КУРС "ИССЛЕДОВАНИЕ ОПЕРАЦИЙ"

Лабораторная работа No 1

Тема:

Линейное программирование.

Решение задач оптимизации производства

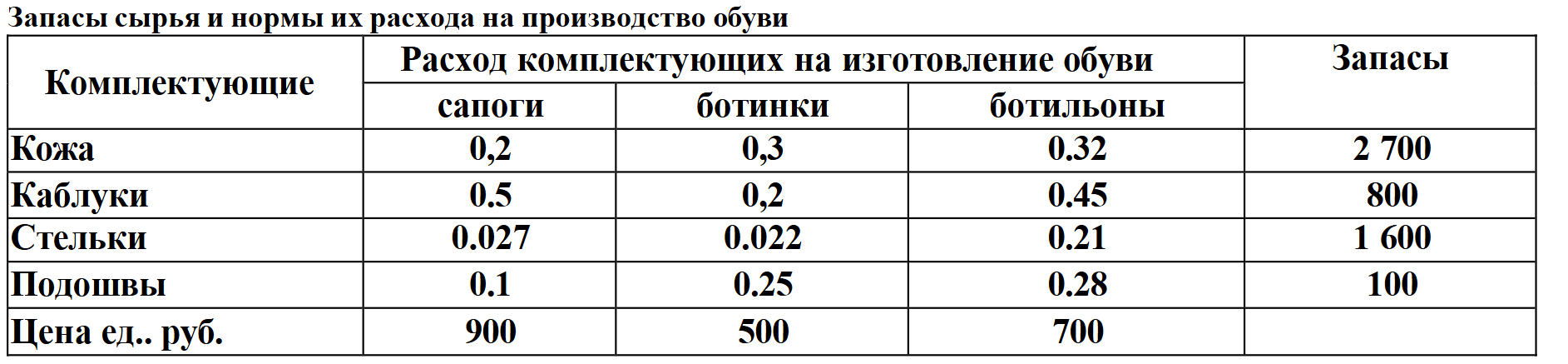
Вариант 33

Сергиенко Лев 12 группа МСС, 4 курс

# **Формулировка проблемы**

В обувном цехе для изготовления трех моделей обуви используются четыре вида комплектующих материалов, запасы которых, расход их на изготовление одной пары обуви и цены, получаемые от реализации пары обуви, приведены в таблице.

Составьте математическую модель задачи и найдите оптимальное решение.



Это классическая задача линейного программирования - задача планирования производства при ограниченных ресурсах.

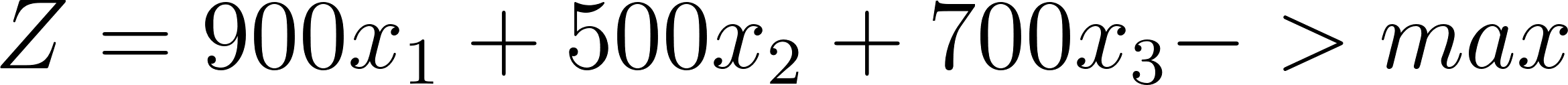
Переменные - объёмы выпуска (действительные, непрерывные), ограничения - линейные по ресурсам, цель - линейная (максимизация прибыли/выручки). Поэтому применимы стандартные методы LP: симплекс, внутренние точки, анализ базиса, исследование вершин многогранника.

# **Математическая модель ЛП**

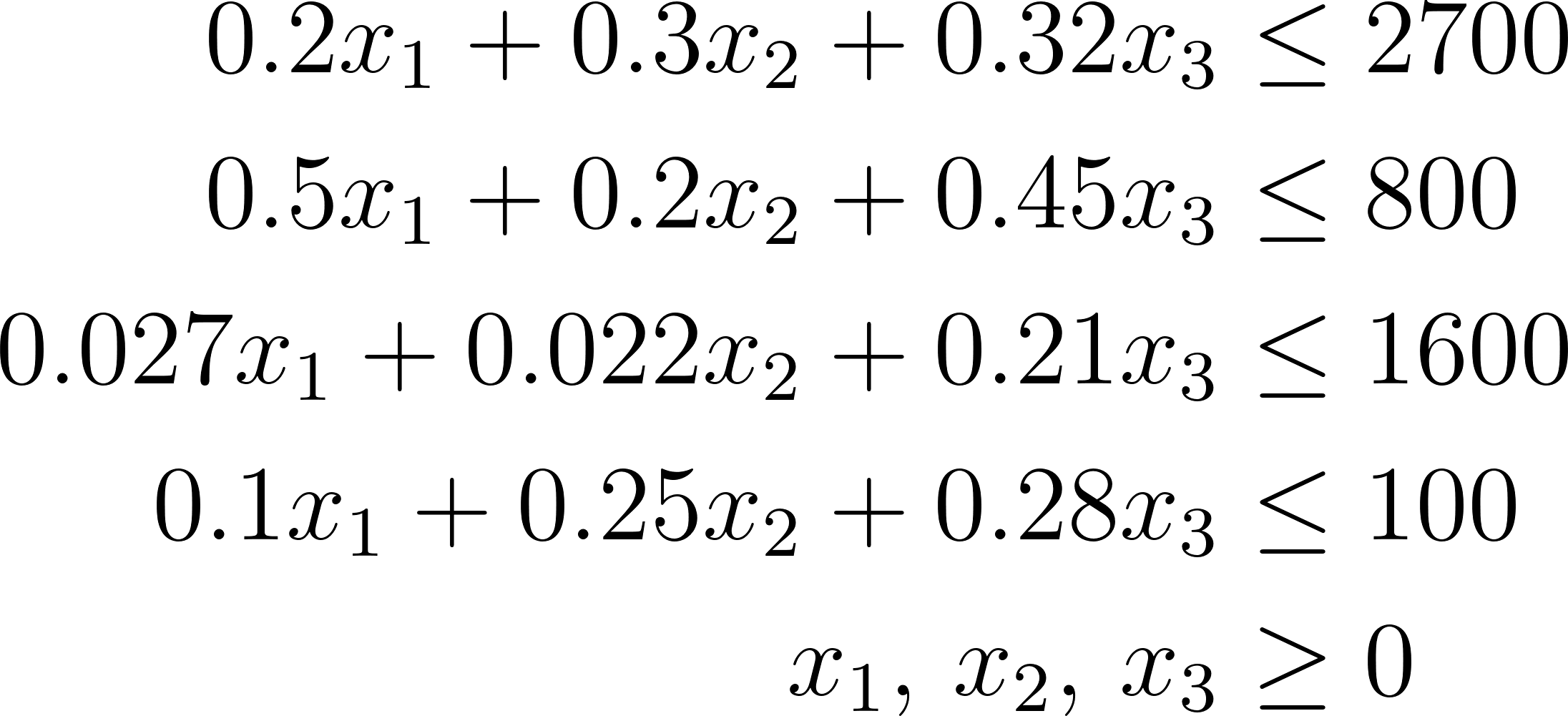
Переменные:

* [](https://www.codecogs.com/eqnedit.php?latex=x_1#0) - число пар сапог;
* [](https://www.codecogs.com/eqnedit.php?latex=x_2#0) - число пар ботинок;
* [](https://www.codecogs.com/eqnedit.php?latex=x_3#0) - число пар ботильонов.

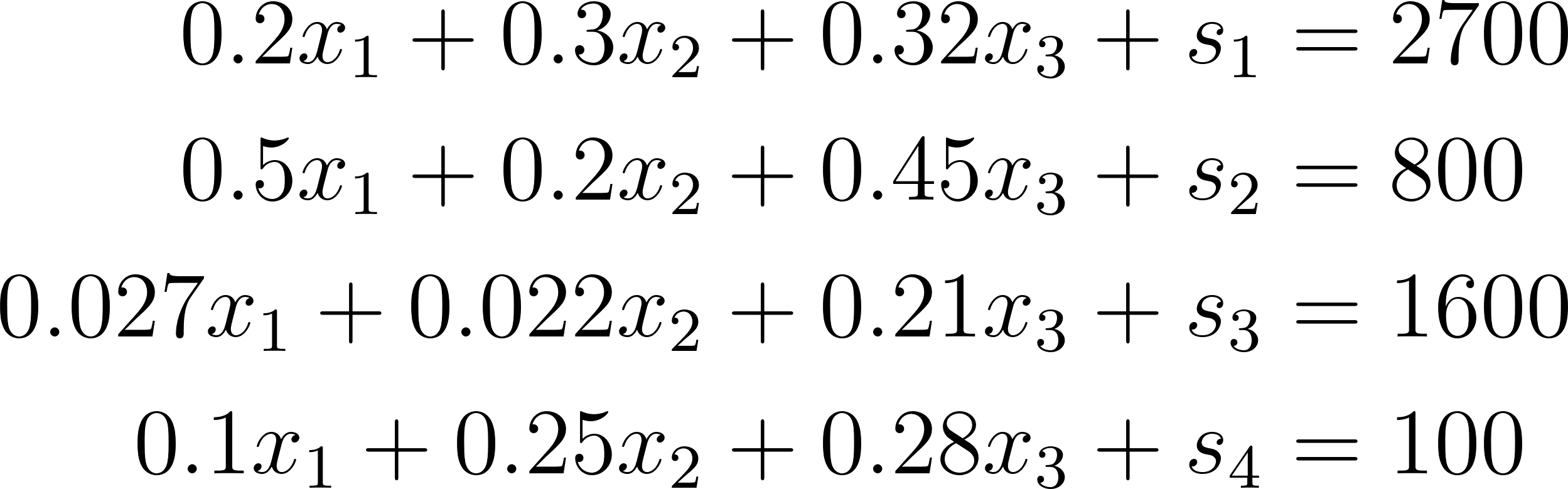
Целевая функция (максимизация выручки):

[](https://www.codecogs.com/eqnedit.php?latex=%20Z%20%3D%20900x_1%20%2B%20500x_2%20%2B%20700x_3%20-%3E%20max%20#0)

Ограничения (ресурсы):

[](https://www.codecogs.com/eqnedit.php?latex=%20%5Cbegin%7Baligned%7D%200.2x_1%20%2B%200.3x_2%20%2B%200.32x_3%20%26%5Cle%202700%20%20%5C%5C%200.5x_1%20%2B%200.2x_2%20%2B%200.45x_3%20%26%5Cle%20800%20%5C%5C%200.027x_1%20%2B%200.022x_2%20%2B%200.21x_3%20%26%5Cle%201600%20%20%5C%5C%200.1x_1%20%2B%200.25x_2%20%2B%200.28x_3%20%26%5Cle%20100%20%20%5C%5C%20x_1%2C%5C%2Cx_2%2C%5C%2Cx_3%20%26%5Cge%200%20%5Cend%7Baligned%7D%20#0)

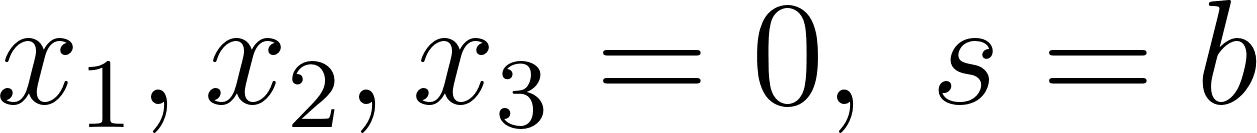
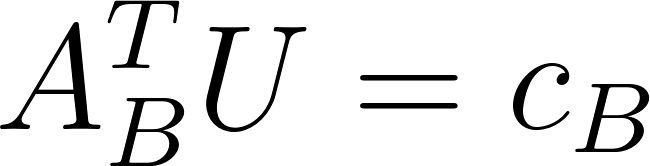
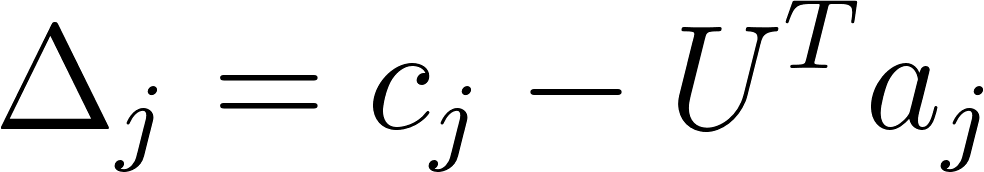
Добавляя s₁..s₄ ≥ 0, получаем систему равенств:

[](https://www.codecogs.com/eqnedit.php?latex=%20%5Cbegin%7Baligned%7D%200.2x_1%20%2B%200.3x_2%20%2B%200.32x_3%20%2B%20s_1%20%26%3D%202700%5C%5C%200.5x_1%20%2B%200.2x_2%20%2B%200.45x_3%20%2B%20s_2%20%26%3D%20800%5C%5C%200.027x_1%20%2B%200.022x_2%20%2B%200.21x_3%20%2B%20s_3%20%26%3D%201600%5C%5C%200.1x_1%20%2B%200.25x_2%20%2B%200.28x_3%20%2B%20s_4%20%26%3D%20100%20%5Cend%7Baligned%7D%20#0)

Это стандартная LP-задача (m=4, n=3).

# **Листинг программы**

Я реализовал прямой симплекс в Python. Код выполняет:

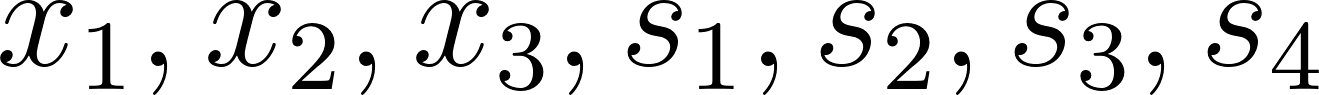
* формирование полной матрицы с искусственными переменными,
* стартовое допустимое решение x0: [](https://www.codecogs.com/eqnedit.php?latex=x_1%2Cx_2%2Cx_3%3D0%2C%5C%3B%20s%3Db#0),
* на итерациях вычисляет коэффициенты U решая [](https://www.codecogs.com/eqnedit.php?latex=A_B%5ET%20U%20%3D%20c_B#0),
* вычисляет редуцированные стоимости [](https://www.codecogs.com/eqnedit.php?latex=%5CDelta_j%20%3D%20c_j%20-%20U%5ET%20a_j#0),
* выбирает входящую переменную с учётом ограничений,
* строит направление изменения переменных,
* обновляет базис и решение.

| import numpy as np   def simplex(x, basis, A, c, x\_min, x\_max):  m, n\_vars = A.shape   # 1. Формирование матрицы базиса и вычисление U  A\_basis = A[:, (basis - 1)] # m x m  c\_basis = c[(basis - 1)] # m   try:  U = np.linalg.solve(A\_basis.T, c\_basis)  except np.linalg.LinAlgError as e:  raise RuntimeError("Базисная матрица вырождена, нельзя продолжить (повторная базовая матрица).") from e   print("Матрица базиса (строками):")  for i in range(m):  print(A\_basis.T[i], " ", c\_basis[i])  print("\nU (дуальные):", U)   # 2. Вычисляем дельты для не базисных переменных  deltas = {}  is\_minus\_exist = False  for j in range(n\_vars):  if j not in (basis - 1):  delta = c[j] - np.dot(U, A[:, j])  # Определяем символ +/-, по вашей логике:  if delta > 1e-12:  sign = "-" if x[j] < x\_max[j] else "+"  elif delta < -1e-12:  sign = "-" if x[j] > x\_min[j] else "+"  else:  # delta == 0  sign = "-" if (x[j] > x\_min[j] and x[j] < x\_max[j]) else "+"  deltas[j] = (delta, sign)  if sign == "-":  is\_minus\_exist = True   print("\nВычисленные дельты и знаки (delta, sign):")  for j, (delta, sign) in deltas.items():  print(f" col {j + 1}: delta={delta:.6f}, sign={sign}")   # Если нет отрицательных, метод сошелся  if not is\_minus\_exist:  print("\nМетод сошёлся: нет входящих переменных. Текущее решение оптимально.")  print("X =", x)  print("Базис =", basis)  print("Z =", float(np.dot(c, x)))  return x, basis, 0   # 3. Выбираем j0 -- первый индекс с sign == "-"  j0 = None  j0\_delta = None  for j, (delta, sign) in deltas.items():  if sign == "-":  j0 = j  j0\_delta = delta  break   print(f"\nВходящая переменная: j0 = {j0 + 1}, delta = {j0\_delta:.6f}")   # 4. Формирование направления l (изменение переменных при увеличении x\_j0)   l = np.zeros(n\_vars)  # логика: если delta>0 => l[j0]=1, иначе -1  l[j0] = 1.0 if j0\_delta > 0 else -1.0  # остальные не базисные (кроме j0) = 0; базисные будем вычислять далее  # Формируем уравнение для изменения базисных переменных: A\_basis \* delta\_x\_basis = - A[:,j0] \* l[j0]  rhs = -A[:, j0] \* l[j0]  # Решаем для изменений базисных переменных  delta\_x\_basis = np.linalg.solve(A\_basis, rhs) # m  # Заполняем l для базисных индексов  for idx\_i, col\_index in enumerate((basis - 1)):  l[col\_index] = delta\_x\_basis[idx\_i]   print("\nВектор направления l (изменения x при увеличении входящей):")  for i in range(n\_vars):  print(f" l[{i + 1}] = {l[i]:.6f}", end=", ")  print()   # 5. Вычисление допустимых шагов Q для каждой переменной  Q = np.full(n\_vars, np.inf)  # Для j0 -- может быть ограничено его верхней границей (x\_max[j0] может быть inf)  # но чаще шаг на входящую тяготеет к ограничениям базисных переменных  if np.isfinite(x\_max[j0]) and np.isfinite(x\_min[j0]):  Q[j0] = x\_max[j0] - x[j0]  else:  Q[j0] = np.inf   for i in range(n\_vars):  if abs(l[i]) < 1e-15:  Q[i] = np.inf  continue  if l[i] > 0:  # увеличивая x\_j0, переменная i растёт, ограничение -- верхняя граница  if np.isfinite(x\_max[i]):  Q[i] = (x\_max[i] - x[i]) / l[i]  else:  Q[i] = np.inf  else:  # l[i] < 0, переменная i убывает, ограничение -- нижняя граница  if np.isfinite(x\_min[i]):  Q[i] = (x\_min[i] - x[i]) / l[i] # l[i] отрицательное => деление даст полож. шаг  else:  Q[i] = np.inf   print("\nКандидатные значения Тета (Q) для ограничений:")  for i in range(n\_vars):  print(f" Q[{i + 1}] = {Q[i]} ", end=", ")  print()   # 6. Выбираем минимальное положительное Q0  # Отсекаем отрицательные и очень малые  valid\_Q = [(i, Q[i]) for i in range(n\_vars) if Q[i] > 1e-12 and np.isfinite(Q[i])]  if not valid\_Q:  # Если нет ограничений -- неограничено в улучшении  raise RuntimeError("Задача неограниченна по направлению улучшения (unbounded).")  j\_star, Q0 = min(valid\_Q, key=lambda t: (t[1], t[0]))   print(f"\nМинимальный шаг Q0 = {Q0:.6f}, это даст уходящую переменную j\* = {j\_star + 1}")   # 7. Обновление базиса: заменить j\_star на j0  basis\_new = basis.copy()  # заменим элемент равный j\_star+1 на j0+1  replaced = False  for idx in range(len(basis\_new)):  if basis\_new[idx] == j\_star + 1:  basis\_new[idx] = j0 + 1  replaced = True  break  if not replaced:  # Если уходящая переменная не в базисе (вырождение), то просто добавим и удалим самый правый  basis\_new = np.append(basis\_new, j0 + 1)  basis\_new = np.delete(basis\_new, 0)  basis\_new = np.sort(basis\_new)  # 8. Обновляем x  x\_new = x + Q0 \* l   print("\nНовый базис:", basis\_new)  print("Новое решение x:")  for i in range(n\_vars):  print(f" x[{i + 1}] = {x\_new[i]:.6f}", end=", ")  print("\nТекущее значение целевой функции Z =", float(np.dot(c, x\_new)))   return x\_new, basis\_new, 1   A\_orig = np.array(  [  [0.2, 0.3, 0.32], # кожа  [0.5, 0.2, 0.45], # каблуки  [0.027, 0.022, 0.21], # стельки  [0.1, 0.25, 0.28], # подошвы  ] ) b = np.array([2700.0, 800.0, 1600.0, 100.0])  m, n = A\_orig.shape  # Добавляем slack-переменные: всего n\_vars = n + m A = np.hstack([A\_orig, np.eye(m)]) # размер m x (n+m) n\_vars = n + m  c = np.concatenate([np.array([900.0, 500.0, 700.0]), np.zeros(m)])  x\_min = np.zeros(n\_vars) x\_max = np.full(n\_vars, np.inf)  # Начальное допустимое решение: x1..x3 = 0, slack = b (s\_i = b\_i) x0 = np.concatenate([np.zeros(n), b])  basis0 = np.arange(n + 1, n + m + 1)  print("Матрица A (с добавленными slack):\n", A) print("Начальное допустимое x0:", x0) print("Начальный базис:", basis0)  x = x0.copy() basis = basis0.copy() cont = 1 iter\_count = 0 while cont and iter\_count < 20:  iter\_count += 1  print("\n======================== Итерация", iter\_count, "========================")  x, basis, cont = simplex(x, basis, A, c, x\_min, x\_max)  print("\nИТОГ:") print("Итераций:", iter\_count) print("Решение x:", x) print("Базис:", basis) print("Z =", float(np.dot(c, x))) |
| --- |

# **Результат**

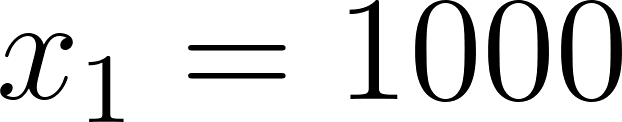
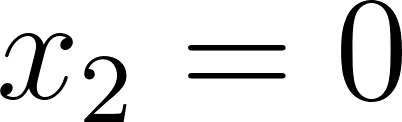
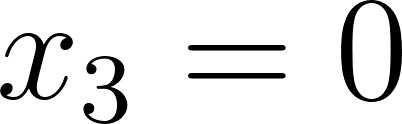
Запуск алгоритма дал:

- Статус: Оптимально найдено

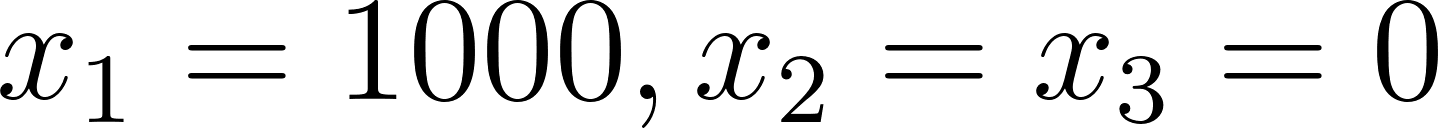
- Оптимальное решение (вектор всех переменных: [](https://www.codecogs.com/eqnedit.php?latex=%20x_1%2Cx_2%2Cx_3%2Cs_1%2Cs_2%2Cs_3%2Cs_4%20#0)):

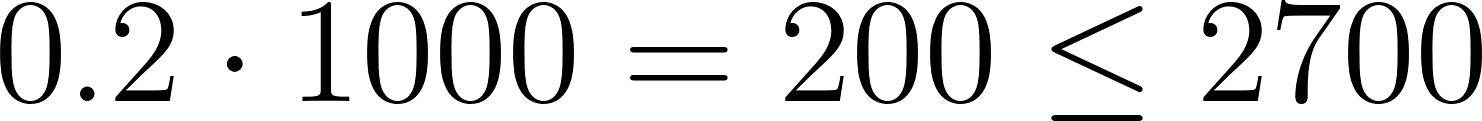
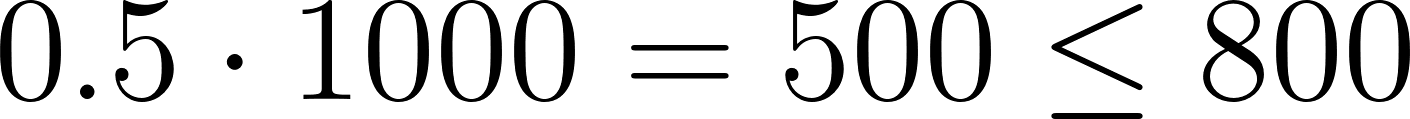
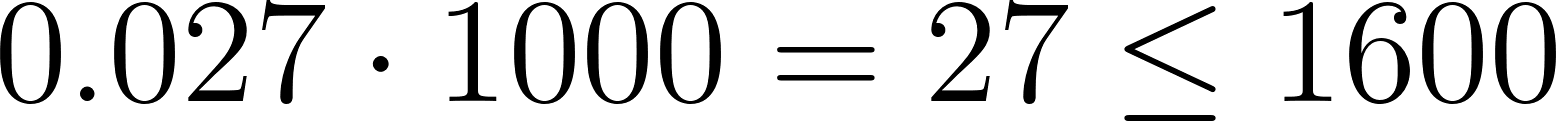
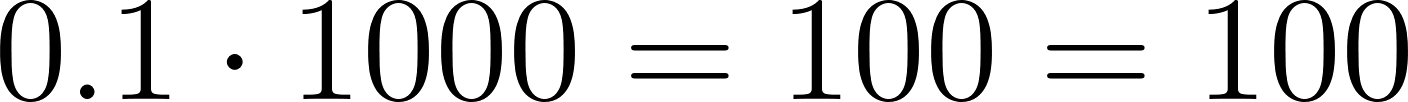
x = [1000. 0. 0. 2500. 300. 1573. 0.]

То есть:

* [](https://www.codecogs.com/eqnedit.php?latex=x_1%20%3D%201000#0) (сапоги)
* [](https://www.codecogs.com/eqnedit.php?latex=x_2%20%3D%200#0) (ботинки)
* [](https://www.codecogs.com/eqnedit.php?latex=x_3%20%3D%200#0) (ботильоны)
* s1..s4 показаны (s4 = 0 - подошвы исчерпаны)

Оптимальная выручка: Z = 900 000 руб.

Проверка ограничений при [](https://www.codecogs.com/eqnedit.php?latex=x_1%20%3D%201000%2C%20x_2%3Dx_3%3D0#0):

* кожа: [](https://www.codecogs.com/eqnedit.php?latex=0.2%5Ccdot1000%3D200%20%5Cle%202700#0)
* каблуки: [](https://www.codecogs.com/eqnedit.php?latex=0.5%5Ccdot1000%3D500%20%5Cle%20800#0)
* стельки: [](https://www.codecogs.com/eqnedit.php?latex=0.027%5Ccdot1000%3D27%20%5Cle%201600#0)
* подошвы: [](https://www.codecogs.com/eqnedit.php?latex=0.1%5Ccdot1000%3D100%20%3D%20100#0) - подошвы полностью использованы.