Прикладные модели оптимизации

Практическое занятие 1

```
import numpy as np
from sympy import Matrix, symbols, eye, zeros, ones,
diag, linsolve
import pyomo.environ as pyo


# 1. Матрицы в библиотеке numpy
print("Матрицы в numpy:")

# Создание матрицы
X = np.array([[3, 5, 7], [2, 4, 6], [1, 3, 5]])
print("Матрица X:\n", X)

# Единичная матрица
J = np.eye(4)
print("Единичная матрица:\n", J)

# Матрица-строка
L = np.array([13, 15, 17])
print("Матрица-строка L:\n", L)

# Транспонирование
L_T = L.T
print("Транспонированная матрица-строка:\n", L_T)

# Элементы матрицы
print("Элемент X[1,2]:", X[1,2])
print("Вторая строка матрицы X:", X[1])
print("Столбцы 1 и 2 матрицы X:\n", X[:, 0:2])

# Операции с матрицами
print("X + J[:3, :3]:\n", X + J[:3, :3])
print("2 * X:\n", 2 * X)

Y = np.array([[1, 0, 1], [0, 1, 0]])
print("Поэлементное умножение X и Y:\n", X[:2, :] * Y)

# Умножение матриц
W = np.array([[0, 1], [1, 0], [0, -1]])
Z = np.array([[2, 3], [4, 5]])
H = W @ Z
```

```
print("Умножение матриц W и Z:\n", H)

# Определитель и обратная матрица
detX = np.linalg.det(X)
print("Определитель матрицы X:", detX)

# Ранг матрицы
rankX = np.linalg.matrix_rank(X)
print("Ранг матрицы X:", rankX)

# 2. Матрицы в библиотеке sympy
print("\nМатрицы в sympy:")

# Создание матрицы
p, q, r, s, t, u = symbols('p q r s t u')
G = Matrix([[p, q, r], [s, t, u]])
print("Матрица G:\n", G)

# Единичная, нулевая и другие специальные матрицы
print("Единичная матрица:\n", eye(4))
print("Нулевая матрица:\n", zeros(3, 4))
print("Матрица из единиц:\n", ones(4, 3))
print("Диагональная матрица:\n", diag(2, 4, -3))

# Размер матрицы
print("Размер матрицы G:", G.shape)

# Элементы матрицы
print("Элемент G[0,2]:", G[0,2])
print("Первый столбец матрицы G:\n", G[:, 0:1])

# Удаление и вставка строк и столбцов
G.row_del(1)
print("Матрица G после удаления второй строки:\n", G)
K = Matrix([[10, 11, 12], [13, 14, 15]])
M = K.row_insert(1, Matrix([[16, 17, 18]]))
print("Матрица M после вставки строки:\n", M)

# Умножение матриц
N = Matrix([[2, 3], [4, 5]])
print("Умножение матриц N и N:\n", N * N)

# Транспонирование, определитель и обратная матрица
print("Транспонированная матрица N:\n", N.T)
```

```
print("Определитель матрицы N:", N.det())
print("Обратная матрица для N:\n", N.inv())

# Ранг матрицы
print("Ранг матрицы N:", N.rank())
#
# ЗАДАНИЕ 1
print("\nЗАДАНИЕ 1")
vectors = Matrix([
    [1, 3, 4, 5, 0],
    [4, -1, 0, 1, 2],
    [3, 2, 5, 5, 3],
    [-2, 0, 1, -1, 1],
    [4, 6, 7, 11, -1]
])

# Находим ранг матрицы, который также является
максимальным числом линейно независимых векторов
rank = vectors.rank()
print(f"Максимальное число линейно независимых векторов:
{rank}")

# ЗАДАНИЕ 2
print("\nЗАДАНИЕ 2")
vectors2 = Matrix([
    [1, 3, 4],
    [4, -1, 0],
    [3, 2, 5],
    [-2, 0, 1],
    [4, 6, 7]
])

# Находим базис системы векторов
basis = vectors2.T.columnspace()
print("Базис системы векторов:")
for vec in basis:
    print(vec)

# ЗАДАНИЕ 3
print("\nЗАДАНИЕ 3")
A = Matrix([
    [1, 2, 0, 9],
    [-3, 7, 1, 1],
    [-9, 4, 2, 5],
```

```
    [8, 4, 3, 1]
])

def Minor_elem(matrix, i, j):
    return matrix.minor(i, j)

minor_elem = Minor_elem(A, 3, 2)
print(f"Минор элемента a32: {minor_elem}")

# ЗАДАНИЕ 4
print("\nЗАДАНИЕ 4")
def Algeb_compl(matrix, i, j):
    minor = matrix.minor(i, j)
    return (-1)**(i+j) * minor

# Заданная матрица
A = Matrix([
    [1, 2, 0, 9],
    [-3, 7, 11, 5],
    [-9, 4, 25, 84],
    [3, 12, -5, 58]
])

# Вычисляем алгебраическое дополнение A32
algeb_compl_A32 = Algeb_compl(A, 3, 2)
print(f"Алгебраическое дополнение A32:
{algeb_compl_A32}")

# ЗАДАНИЕ 5
print("\nЗАДАНИЕ 5")
def Minor_Matrix(matrix, rows, cols):
    """Возвращает минор матрицы по заданным строкам и
столбцам."""
    return matrix.extract(rows, cols).det()
rows_indices = [0, 2] # [1, 3]
cols_indices = [2, 3] # [3, 4]
minor_matrix = Minor_Matrix(A, rows_indices,
cols_indices)
print(f"Минор, образованный 1-й и 3-й строками и 3-м и 4-
м столбцами:\n{minor_matrix}")

# ЗАДАНИЕ 6
print("\nЗАДАНИЕ 6")
# Заданная матрица
```

```
A = Matrix([
    [1, 3, 2, 4, 5],
    [0, 0, -1, 2, 7],
    [3, 9, 6, 12, 15],
    [5, 15, 9, 26, 22],
    [1, 3, 1, 10, 2]
])
def Basis_Minor(matrix):
    """Возвращает базисный минор и базисные строки и
столбцы матрицы."""
    rref_matrix, pivots = matrix.rref()
    basis_minor = rref_matrix.extract(pivots,
pivots).det()
    return rref_matrix.extract(pivots, pivots),
basis_minor

A_basis, M_basis = Basis_Minor(A)
print(f"Базисный минор: {M_basis}")
print(f"Базисные строки и столбцы:\n{A_basis}")

# ЗАДАНИЕ 7
print("\nЗАДАНИЕ 7")
# Матрица коэффициентов
A = np.array([
    [3, 2, 0],  # коэффициенты уравнения 3x + 2y = 2
    [1, -1, 0],  # коэффициенты уравнения x - y = 4
    [0, 5, 1]   # коэффициенты уравнения 5y + z = -1
])

# Вектор правой части системы
b = np.array([2, 4, -1])

# Решение системы
u = np.linalg.solve(A, b)
print(u)

# Проверка
print(np.allclose(np.dot(A, u), b))

# ЗАДАНИЕ 8
print("\nЗАДАНИЕ 8")

x1, x2, x3 = symbols('x1 x2 x3')
"""Функции, реализующие уравнения системы"""
```

```
y1 = x1 + x2 + 3*x3 - 18
y2 = 2*x1 - x2 + 9*x3 - 30
"""Решаем систему относительно переменных x1 и x2"""
print(linsolve([y1,y2], [x1,x2]))
from sympy import Matrix, symbols, linsolve

# ЗАДАНИЕ 9
print("\nЗАДАНИЕ 9")

# Определение переменных
x1, x2, x3 = symbols("x1 x2 x3")

# Матрица коэффициентов и вектор свободных членов
A1 = Matrix([
    [1, -2, 4],
    [1, -2, 1],
    [-3, 6, -12]
])
b1 = Matrix([6, 4, -18])

# Решение системы
solution1 = linsolve((A1, b1), x1, x2, x3)
print(solution1)

# ЗАДАНИЕ 10
print("\nЗАДАНИЕ 10")

# Определение переменных
x, y, z = symbols("x y z")

# Матрица коэффициентов и вектор свободных членов
A2 = Matrix([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 10]
])
b2 = Matrix([3, 6, 9])

# Решение системы
solution2 = linsolve((A2, b2), x, y, z)
print(solution2)

# 2 часть
```

```
model = pyo.ConcreteModel()
model.x_1 = pyo.Var(within=pyo.NonNegativeReals)
model.x_2 = pyo.Var(within=pyo.NonNegativeReals)
model.obj = pyo.Objective(expr=model.x_1 + 2*model.x_2)
model.con1 = pyo.Constraint(expr=3*model.x_1 +
4*model.x_2 >= 1)
model.con2 = pyo.Constraint(expr=2*model.x_1 +
5*model.x_2 >= 2)

N = [1,2]
M = [1,2]
c = {1:1, 2:2}
a = {(1,1):3, (1,2):4, (2,1):2, (2,2):5}
b = {1:1, 2:2}

model = pyo.ConcreteModel()
model.x = pyo.Var(N, within=pyo.NonNegativeReals)
def obj_rule(model):
    return sum(c[i]*model.x[i] for i in N)
model.obj = pyo.Objective(rule=obj_rule)
def con_rule(model, m):
    return sum(a[m,i]*model.x[i] for i in N) >= b[m]
model.con = pyo.Constraint(M, rule=con_rule)

model = pyo.AbstractModel()
model.N = pyo.Set()
model.M = pyo.Set()
model.c = pyo.Param(model.N)
model.a = pyo.Param(model.M, model.N)
model.b = pyo.Param(model.M)
model.x = pyo.Var(model.N, within=pyo.NonNegativeReals)
def obj_rule(model):
    return sum(model.c[i]*model.x[i] for i in model.N)
model.obj = pyo.Objective(rule=obj_rule)
def con_rule(model, m):
    return sum(model.a[m,i]*model.x[i] for i in model.N)
>= model.b[m]
model.con = pyo.Constraint(model.M, rule=con_rule)
```

Вывод:

```
/Users/andrey/Documents/PyCharm/pythonProject/bin/python /Users/andrey/Documents/PyCharm/pythonProject/main.py
Матрицы в numpy:
Матрица X:
 [[3 5 7]
 [2 4 6]
 [1 3 5]]
Единичная матрица:
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
Матрица-строка L:
 [13 15 17]
Транспонированная матрица-строка:
 [13 15 17]
Элемент X[1,2]: 6
Вторая строка матрицы X: [2 4 6]
Столбцы 1 и 2 матрицы X:
 [[3 5]
 [2 4]
 [1 3]]
X + J[:3, :3]:
 [[4. 5. 7.]
 [2. 5. 6.]
 [1. 3. 6.]]
2 * X:
 [[ 6 10 14]
 [ 4  8 12]
 [ 2  6 10]]
Поэлементное умножение X и Y:
 [[3 0 7]
 [0 4 0]]
Умножение матриц W и Z:
 [[ 4  5]
 [ 2  3]
 [-4 -5]]
```

```
Определитель матрицы X: 1.184237892933498e-15
Ранг матрицы X: 2

Матрицы в sympy:
Матрица G:
 Matrix([[p, q, r], [s, t, u]])
Единичная матрица:
 Matrix([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
Нулевая матрица:
 Matrix([[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]])
Матрица из единиц:
 Matrix([[1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1]])
Диагональная матрица:
 Matrix([[2, 0, 0], [0, 4, 0], [0, 0, -3]])
Размер матрицы G: (2, 3)
Элемент G[0,2]: r
Первый столбец матрицы G:
 Matrix([[p], [s]])
Матрица G после удаления второй строки:
 Matrix([[p, q, r]])
Матрица M после вставки строки:
 Matrix([[10, 11, 12], [16, 17, 18], [13, 14, 15]])
Умножение матриц N и N:
 Matrix([[16, 21], [28, 37]])
Транспонированная матрица N:
 Matrix([[2, 4], [3, 5]])
Определитель матрицы N: -2
Обратная матрица для N:
 Matrix([[-5/2, 3/2], [2, -1]])
Ранг матрицы N: 2

ЗАДАНИЕ 1
Максимальное число линейно независимых векторов: 3
```

```
ЗАДАНИЕ 2
Базис системы векторов:
Matrix([[1], [3], [4]])
Matrix([[4], [-1], [0]])
Matrix([[3], [2], [5]])

ЗАДАНИЕ 3
Минор элемента a32: 502

ЗАДАНИЕ 4
Алгебраическое дополнение A32: -1441

ЗАДАНИЕ 5
Минор, образованный 1-й и 3-й строками и 3-м и 4-м столбцами:
-225

ЗАДАНИЕ 6
Базисный минор: 0
Базисные строки и столбцы:
Matrix([[1, 0, 0], [0, 0, 1], [0, 0, 0]])

ЗАДАНИЕ 7
[ 2. -2.  9.]
True

ЗАДАНИЕ 8
{(16 - 4*x3, x3 + 2)}

ЗАДАНИЕ 9
{(2*x2 + 10/3, x2, 2/3)}

ЗАДАНИЕ 10
{(-1, 2, 0)}

Process finished with exit code 0
```