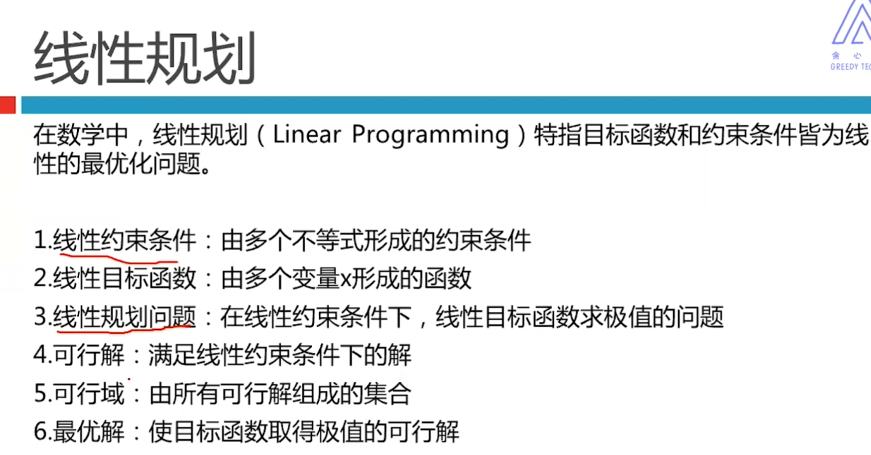
# 线性规划：Linear programming



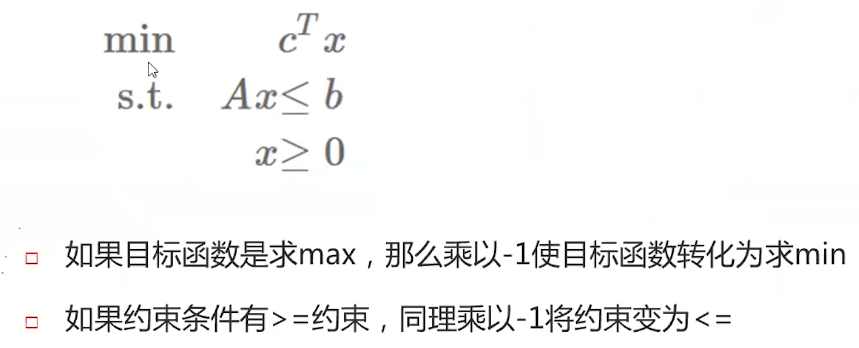
## 定义线性规划模型的步骤：

线性规划，在线性约束条件下，线性目标函数求极值。（凸优化问题）

步骤：

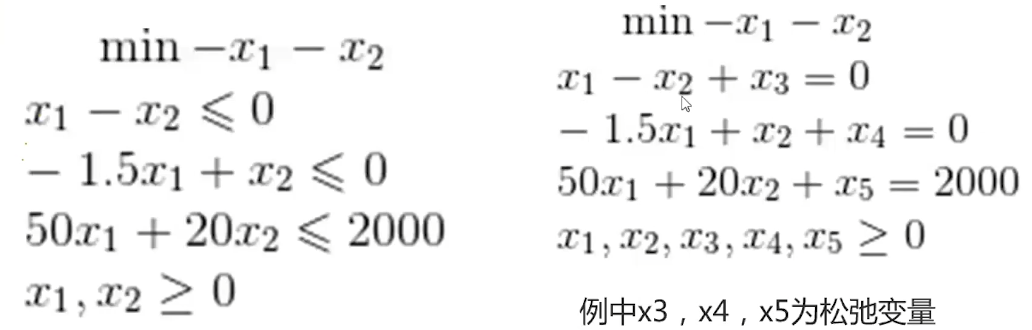
* 确定决策变量
* 确定线性目标函数，求max或min
* 确定线性约束条件
* 写出数学模型

## 线性规划标准形式

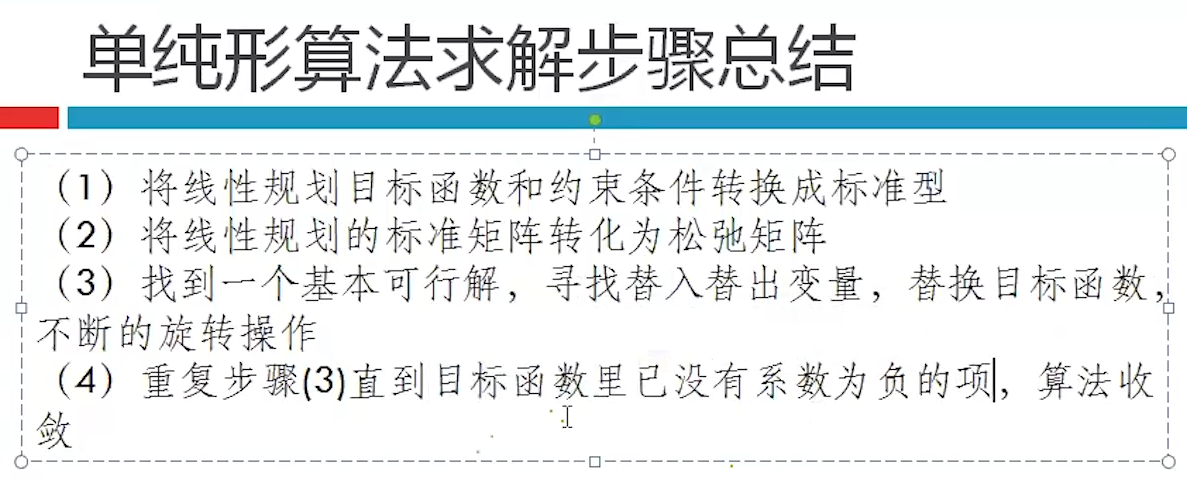


## 转换为松驰型

松驰型：用等式约束来等价描述不等式约束松弛变量度量了等式约束与原不等式约束之间的松弛或差别。



## 1.4 单纯形算法求解步骤





|  |
| --- |
| '''  原题目：  有2000元经费，需要采购单价为50元的若干桌子和单价为20元的若干椅子，你希望桌椅的总数尽可能的多，但要求椅子数量不少于桌子数量，且不多于桌子数量的1.5倍，那你需要怎样的一个采购方案呢？  解：要采购x1张桌子，x2把椅子  max z= x1 + x2  s.t. x1 - x2 <= 0  1.5x1 >= x2  50x1 + 20x2 <= 2000  x1, x2 >=0  在python中此类线性规划问题可用lp solver解决  scipy.optimize.\_linprog def linprog(c: int,  A\_ub: Optional[int] = None,  b\_ub: Optional[int] = None,  A\_eq: Optional[int] = None,  b\_eq: Optional[int] = None,  bounds: Optional[Iterable] = None,  method: Optional[str] = 'simplex',  callback: Optional[Callable] = None,  options: Optional[dict] = None) -> OptimizeResult  矩阵A：就是约束条件的系数（等号左边的系数）  矩阵B：就是约束条件的值（等号右边）  矩阵C：目标函数的系数值  '''  from scipy import optimize as opt  import numpy as np  #参数  #c是目标函数里变量的系数  c=np.array([1,1])  #a是不等式条件的变量系数  a=np.array([[1,-1],[-1.5,1],[50,20]])  #b是是不等式条件的常数项  b=np.array([0,0,2000])  #a1，b1是等式条件的变量系数和常数项，这个例子里无等式条件,不要这两项  #a1=np.array([[1,1,1]])  #b1=np.array([7])  #限制  lim1=(0,None) #(0,None)->(0,+无穷)  lim2=(0,None)  #调用函数  ans=opt.linprog(-c,a,b,bounds=(lim1,lim2))  #输出结果  print(ans)  #注意：我们这里的应用问题，椅子不能是0.5把，所以最后应该采购37把椅子 |

松驰形

|  |
| --- |
| import numpy as np  class Simplex(object):  def \_\_init\_\_(self, obj, max\_mode=False):  self.max\_mode = max\_mode # 默认是求min，如果是max目标函数要乘-1  self.mat = np.array([[0] + obj]) \* (-1 if max\_mode else 1) #矩阵先加入目标函数  def add\_constraint(self, a, b):  self.mat = np.vstack([self.mat, [b] + a]) #矩阵加入约束  def solve(self):  m, n = self.mat.shape # 矩阵里有1行目标函数，m - 1行约束，应加入m-1个松弛变量  temp, B = np.vstack([np.zeros((1, m - 1)), np.eye(m - 1)]), list(range(n - 1, n + m - 1)) # temp是一个对角矩阵，B是个递增序列  mat = self.mat = np.hstack([self.mat, temp]) # 横向拼接  while mat[0, 1:].min() < 0: #判断目标函数里是否还有负系数项  col = np.where(mat[0, 1:] < 0)[0][0] + 1 # 在目标函数里找到第一个负系数的变量，找到替入变量  row = np.array([mat[i][0] / mat[i][col] if mat[i][col] > 0 else 0x7fffffff for i in  range(1, mat.shape[0])]).argmin() + 1 # 找到最严格约束的行，也就找到替出变量  if mat[row][col] <= 0: return None # 若无替出变量，原问题无界  mat[row] /= mat[row][col] #替入变量和替出变量互换  ids = np.arange(mat.shape[0]) != row  mat[ids] -= mat[row] \* mat[ids, col:col + 1] # 对i!= row的每一行约束条件，做替入和替出变量的替换  B[row] = col #用B数组记录替入的替入变量  return mat[0][0] \* (1 if self.max\_mode else -1), {B[i]: mat[i, 0] for i in range(1, m) if B[i] < n} #返回目标值，对应x的解就是基本变量为对应的bi，非基本变量为0  """  minimize -x1 - 14x2 - 6x3  st  x1 + x2 + x3 <=4  x1 <= 2  x3 <= 3  3x2 + x3 <= 6  x1 ,x2 ,x3 >= 0  answer :-32  """  t = Simplex([-1, -14, -6])  t.add\_constraint([1, 1, 1], 4)  t.add\_constraint([1, 0, 0], 2)  t.add\_constraint([0, 0, 1], 3)  t.add\_constraint([0, 3, 1], 6)  print(t.solve())  print(t.mat) |