# Лабораторный практикум 5. Плоскость и прямая в пространстве

# (2 занятия)

## Уравнения плоскости в пространстве

Загрузите необходимые библиотеки:

import numpy as np

import matplotlib.pyplot as plt

import math

from sympy import \*

from matplotlib import cm

%matplotlib widget

Точки пространства задаются с помощью функции ***Plane****.*

1) Уравнение плоскости, проходящей через заданную точку с заданным нормальным вектором.

**Пример 1.** Найти уравнение плоскости, проходящей через точку (1,4,2) с нормальным вектором (6,6,6).

b = Plane(Point(1, 4, 2), normal\_vector=(6, 6, 6))

print(b.equation())

Если в качестве параметров указаны два массива из трех элементов, то первый массив задаст точку, через которую проводится плоскость, а второй – нормальный вектор:

b1=Plane((1, 4, 2),(6, 6, 6))

print(b1.equation())

2) Уравнение плоскости, проходящей через три точки.

**Упражнение 5.1.** Найти уравнение плоскости, проходящей через точки (1,1,2), (2,4,7) и (3,5,1) (в функции *Plane* необходимо задать три точки). Сделать проверку, используя пример 1.

## Функции, используемые при решении задач

3) Создание плоскости, параллельной данной и проходящей через заданную точку – метод ***.parallel\_plane*()**.

**Пример 2.** Плоскость с нормальным вектором (2,4,6) проходит через точку (1,4,6). Найти уравнение параллельной ей плоскости, проходящей через точку (2,3,5). Сравните уравнения плоскостей и сформулируйте условие параллельности двух плоскостей.

c = Plane(Point(1, 4, 6), normal\_vector=(2, 4, 6))

с1=c.parallel\_plane(Point3D(2, 3, 5))

print(c1.equation())

print(c.equation())

Полученные уравнения:

2\*x + 4\*y + 6\*z - 46

2\*x + 4\*y + 6\*z - 54

Так как отношения соответствующих координат нормальных векторов равны, то плоскости параллельны.

4) Создание плоскости, перпендикулярной данной и проходящей через две точки – метод ***.perpendicular\_plane*()**.

**Упражнение 5.2.** Найти уравнение плоскости, проходящей через точки (4,2,3), (2,0,1) и перпендикулярной плоскости . Проверить ортогональность нормальных векторов полученных плоскостей.

5) Нахождение расстояния от точки или до плоскости – метод .***distance*()**.

**Упражнение 5.3.** Найти расстояние от точки (1,2,3) до плоскости .

## Построение трехмерных графиков

Для построения 3D-графика функции в **Python** нужно задать саму функцию. Её можно задать с помощью **лямбда-функции**. **Лямбда-функция** — это краткий способ записи обычной функции в одну строчку.

В трёхмерном пространстве каждая точка задаётся тремя координатами, следовательно, в трёхмерном пространстве нужно два аргумента для задания функции:

Чтобы начать рисовать трехмерные поверхности в Python нужно сначала задать область построения с помощью *plt.figure*, которая принимает параметр , где *x* и *y* – ширина и высота рисунка в дюймах. В построенной области создается рисунок, в котором будут отображено трёхмерное пространство с координатными осями и сама поверхность. Функция *fig.add\_subplot*() разбивает область построения на клетки и задает в какой клетке рисовать трехмерный график. Команда *ax = fig.add\_subplot*(1,1,1, projection = '3d') разбивает область построения на две клетки и в первую клетку будет отображаться трехмерный график, благодаря аргументу projection = ‘3d’ . Затем вводятся области отображения функции для каждого аргумента: *xval = np.linspace*(), *yval = np.linspace*() и создается прямоугольная сетка на плоскости XY, в узлах которой будут рассчитываться значения отображаемой функции – значения по оси Z. Для создания такой сетки используется функция *np.meshgrid*(). Чтобы создать поверхность, которая будет отображаться на рисунке, используется *surf = ax.plot\_surface(x, y, z, rstride =, cstride = , cmap=cm.plasma*), где *x* и *y* – принимаемые аргументы, *z* – получаемая функция, *rstride* и *cstride*отвечает за шаг прорисовки поверхности, чем меньше будут эти значения, тем более плавно будет выглядеть градиент на поверхности. С помощью *cm.plasma* поверхность будет отображаться с цветовой схемой *plasma*. Существуют другие цветовые схемы, такие как *viridis,* *magma, rainbow* и т.д. Метод ***set\_alpha*** позволяет сделать поверхность прозрачной.

К сожалению, трёхмерную картинку нельзя вертеть мышкой. Но можно задать, с какой стороны мы смотрим. Угол обзора можно изменить с помощью метода ***view\_init*()**.

**Пример 3**. Построить плоскость, заданную общим уравнением . В качестве заголовка задать общее уравнение плоскости, изобразить нормальный вектор.

# параметры плоскости

A = 3; B = 4; C = -4; D = -12

f = lambda x, y: (-A \* x – B \* y – D) / C # строим плоскость

fig = plt.figure(figsize=(10, 10)) # создаём холст

ax = fig.add\_subplot(1, 1, 1, projection=’3d’) #

xval = np.linspace(-4, 4, 100) # разбиение для x

yval = np.linspace(-4, 4, 100) # разбиение для y

x, y = np.meshgrid(xval, yval)

p = np.array([0, 0, - 3]) # точка на плоскости

v = np.array([3, 4, -4]) # вектор нормали

ax.quiver(\*p, \*v, color=’Blue’) # строим вектор нормали

z = f(x, y) # вычисляем значения z в точках

surf = ax.plot\_surface(x, y, z, rstride=10, cstride=10, cmap=cm.plasma) # строим плоскость

plt.xlabel(‘x’)

plt.ylabel(‘y’)

plt.title(‘Плоскость 3x + 4y – 4z – 12 = 0’)

ax.view\_init(10, -30)

plt.show()

**Упражнение 5.4.** Построить плоскость, проходящую через точку *М*(1,-1,-3) и имеющую нормальный вектор ***n***(2,-3,5). Найти расстояние от начала координат до данной плоскости. Вывести в заголовок графика общее уравнение плоскости. Изобразить нормальный вектор и данную точку.

**Упражнение 5.5.** Построить плоскость, проходящую через точку *М*(2,3,-4) и параллельно векторам ***a***(-3,2,-1) и ***b***(0,3,1). Найти расстояние от точки *N*(-3,1,3) до данной плоскости. Вывести в заголовок графика общее уравнение плоскости. Изобразить нормальный вектор и данные точки. Указать координаты нормального вектора.

## Уравнения прямой в пространстве

1. Общее уравнение прямой – функция ***Line***.

**Пример 4.** Найти уравнение прямой, проходящей через точки (1,0,0) и (5,3,2). Сделать рисунок.

# прямая, проходящая через две точки р1 и р2

pl, р2 = Point(l,0,0), Point(5,3,2)

L1 = Line(pl, р2)

L1=L1.equation()

print('Общее уравнение прямой:',L1)

Построение прямой, проходящей через две точки

# построение прямой по двум точкам (1 0 0) и (5 3 2)

ax = plt.axes(projection ='3d')

x = np.array([1, 5])

y = np.array([0, 3])

z = np.array([0, 2])

ax.plot(x, y, z)

plt.show()

Можно найти уравнение прямой в пространстве по точке A и направляющему вектору q:

Line(A, direction\_ratio = q)

1. Параметрические уравнения прямой – **метод *.arbitrary\_point*().**

L2 = Line(p1, р2)

L2=L2.arbitrary\_point()

print('Параметрические уравнения прямой:',L2)

## Функции, используемые при решении задач

3) Расстояние от точки до прямой – метод ***.distance*()** (аналогично двумерному случаю).

4) Перпендикуляр к плоскости, проходящий через заданную точку – метод .***perpendicular\_line*().**

**Упражнение 5.6.** Найти уравнение прямой, проходящей через точку (9,8,7) перпендикулярно плоскости .

Найти векторное произведение нормалей полученных плоскостей. Как оно связано с уравнением плоскости?

5) Угол между прямой и плоскостью – метод ***.angle\_between*()**.

**Пример 5.** Найти угол между прямой и плоскостью . Проверить результат с помощью свойства скалярного произведения.

Из параметрического уравнения прямой видно, что прямая проходит через точку (3, 6, –7) и имеет направляющий вектор (1, 1, –2). Плоскость имеет нормальный вектор (4, –2, –2) и проходит, например, через точку (0, 0, –1.5).

a = Plane (Point(0,0,-1.5), normal\_vector=(4,-2,-2))

b = Line(Point(3,6,-7), direction\_ratio=np.array([1,1,-2]))

print(a.angle\_between(b))

Проверка: найдем арксинус модуля скалярного произведения нормального вектора плоскости и направляющего вектора прямой:

n = np.array([4, -2, -2])

q = np.array([-1, -1, 2])

sin\_angle=np.dot(n, q) / np.linalg.norm(n) / np.linalg.norm(q)

alfa= np.arcsin(sin\_angle)

a=np.abs(alfa) # арксинус угла

print ('Угол между n и q в градусах:', math.degrees(a))

6)Пересечение прямой и плоскости или пересечение плоскостей – метод **.*intersection*()**.

**Пример 6**. Общее уравнение прямой преобразовать к каноническому виду и определить величины углов, образованных этой прямой с координатными осями. Построить эту прямую. Изобразить точку *М*, принадлежащую этой прямой и направляющий вектор. Вывести заголовок графика (полученное уравнение).

Для решения данной задачи надо знать какую-либо точку прямой и ее направляющий вектор. Выберем точку на прямой следующим образом: положим, например, , тогда для определения остальных координат этой точки, получим систему уравнений

которую решаем с помощью обратной матрицы:

a = np.array([[1, 2], [2, - 2]])

b = np.array([-2, 5])

x = np.linalg.inv(a).dot(b)

print(x)

[1. -1.5]

Точка *М* имеет координаты (1, -1.5, 0).

Или можно использовать метод .*intersection*(). Точки на плоскостях подобрали:

0;3;В качестве направляющего вектора возьмем вектор ***q*** – векторное произведение нормальных векторов данных плоскостей:

n1 = np.array([1, 2, -3])

n2 = np.array([2, -2, 1])

q = np.cross(n1, n2)

print(q)

[-4 -7 -6]

Искомое уравнение прямой:

или .

Направление прямой задает вектор ***q***(4,7,6). Он образует с координатными осями углы – соответственно. Находим эти углы по известным формулам

.

qq = np.linalg.norm(q) # модуль вектора q

a = q[0] / qq

b = q[1] / qq

c = q[2] / qq

print(a, b, c)

0.39801487608399566 0.6965260331469925 0.5970223141259935

Заметим, для контроля, что равенство выполняется:

print(a \*\* 2 + b \*\* 2 + c \*\* 2)

Строим график. Построим обе плоскости, прямую их пересечения и направляющий вектор. Прямую можно провести через две точки с помощью функции *plot* (обратите внимание, на вход она принимает координаты X, Y, Z (начало и СМЕЩЕНИЕ), то есть конечная координата – это начальная плюс координата направляющего вектора.

# коэффициенты первой плоскости

A1 = 1; B1 = 2; C1 = -3; D1 = 2

#коэффициенты второй плоскости

A2 = 2; B2 = -2; C2 = 1; D2 = -5

f1 = lambda x, y: (- A1 \* x – B1 \* y – D1) / C1

f2 = lambda x, y: (- A2 \* x – B2 \* y – D2) / C2

fig = plt.figure(figsize=(10, 10))

ax = fig.add\_subplot(1, 1, 1, projection=’3d’)

xval = np.linspace(-4, 4, 100)

yval = np.linspace(-4, 4, 100)

x, y = np.meshgrid(xval, yval)

z1 = f1(x, y)

surf = ax.plot\_surface(x, y, z1, rstride=10, cstride=10, cmap=cm.plasma).set\_alpha(0.3)

z2 = f2(x, y)

surf = ax.plot\_surface(x, y, z2, rstride=10, cstride=10, cmap=cm.plasma).set\_alpha(0.3)

p = np.array([1, -1.5, 0]) # точка на прямой

v = np.array([4, 7, 6]) # направляющий вектор

ax.quiver(\*p, \*v/2, color=’Red’) # строим вектор

# строим прямую, внимание на координаты

ax.plot([1, 5], [-1.5, 5.5], [0, 6],'--',color=’Blue’)

ax.plot([1, -3], [-1.5, -8.5], [0, -6],'--',color=’Blue’)

plt.xlabel(‘x’)

plt.ylabel(‘y’)

plt.show()

**Упражнение 5.7.** Найти параметрические уравнения прямой, проходящей через точку *М*(2, -1, -3) параллельна прямой

Проверить ортогональность нормального вектора плоскости и направляющего вектора прямой. Найти уравнение плоскости, содержащей эти прямые. Сделать рисунок.

**Упражнение 5.8.** Найти величину острого угла между прямыми

и .

Ответ записать в градусах.

**Упражнение 5.9.** Найти расстояние между параллельными прямыми с помощью функции *distance*. Сделать проверку, используя свойство векторного произведения.

и .

Функция в Python определяется с помощью оператора ***def*** (оператор определения функции). Например, функция, возвращающая сумму двух объектов, задается так:

In[ ] : def v(x,y):

return x+y

Инструкция ***return*** говорит, что нужно вернуть значение:

In[ ] : v(3, 5)

Out[ ]: 8

## Дополнительное задание

Создать функцию, которая находит расстояние между прямыми *L*1 и *L*2, в том числе и скрещивающимися, используя следующий алгоритм:

1) провести через точку *А* прямой *L*1 плоскость *Р*, перпендикулярную *L*1;

2) найти проекцию *PL*2 прямой *L*2 на плоскость *Р*;

3) найти расстояние *d* от точки *А* до прямой *PL*2.

Число *d* и будет искомым расстоянием между прямыми.

Проверьте работу функции на примере упражнения 5.9.