数理計画とアルゴリズム rep3 2055713B 朝比奈太郎 2022/06/25

テキスト, 手紙

自動的に生成された説明

コード

|  |
| --- |
| #タブローのアルゴリズムの実装ができなかった。  #タブロー以外の処理は行えた  from traceback import print\_tb  Z = [0, -1, -1, -1]  X\_4 = [8, 1, -1, 2]  X\_5 = [1, 2, -3, -1]  def draw\_1():  print(" | 定 -X\_1 -X\_2 -X\_3")  print("-------------------------")  print("X\_4|" + " " + str(X\_4[0]) + " " + str(X\_4[1]) + " " + str(X\_4[2]) + " " + str(X\_4[3]))  print("X\_5|" + " " + str(X\_5[0]) + " " + str(X\_5[1]) + " " + str(X\_5[2]) + " " + str(X\_5[3]))  print("Z |" + " " + str(Z[0]) + " " + str(Z[1]) + " " + str(Z[2]) + " " + str(Z[3]))  print("")  def draw\_2():  print(" | 定 -X\_5 -X\_4 -X\_3")  print("-------------------------")  print("X\_2|" + " " + str(X\_2[0]) + " " + str(X\_2[1]) + " " + str(X\_2[2]) + " " + str(X\_2[3]))  print("X\_1|" + " " + str(X\_1[0]) + " " + str(X\_1[1]) + " " + str(X\_1[2]) + " " + str(X\_1[3]))  print("Z |" + " " + str(Z[0]) + " " + str(Z[1]) + " " + str(Z[2]) + " " + str(Z[3]))  print("")  print("最初のシンプレックスのタブローは")  print("")  draw\_1()  ##PEを探す過程#############  def search\_PE\_1(list):#絶対値が大きい列の番号を返す  num = 0  result = max(list, key=abs)  for i in range(len(list)):  if result == list[i]:  num += i  #print(num)  return num  search\_PE\_1(Z)#1を返す  def search\_PE\_2(list1, list2): #1回目はX\_5を返す　2回目 -> x ...  num1 = list1[0] / list1[search\_PE\_1(Z)]  num2 = list2[0] / list2[search\_PE\_1(Z)]  if list1[search\_PE\_1(Z)] >= 0 and list2[search\_PE\_1(Z)] >= 0:  if num1 >= num2:  return list2  else:  return list1  elif list1[search\_PE\_1(Z)] < 0:  return list2  else:  return list1  #print(search\_PE\_2(X\_4, X\_5))  ##ここから求めたPEを対象に計算###########  def calc\_PE(list):  for i in range(len(list)):  if not i == search\_PE\_1(Z): #PE以外の同じ行の値をPEで割る  list[i] = list[i] / list[search\_PE\_1(Z)]  #最後にPEをPE自身で割る -> 1になる  list[search\_PE\_1(Z)] = 1  #計算結果を上書き  X\_5 = list  calc\_PE(search\_PE\_2(X\_4, X\_5))  print("PEを探し、計算すると")  print("")  draw\_1()  X\_2 = [15/2, 0, 1/2, 5/2]  X\_1 = X\_5  Z = [1/2, 0, -5/2, -3/2]  print("次のシンプレックスタブローは")  print("")  draw\_2()#次のシンプレクスタブロー  search\_PE\_1(Z) #-2.5が絶対値最大なので2を返す  #print(search\_PE\_2(X\_1, X\_2))  calc\_PE(search\_PE\_2(X\_1, X\_2))  print("PEを探し、計算すると")  print("")  draw\_2()  X\_1 = [23, 1, 0, 7]  Z = [38, 0, 0, 11]  print("次のタブローを計算すると")  print("")  draw\_2()  ##最後に条件を満たしているか判断###########  def check\_finish(list):  for i in range(len(list)):  if list[i] >= 0:  return True  else:  return False  def final\_check():  if check\_finish(Z) == True:  print("条件を満たしています")  else:  print("条件を満たしていません")  check\_finish(Z)  final\_check()  print("従って、求める答えは")  print("X\_1= " + str(X\_1[0]))  print("X\_2= " + str(X\_2[0]))  print("X\_3= 0")  print("Z= " + str(Z[0])) |

考察

このコードは練習問題①のコードだが、このコードの変数を変更することによって②、③も同様にして解くことができる。次のシンプレックスタブローを計算するアルゴリズムを実装することはできなかった。練習問題①の解答例を見ながら、実装したが、PEを探し、計算する関数(search\_PE\_1, search\_PE\_2, calc\_PE)を用いて、関数内で他の関数を実装する部分を工夫した。このように実装することによって、関数間のつながりが強くなり、実行する際に見えやすいものになると考えた。また、課題のB)の条件である、停止条件判定を正しくできるかについて、check\_finish(), final\_check()の２関数を実装して判断を行うようにした。

実行結果

テキスト

自動的に生成された説明