 0.0 \\ -7.0 \end{bmatrix},$$

Step 9.

New set of basic and nonbasic indices

$$\beta = \{1, 4, 5\} \quad \text{and} \quad \mathcal{N} = \{3, 2\}$$

Corresponding new basis and nonbasis submatrices of A,

$$B = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 2.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad \text{and} \quad N = \begin{bmatrix} 1.0 & -1.0 \\ 0.0 & -1.0 \\ 0.0 & 1.0 \end{bmatrix}$$

New Basic primal variables and nonbasic dual variables :

$$x_{\mathcal{B}}^* = \begin{bmatrix} x_1^* \\ x_4^* \\ x_5^* \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \\ 5.0 \end{bmatrix} \quad z_{\mathcal{N}}^* = \begin{bmatrix} z_3^* \\ z_2^* \end{bmatrix} = \begin{bmatrix} 4.0 \\ -7.0 \end{bmatrix}$$

Iteration No 2

Step 1.

Since z_N^ has some negative components, the current solution is not optimal.*

Step 2.

Since $z_2^ = -7.0$ and this is the most negative dual variables,*

we see that the entering index is $j = 2$

Step 3.

$$\Delta x_B = B^{-1}Ne_j = \begin{bmatrix} 1.0 & -1.0 \\ -2.0 & 1.0 \\ 0.0 & 1.0 \end{bmatrix} \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} -1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

Step 4.

$$t = \left(\max \left\{ \frac{-1.0}{1.0}, \frac{1.0}{1.0}, \frac{1.0}{5.0} \right\} \right)^{-1} = 1.0$$

Step 5.

In step 4, the ratio corresponds to basic index 4

$$i = 4$$

Step 6.

$$\Delta z_N = -(B^{-1}N)^T e_i = - \begin{bmatrix} 1.0 & -2.0 & 0.0 \\ -1.0 & 1.0 & 1.0 \end{bmatrix} \begin{bmatrix} 0.0 \\ 1.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} 2.0 \\ -1.0 \end{bmatrix}$$

Step 7.

$$s = \frac{z_2^*}{\Delta z_2} = \frac{-7.0}{-1.0} = 7.0$$

Step 8.

$$x_2^* = 1.0, \quad x_B^* = \begin{bmatrix} 1.0 \\ 1.0 \\ 5.0 \end{bmatrix} - 1.0 \begin{bmatrix} -1.0 \\ 1.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 0.0 \\ 4.0 \end{bmatrix},$$

$$z_4^* = 7.0, \quad z_N^* = \begin{bmatrix} 4.0 \\ -7.0 \end{bmatrix} - 7.0 \begin{bmatrix} 2.0 \\ -1.0 \end{bmatrix} = \begin{bmatrix} -10.0 \\ 0.0 \end{bmatrix},$$

Step 9.

New set of basic and nonbasic indices

$$\beta = \{1, 2, 5\} \quad \text{and} \quad \mathcal{N} = \{3, 4\}$$

Corresponding new basis and nonbasis submatrices of A,

$$B = \begin{bmatrix} 1.0 & -1.0 & 0.0 \\ 2.0 & -1.0 & 0.0 \\ 0.0 & 1.0 & 1.0 \end{bmatrix} \quad \text{and} \quad N = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \\ 0.0 & 0.0 \end{bmatrix}$$

New Basic primal variables and nonbasic dual variables :

$$x_{\mathcal{B}}^* = \begin{bmatrix} x_1^* \\ x_2^* \\ x_5^* \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.0 \\ 4.0 \end{bmatrix} \quad z_{\mathcal{N}}^* = \begin{bmatrix} z_3^* \\ z_4^* \end{bmatrix} = \begin{bmatrix} -10.0 \\ 7.0 \end{bmatrix}$$

Iteration No 3

Step 1.

Since z_N^ has some negative components, the current solution is not optimal.*

Step 2.

Since $z_3^ = -10.0$ and this is the most negative dual variables,*

we see that the entering index is $j = 3$

Step 3.

$$\Delta x_{\mathcal{B}} = B^{-1} N e_j = \begin{bmatrix} -1.0 & 1.0 \\ -2.0 & 1.0 \\ 2.0 & -1.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} -1.0 \\ -2.0 \\ 2.0 \end{bmatrix}$$

Step 4.

$$t = \left(\max \left\{ \frac{-1.0}{2.0}, \frac{-2.0}{1.0}, \frac{2.0}{4.0} \right\} \right)^{-1} = 2.0$$

Step 5.

In step 4, the ratio corresponds to basic index 5

$$i = 5$$

Step 6.

$$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i = - \begin{bmatrix} -1.0 & -2.0 & 2.0 \\ 1.0 & 1.0 & -1.0 \end{bmatrix} \begin{bmatrix} 0.0 \\ 0.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} -2.0 \\ 1.0 \end{bmatrix}$$

Step 7.

$$s = \frac{z_3^*}{\Delta z_3} = \frac{-10.0}{-2.0} = 5.0$$

Step 8.

$$x_3^* = 2.0, \quad x_{\mathcal{B}}^* = \begin{bmatrix} 2.0 \\ 1.0 \\ 4.0 \end{bmatrix} - 2.0 \begin{bmatrix} -1.0 \\ -2.0 \\ 2.0 \end{bmatrix} = \begin{bmatrix} 4.0 \\ 5.0 \\ 0.0 \end{bmatrix},$$

$$z_5^* = 5.0, \quad z_{\mathcal{N}}^* = \begin{bmatrix} -10.0 \\ 7.0 \end{bmatrix} - 5.0 \begin{bmatrix} -2.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 0.0 \\ 2.0 \end{bmatrix},$$

Step 9.

New set of basic and nonbasic indices

$$\beta = \{1, 2, 3\} \quad \text{and} \quad \mathcal{N} = \{5, 4\}$$

Corresponding new basis and nonbasis submatrices of A,

$$B = \begin{bmatrix} 1.0 & -1.0 & 1.0 \\ 2.0 & -1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \quad \text{and} \quad N = \begin{bmatrix} 0.0 & 0.0 \\ 0.0 & 1.0 \\ 1.0 & 0.0 \end{bmatrix}$$

New Basic primal variables and nonbasic dual variables :

$$x_{\mathcal{B}}^* = \begin{bmatrix} x_1^* \\ x_2^* \\ x_3^* \end{bmatrix} = \begin{bmatrix} 4.0 \\ 5.0 \\ 2.0 \end{bmatrix} \quad z_{\mathcal{N}}^* = \begin{bmatrix} z_5^* \\ z_4^* \end{bmatrix} = \begin{bmatrix} 5.0 \\ 2.0 \end{bmatrix}$$

Iteration No 4

Step 1.

Since $z_{\mathcal{N}}^$ has all nonnegative components, the current solution is optimal.*

$$\zeta^* = 4.0x_1^* + 3.0x_2^* = 31.0$$

Dual Simplex Method Initial Matrices and Vector

$$A = \begin{bmatrix} -2.0 & -1.0 & 1.0 & 0.0 & 0.0 \\ -2.0 & 4.0 & 0.0 & 1.0 & 0.0 \\ -1.0 & 3.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Initial set of basic and nonbasic indices

$$\beta = \{3, 4, 5\} \quad \text{and} \quad \mathcal{N} = \{1, 2\}$$

Submatrice of A

$$B = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad \text{and} \quad N = \begin{bmatrix} -2.0 & -1.0 \\ -2.0 & 4.0 \\ -1.0 & 3.0 \end{bmatrix}$$

Initial values of the basic variables are given by

$$x_B^* = b = \begin{bmatrix} 4.0 \\ -8.0 \\ -7.0 \end{bmatrix}$$

Initial values of the nonbasic dual variables are given by

$$z_N^* = -c_N = \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$$

Since $z_N^ \geq 0$, the initial solution is Dual feasible.*

Iteration No 1

Step 1.

Since x_B^ has some negative components, the current solution is not optimal.*

Step 2.

Since $x_4^ = -8.0$ and this is the most negative dual variables,*

we see that the entering index is $i = 4$

Step 3.

$$\Delta z_N = -(B^{-1}N)^T e_i = - \begin{bmatrix} -2.0 & -2.0 & -1.0 \\ -1.0 & 4.0 & 3.0 \end{bmatrix} \begin{bmatrix} 0.0 \\ 1.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} 2.0 \\ -4.0 \end{bmatrix}$$

Step 4.

$$t = \left(\max \left\{ \frac{2.0}{1.0}, \frac{-4.0}{1.0} \right\} \right)^{-1} = 0.5$$

Step 5.

In step 4, the ratio corresponds to basic index 1
 $j = 1$

Step 6.

$$\Delta x_B = B^{-1} N e_j = \begin{bmatrix} -2.0 & -1.0 \\ -2.0 & 4.0 \\ -1.0 & 3.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} -2.0 \\ -2.0 \\ -1.0 \end{bmatrix}$$

Step 7.

$$t = \frac{x_1^*}{\Delta x_1} = \frac{-8.0}{-2.0} = 4.0$$

Step 9.

New set of basic and nonbasic indices
 $\beta = \{3, 1, 5\}$ and $\mathcal{N} = \{4, 2\}$

Corresponding new basis and nonbasis submatrices of A,

$$B = \begin{bmatrix} 1.0 & -2.0 & 0.0 \\ 0.0 & -2.0 & 0.0 \\ 0.0 & -1.0 & 1.0 \end{bmatrix} \quad \text{and} \quad N = \begin{bmatrix} 0.0 & -1.0 \\ 1.0 & 4.0 \\ 0.0 & 3.0 \end{bmatrix}$$

New Basic primal variables and nonbasic dual variables :

$$x_B^* = \begin{bmatrix} x_3^* \\ x_1^* \\ x_5^* \end{bmatrix} = \begin{bmatrix} 12.0 \\ 4.0 \\ -3.0 \end{bmatrix} \quad z_{\mathcal{N}}^* = \begin{bmatrix} z_4^* \\ z_2^* \end{bmatrix} = \begin{bmatrix} 0.5 \\ 3.0 \end{bmatrix}$$

Iteration No 2

Step 1.

Since x_B^ has some negative components, the current solution is not optimal.*

Step 2.

Since $x_5^ = -3.0$ and this is the most negative dual variables,*

we see that the entering index is $i = 5$

Step 3.

$$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i = - \begin{bmatrix} -1.0 & -0.5 & -0.5 \\ -5.0 & -2.0 & 1.0 \end{bmatrix} \begin{bmatrix} 0.0 \\ 0.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ -1.0 \end{bmatrix}$$

Step 4.

$$t = \left(\max \left\{ \frac{0.5}{0.5}, \frac{-1.0}{3.0} \right\} \right)^{-1} = 1.0$$

Step 5.

In step 4, the ratio corresponds to basic index 4

$$j = 4$$

Step 6.

$$\Delta x_{\mathcal{B}} = B^{-1} N e_j = \begin{bmatrix} -1.0 & -5.0 \\ -0.5 & -2.0 \\ -0.5 & 1.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} -1.0 \\ -0.5 \\ -0.5 \end{bmatrix}$$

Step 7.

$$t = \frac{x_4^*}{\Delta x_4} = \frac{-3.0}{-0.5} = 6.0$$

Step 9.

New set of basic and nonbasic indices

$$\beta = \{3, 1, 4\} \quad \text{and} \quad \mathcal{N} = \{5, 2\}$$

Corresponding new basis and nonbasis submatrices of A,

$$B = \begin{bmatrix} 1.0 & -2.0 & 0.0 \\ 0.0 & -2.0 & 1.0 \\ 0.0 & -1.0 & 0.0 \end{bmatrix} \quad \text{and} \quad N = \begin{bmatrix} 0.0 & -1.0 \\ 0.0 & 4.0 \\ 1.0 & 3.0 \end{bmatrix}$$

New Basic primal variables and nonbasic dual variables :

$$x_{\mathcal{B}}^* = \begin{bmatrix} x_3^* \\ x_1^* \\ x_4^* \end{bmatrix} = \begin{bmatrix} 18.0 \\ 7.0 \\ 6.0 \end{bmatrix} \quad z_{\mathcal{N}}^* = \begin{bmatrix} z_5^* \\ z_2^* \end{bmatrix} = \begin{bmatrix} 1.0 \\ 4.0 \end{bmatrix}$$

Iteration No 3

Step 1.

Since x_B^ has all nonnegative components, the current solution is optimal.*

$$\zeta^* = -1.0x_1^* + -1.0x_2^* = -7.0$$