УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

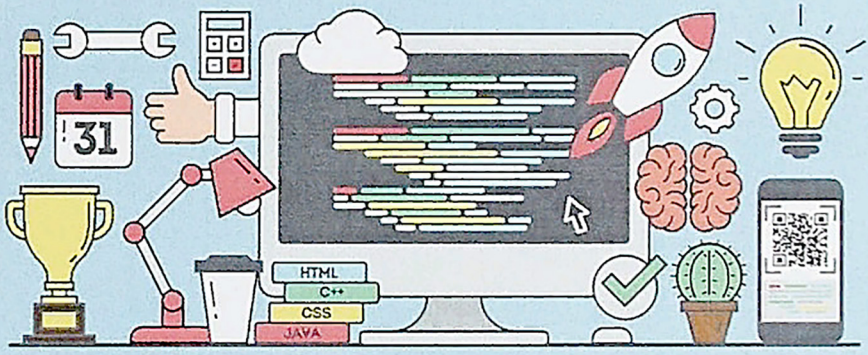
100 ЛЕТ
ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ
РОССИЙСКОЙ ФЕДЕРАЦИИ

А. С. Балджы
М. Б. Хрипунова
И. А. Александрова

МАТЕМАТИКА НА PYTHON

Часть 1

ЭЛЕМЕНТЫ ЛИНЕЙНОЙ АЛГЕБРЫ И АНАЛИТИЧЕСКОЙ ГЕОМЕТРИИ



ИЗДАТЕЛЬСТВО Прометей

Федеральное государственное образовательное бюджетное
учреждение высшего образования
«ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ
РОССИЙСКОЙ ФЕДЕРАЦИИ»

А. С. БАЛДЖЫ М. Б. ХРИПУНОВА
И. А. АЛЕКСАНДРОВА

МАТЕМАТИКА НА PYTHON. ЧАСТЬ I

ЭЛЕМЕНТЫ ЛИНЕЙНОЙ АЛГЕБРЫ И АНАЛИТИЧЕСКОЙ ГЕОМЕТРИИ

Учебно-методическое пособие

ИЗДАТЕЛЬСТВО Прометей

Москва 2018

УДК 512.64(075)
ББК 22.134я73
Б 20

Рецензент:

Золотарева Е. Л., кандидат экономических наук, старший преподаватель Департамента анализа данных, принятия решений и финансовых технологий Финансового университета при Правительстве Российской Федерации.

Балджы Анна Сергеевна
Хрипунова Марина Борисовна
Александрова Ирина Александровна

Б20 **Математика на Python. Часть I. Элементы линейной алгебры и аналитической геометрии:** *учебно-методическое пособие* / А. С. Балджы, М. Б. Хрипунова, И. А. Александрова; Финансовый университет при Правительстве РФ. — М.: Прометей, 2018. — 76 с.

Дисциплина «Компьютерный практикум» является обязательной дисциплиной базовой части профессионального цикла ООП по направлению 38.03.01 «Экономика» всех профилей (модуль математики и информатики Б.1.1.2.3.). Изучение данной дисциплины нацелено на формирование у слушателей практических навыков по реализации математических методов и моделей, применяемых в профессиональных задачах, с помощью компьютерных вычислений. В учебном пособии представлены задачи по высшей математике и их реализация на языке *Python*.

Соответствует требованиям Федерального государственного образовательного стандарта высшего образования последнего поколения.

Учебно-методическое пособие предназначено для проведения занятий по дисциплине «Компьютерный практикум» для студентов, обучающихся по направлениям подготовки 38.03.01 «Экономика» и 38.03.02 «Менеджмент» (уровень бакалавриата) в Финансовом университете при Правительстве Российской Федерации, а также в других образовательных организациях высшего образования.

ISBN 978-5-907003-86-6

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Владение компьютерными математическими методами анализа количественных данных является необходимым условием достижения целей, стоящих перед экономистом. Многие профессиональные навыки, формируемые в ходе изучения дисциплин математического блока, могут быть наилучшим образом развиты с применением информационных технологий. Для решения классических математических задач могут быть использованы различные современные средства обработки данных — специальные программы и пакеты. Язык Python — это высокоуровневый объектно-ориентированный язык программирования, ориентированный на обработку больших данных. Синтаксис языка минималистичен, а библиотеки содержат много полезных для применения функций. Язык постоянно развивается, на настоящий момент существует две версии: 2 и 3. В последнее время он становиться стандартом применения в различных разделах анализа данных.

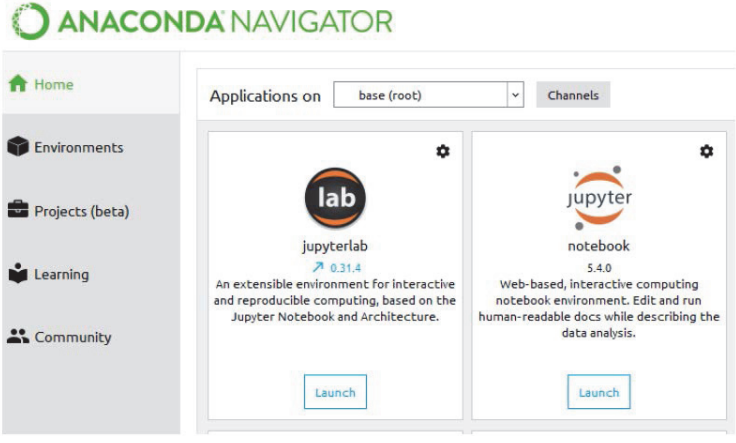
Особенностью настоящего пособия является то, что оно содержит большое количество практических примеров использования языка Python для решения математических задач. Каждая тема включает примеры решения типовых задач и задачи для самостоятельного решения. Учебное пособие логически связно с учебником [1] и фактически представляет собой сборник задач и кейсов к этому учебнику.

Учебное пособие содержит материалы по дисциплинам «Математика» и «Компьютерный практикум», и может быть использовано студентами, магистрантами и аспирантами, а как же всеми интересующимися владеющими начальными навыками программирования и знаниями по линейной алгебре.

Авторы благодарят рецензента Золотареву Екатерину Леоновну за сделанные замечания и внесенные предложения.

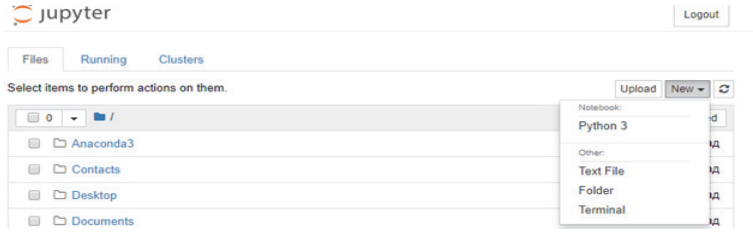
ТЕМА1. УСТАНОВКА PYTHON В СОСТАВЕ ANACONDA ДЛЯ WINDOWS. ЗНАКОМСТВО С PYTHON. ОСНОВНЫЕ КОМАНДЫ. СИНТАКСИС

Существует несколько способов установки Python на компьютер (ноутбук). Предлагаем поступить следующим образом: установим на компьютер дистрибутив Anaconda, удобный для запуска программы и изучения языка. Этот пакет включает в себя интерпретатор языка Python (есть версии 2 и 3, мы установим 3, 64 bit) и набор наиболее часто используемых библиотек. Например, если у вас установлена ОП Windows, с сайта https://www.anaconda.com/download/ нужно скачать установочный файл и следовать указаниям. Вы можете ознакомиться с лицензией и указать папку для установки. Отметим, что установка возможна если в названии вашего профиля на компьютере только латинские буквы, иначе нужно будет создать новый профиль с латинским названием и потом установить Anaconda. После установки, активируем программу, откроется Анаконда навигатор.

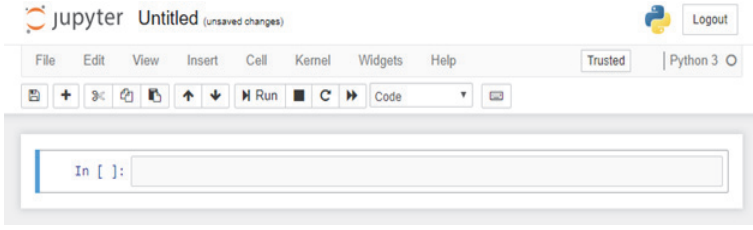


Нажмем на jupyter, в результате получим на экране следующее:

Создадим папку для нашего документа. Для этого нажмем на New справа на экране и выберем Folder. Поставим галочку напротив имени папки “Untitled folder” и нажмем Rename, назовем папку “Python, hi”.



Зайдем в эту папку, и создадим в ней ноутбук, для этого нажмем на New и выберем Python 3. Все, можем работать. Код или текст будем вводить в строки, показанные ниже.



Если это код, то на панели инструментов нужно выбрать “Code”, если текст, то “Markdown”. Если мы хотим написать пояснения в одной строчке с кодом, введем # и потом текст пояснения.

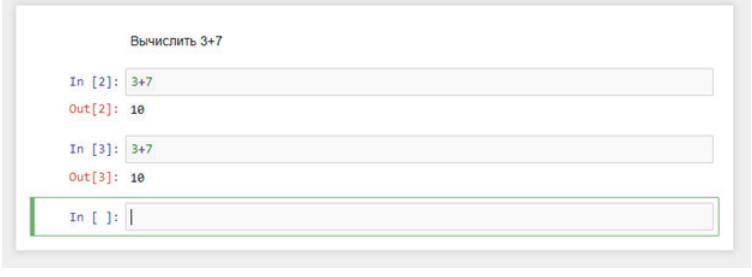
Для запуска программы используйте сочетание клавиш Ctrl+Enter или Shift+Enter, или нажмите на значок 

.

Python поддерживает набор самых обычных математических операций, что позволяет использовать его как калькулятор.

Сложение	a+b
Вычитание	a–b
Умножение	a*b
Деление	a/b
Неполное частное от деления	a//b
Остаток от деления	a%b
Изменение знака числа	–a
Модуль числа	abs(a)
Возведение в степень	a**b

Например, вычислим 3+7. Для этого выставим “Code”, введем в ячейку 3+7 и нажмем Ctrl+Enter или Shift+Enter. В первом случае мы получим 10 под строчкой ввода, во втором будет создана новая ячейка ниже первоначальной.



ОПЕРАТОРЫ PYHTON

Перечислим основные операторы:

1. if — условный оператор
2. Else -последовательность действий которая выполняется если условие в блоке if ложно
3. while, for — операторы цикла
4. def — оператор определения функции
5. return -возврат функции
6. pass — оператор ничего не делает. Используется для пустых блоков кода

ИМЕННЫЕ ФУНКЦИИ, ИНСТРУКЦИИ DEF И RETURN, PASS, IMPORT

Функция определяется с помощью оператора **def**.

Например, функция, возвращающая сумму двух объектов, задается так

```
In [7]:  1 def v(x,y):
         2     return x+y
```

Инструкция **return** говорит, что нужно вернуть значение. В нашем случае функция возвращает сумму x и y как числовых величин. Например, если сумма двух чисел 10 и 1 равна 11 (является числом):

```
In [8]:  1  v(10,1)
Out[8]: 11
```

Интересно, что операцию сложения можно осуществить и при нечисловых значениях. Например, к строке “abcd” прибавить подстроку “e”, в результате получим новую строку:

```
In [9]:  1 v('abcd','e')
Out[9]: 'abcde'
```

Отметим, что функция может и не заканчиваться инструкцией return, при этом функция вернет значение None, то есть ничего не выведется:

```
In [8]: def func():
            pass
        print(func())
```

```
None
```

где в качестве тела приведенной выше функции используется «пустой оператор» pass.

Импортировать файлы с программами, называемые библиотеками (модулями), позволяет инструкция import. Подробнее остановимся на ней и ее синтаксисе в разделе библиотеки.

АРГУМЕНТЫ ФУНКЦИИ

Аргументы функции указываются в скобках и обозначаются буквами. При вызове функции из основной программы аргументам придаются конкретные значения в зависимости от типа аргументов в описании функции. Ниже приведен пример функции, вычисляющей произведение трех чисел. При этом аргумент с является необязательным, он задан изначально равным 4, если аргумент с в вызове отсутствует, он считается равным 4, если же он присутствует, то в функцию передается указанное значение.

INPUT, PRINT, УСЛОВНЫЕ КОНСТРУКЦИИ, ЦИКЛЫ

Для ввода данных в программу используется функция **input()**. Она считывает одну строку. Для того, чтобы вывести значения на экран используется функция **print()**. Проиллюстрируем на примере.

```
In [12]: def func(a,b,c=4): # c-необязательный аргумент
             return a*b*c
         func(1,2) # c=4 по умолчанию

Out[12]: 8

In [13]: func(1,2,3) # c=3

Out[13]: 6

In [15]: func(a=1,b=3) # c=4

Out[15]: 12

In [16]: func(a=3,c=6) #b не определен

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-55e55c02b760> in <module>()
----> 1 func(a=3,c=6) #b не определен

TypeError: func() missing 1 required positional argument: 'b'
```

Чтобы ввести данные с клавиатуры в программу используют функцию input. Считаем два числа и сохраним их в переменные a и b, пользуясь оператором присваивания =. Затем найдем сумму этих чисел.

```
In [*]: 1 a = input()
        2 b = input()
        3 s = a + b
        4 print(s)
        5
```

Если вы попробуете запустить данную программу и введете для примера, числа 4 и 5, результат будет таким:

```
In [10]: 1 a = input()
         2 b = input()
         3 s = a + b
         4 print(s)
         5

4
5
45
```

То есть, данная программа находит сумму не двух чисел 4 и 5, а складывает две строки “4” и “5”, в результате получается 45. Чтобы посчитать сумму чисел 4 и 5 нужно использовать функцию преобразования типов int, которая “делает” из строки число:

```
In [*]:
1 a = int(input())
2 b = int(input())
3 s = a + b
4 print(s)
5
```

В результате получим правильный результат:

```
In [11]:
1 a = int(input())
2 b = int(input())
3 s = a + b
4 print(s)
5

4
5
9
```

Условная конструкция if-elif-else (если-иначе оператор ветвления) выбирает, какое действие следует выполнить, в зависимости от значения переменных в момент проверки условия, которое следует за словом if. Синтаксис условной конструкции if таков: if(если) условное выражение: вывод, дальше необязательные части elif условные выражения: и else:

В Питоне принято использовать укороченную форму проверки условия

`if a:` вместо `if a == True :`, Elif –не обязательная часть, следует за if и показывает альтернативное условие, которое требуется проверить. Далее else: тоже не обязательное, показывает, что будет, если не if не elif не выполняются.

Например, с клавиатуры вводится два числа — координаты точки на плоскости. Требуется определить к какой координатной четверти принадлежит данная точка. В примере мы будем использовать операторы print, input и условную конструкцию:

In [12]:
```
x = int(input())
y = int(input())
if x > 0 and y > 0:
    print("Первая четверть")
elif x > 0 and y < 0:
    print("Четвертая четверть")
elif y > 0:
    print("Вторая четверть")
else:
    print("Третья четверть")
```

В результате, введя, например, числа 5 и -7 получим:

In [13]:
```
x = int(input())
y = int(input())
if x > 0 and y > 0:
    print("Первая четверть")
elif x > 0 and y < 0:
    print("Четвертая четверть")
elif y > 0:
    print("Вторая четверть")
else:
    print("Третья четверть")

```

```
5
-7
Четвертая четверть
```

Далее рассмотрим пример на «print»

In [14]:
```
print(3 + 5)
print(2 * 8, (16 - 1) * 5)
print(3 ** 2)  # ** --  означают возведение в степень
print(38 / 5)  # /— это деление с ответом-дробью
print(33 // 3)  # //  считает частное от деления нацело
print(47 % 5)  # процент считает остаток от деления нацело

```

Результатом выполнение данной программы будет выведенная на экран последовательность чисел

In [14]:
```
print(3 + 5)
print(2 * 8, (16 - 1) * 5)
print(3 ** 2)  # ** --  означают возведение в степень
print(38 / 5)  # /— это деление с ответом-дробью
print(33 // 3)  # //  считает частное от деления нацело
print(47 % 5)  # процент считает остаток от деления нацело

```

```
8
16 75
9
7.6
11
2
```

Циклические конструкции используются, когда нам нужно повторить какое-либо действие несколько раз.

Существуют две возможные реализации цикла в Python: while и for. Мы будем использовать цикл for. Если нужно указать некоторый диапазон чисел, удобно воспользоваться функцией range(). Рассмотрим пример

```
In [15]:
1 for i in range(5):
2     print(i)
3
```

Данный фрагмент кода выводит на экран числа в диапазоне от 0 до 4. Отметим, что на Python нумерация начинается с нуля, соответственно 5ым значением в нашем примере будет 4. диапазоне. Переменной i в цикле for последовательно передаются значения из заданного диапазона, и они выводятся на экран:

```
In [15]:
1 for i in range(5):
2     print(i)
3

0
1
2
3
4
```

БИБЛИОТЕКИ PYTHON

Библиотека Math

Для вычислений с действительными числами применяется библиотека Math, которая содержит много полезных функций. Ниже приведены основные функции, которые включены в эту библиотеку.

math.ceil(X)	— округление до ближайшего большего числа.
math.fabs(X)	— модуль X.
math.factorial(X)	— факториал числа X.
math.floor(X)	— округление вниз.
math.fmod(X, Y)	— остаток от деления X на Y.

math.fsum	— сумма всех членов последовательности
math.isfinite(X)	— является ли X числом.
math.isinf(X)	— является ли X бесконечностью.
math.modf(X)	— возвращает дробную и целую часть числа X. Оба числа имеют тот же знак, что и X.
math.trunc(X)	— усекает значение X до целого.
math.exp(X)	— e^X.
math.expm1(X)	— e^X — 1
math.log(X, [base])	— логарифм X по основанию base. Если base не указан, вычисляется натуральный логарифм.
math.log10(X)	— логарифм X по основанию 10.
math.log2(X)	— логарифм X по основанию 2.
math.pow(X, Y)	— X^Y.
math.sqrt(X)	— квадратный корень из X.
math.acos(X)	— арккосинус X. В радианах.
math.asin(X)	— арксинус X. В радианах.
math.atan(X)	— арктангенс X. В радианах.
math.cos(X)	— косинус X (X указывается в радианах
math.sin(X)	— синус X (X указывается в радианах
math.tan(X)	— тангенс X (X указывается в радианах
math.hypot(X, Y)	— вычисляет гипотенузу треугольника с катетами X и Y (math.sqrt(x * x + y * y
math.degrees(X)	— конвертирует радианы в градусы.
math.radians(X)	— конвертирует градусы в радианы.
math.pi — pi = 3,1415926…	Число ϖ
math.e — e = 2,718281…	Число e

Для подключения библиотеки используем **import.** Загрузим библиотеку **math** и выведем число e.

```
import math
math.e
```

```
2.718281828459045
```

Можно это сделать, используя псевдоним библиотеки. Например, так

```
import math as m
m.e
```

```
2.718281828459045
```

Библиотека Mathplotlib

Библиотека matplotlib — это набор методов для создания двумерной графики для языка программирования python. Эта библиотека включает множество различных методов, мы остановимся только на некоторых из них. Графика в Mathplotlib представляет собой отдельную тему, подробно рассмотренную, например, в пособии [12]. Остановимся подробней на ней при рассмотрении кривых II порядка.

Библиотека Sympy

Библиотека Sympy предназначена для символьных вычислений. В нее входят библиотеки для работы с матрицами, с векторами, прямыми и плоскостями, для решения статистических и теоретико-вероятностных задач.

ОБРАБОТКА МАТРИЦ В PYTHON. БИБЛИОТЕКА NUMPY

Ниже мы рассмотрим основные операции над матрицами на языке Python.

Напомним, что матрицей размера m x n называется таблица чисел, содержащая m строк и n столбцов. Числа, составляющие матрицу, называются элементами матрицы и в общем виде обозначаются где i — номер строки, j — номер столбца. Определения и операции матричной алгебры подробно описаны, например, в [1], глава 1.

Для работы с матричной алгеброй в Python разработано специальное расширение языка Python — это библиотека NumPy. В программировании матрицы принято называть двумерными массивами, векторы одномерными массивами, возможна также работа с массивами размерности больше двух. Наиболее важным объектом NumPy является ndarray (однородный массив).

СОЗДАНИЕ МАТРИЦ

Рассмотрим один из способов создания матрицы в NumPy. Используем функцию numpy.array() пита numpy. ndarray . При этом в квадратных скобках, через запятую указываются строки матрицы. Вся последовательность строк также заключается в квадратные скобки, а аргумент функции указывается в круглых скобках. Например, в результате выполнения следующего кода будет создан вектор (одномерный массив) из трех элементов 1,2,3.

```
In [13]: import numpy as np
         a = np.array([[1, 2, 3]])
         a

Out[13]: array([[1, 2, 3]])
```

Ниже приведен пример создания матрицы b размером 2x3.

```
In [11]: b=np.array([[1.5,2,3],[4,5,6]])
         b

Out[11]: array([[1.5, 2. , 3. ],
                [4. , 5. , 6. ]])
```

Чтобы посмотреть какого типа объект можно использовать функцию type.

```
type(a)
```

```
Out[14]: numpy.ndarray
```

Элементами матрицы могут быть величины различных типов. В этом случае тип элементов должен быть указан как параметр после определения всех элементов матрицы. Так, ниже рассмотрен пример матрицы, состоящей из комплексных чисел, тип данных np.complex.

```
In [17]: b=np.array([[1.5,2,3],[4,5,6]],dtype=np.complex)
         b

Out[17]: array([[1.5+0.j, 2. +0.j, 3. +0.j],
                [4. +0.j, 5. +0.j, 6. +0.j]])
```

Функция array() не единственная функция для создания массивов.

Обычно на практике элементы массива вначале неизвестны, а массив, в котором они будут храниться, уже нужен. Поэтому имеется несколько функций для того, чтобы создавать массивы с каким-то исходным содержимым (по умолчанию тип создаваемого массива — float64).

Функция zeros() создает массив из нулей, а функция ones() — массив из единиц. Обе функции принимают в качестве аргументов размеры матрицы, и необязательный аргумент dtype — тип элементов матрицы:

```
In [21]: np.zeros((3,5))

Out[21]: array([[0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.]])
```

```
In [22]: np.ones((2,2,2))

Out[22]: array([[[1., 1.],
                 [1., 1.]],

                [[1., 1.],
                 [1., 1.]]])
```

Функция eye() создаёт единичную матрицу (двумерный массив)

```
In [23]: np.eye(5)

Out[23]: array([[1., 0., 0., 0., 0.],
                [0., 1., 0., 0., 0.],
                [0., 0., 1., 0., 0.],
                [0., 0., 0., 1., 0.],
                [0., 0., 0., 0., 1.]])
```

Функция empty() создает матрицу нужного размера, исходное содержимое формируется случайно. Например, при создании матрицы размером 3X3, получаемые значения могут быть такими:

```
In [24]: np.empty((3,3))

Out[24]: array([[2.37663529e-312, 2.14321575e-312, 2.37663529e-312],
                [2.56761491e-312, 8.48798317e-313, 9.33678148e-313],
                [8.70018275e-313, 2.02566915e-322, 2.47032823e-323]])
```

Для создания последовательностей чисел, в NumPy имеется функция arange(). Она формирует последовательность чисел от начального до конечного значения, включая первое и исключая последнее, с некоторым шагом. Например, последовательность чисел от 10 до 30 с шагом 5 будет иметь следующий вид:

```
In [27]: np.arange(10,30,5)

Out[27]: array([10, 15, 20, 25])
```

А последовательность чисел от 0 до 1 с шагом 0,1 будет иметь следующий вид:

```
In [26]: np.arange(0,1,0.1)

Out[26]: array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

Обратим внимание, что числа имеющие дробную часть записываются через точку, например, 0.1.

С официальной документацией по NumPy можно познакомиться по ссылке (на английском языке) https://docs.scipy.org/doc/numpy/reference/.

Далее рассмотрим решение задач линейной алгебры на Python.

План изложения следующий: формулировка задачи, решение ее в Python, примеры задач для самостоятельного решения.

ТЕМА 2. ЭЛЕМЕНТЫ ЛИНЕЙНОЙ АЛГЕБРЫ. ПРИМЕРЫ РЕШЕНИЯ ЗАДАЧ

ОПЕРАЦИИ НАД МАТРИЦАМИ И ИХ СВОЙСТВА

Ввод и вывод матрицы (пакет Numpy)

1. Ввести матрицу $A_{2\times3}=\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}$

Решение

```
import numpy as np
A=np.array([[1,2,3],[4,5,6]])
print (A)
[[1 2 3]
[4 5 6]]
```

2. Ввести матрицу $B_{2\times3}=\begin{pmatrix}10 & 20 & 30\\ 40 & 50 & 60\end{pmatrix}$

Решение

```
B=np.array([[10,20,30],[40,50,60]])
print (B)
[[10 20 30]
[40 50 60]]
```

3. Найти сумму A+B, заданных в 1. и 2.

Решение

```
print (A+B)
[[11 22 33]
[44 55 66]]
```

4. Найти 5A, заданной в 1.

Решение

```
print (5*A)
[[ 5 10 15]
[20 25 30]]
```

5. Найти A-B, заданных в 1. и 2.

Решение

```
print (A–B)
[[ -9 -18 -27]
[-36 -45 -54]]
```

Найти B-A, заданных в 1. и 2.

Решение

```
print (B–A)
[[ 9 18 27]
[36 45 54]]
```

6. Ввести матрицу $B_{3\times4}=\begin{pmatrix}10 & 20 & 30 & 40\\ 50 & 60 & 70 & 80\\ 90 & 100 & 110 & 120\end{pmatrix}$

Решение

```
matrix2=np.array([[10,20,30,40],[50,60,70,80],
[90,100,110,120]])
print(matrix2)
[[ 10 20 30 40]
[ 50 60 70 80]
[ 90 100 110 120]]
```

Умножение матриц

Для вычисления произведения двух матриц A и B применяется функция np.dot(A,B)

7. Умножить $A_{2\times3}=\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}$ на $B_{3\times4}=\begin{pmatrix}10 & 20 & 30 & 40\\ 50 & 60 & 70 & 80\\ 90 & 100 & 110 & 120\end{pmatrix}$

Решение

```
A=np.array([[1,2,3],[4,5,6]])
B=np.array([[10,20,30,40],
[50,60,70,80],[90,100,110,120]])
print (np.dot(A,B))
```

[[380 440 500 560]

[830 980 1130 1280]]

8. Проверить свойство некоммутативности умножения матриц

$$\underset{2\times2}{A}=\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}, \underset{2\times2}{B}=\begin{pmatrix}5 & 1\\ -1 & 0\end{pmatrix}$$

Решение

```
A=np.array([[1,2],[3,4]])
B=np.array([[5,1],[-1,0]])
print(np.dot(A,B))
print (np.dot(B,A))
```

[[3 1]

[11 3]]

[[8 14]

[-1 -2]]

Видим, что в общем случае умножение матриц не коммутативно, то есть A*B B*A.

Возведение матрицы в степень

Используем модуль linalg из библиотеки numpy, который позволяет осуществлять операции из линейной алгебры для матриц. Для этого импортируем его из библиотеки numpy, заменив, linalg на псевдоним ln.

9. Возвести $\underset{2\times2}{A}=\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}$ **в 3 степень**

Решение

```
A=np.array([[1,2],[3,4]])
from numpy import linalg as ln
```

```
print (ln.matrix_power(A,3))
[[ 37 54]
[ 81 118]]
```

Транспонирование матрицы

A.transpose() — один из операторов для получения транспонированной матрицы.

10. Транспонировать матрицы $A_{2\times2}=\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}$,

$B_{3\times4}=\begin{pmatrix}10 & 20 & 30 & 40\\ 50 & 60 & 70 & 80\\ 90 & 100 & 110 & 120\end{pmatrix}$

Решение

```
A=np.array([[1,2,3],[4,5,6]])
B=np.array([[10,20,30,40],
[50,60,70,80],[90,100,110,120]])

print (A.transpose())
print (matrix2.transpose())
[[1 3]
[2 4]]
[[ 10 50 90]
[ 20 60 100]
[ 30 70 110]
[ 40 80 120]]
```

11. Вычислить

$G = \left(A^T + 2B\right)\cdot(2C - 5D)$, $A=\begin{pmatrix}1 & 0 & -1\\ 2 & 3 & -2\end{pmatrix}$, $B=\begin{pmatrix}5 & 1\\ 3 & 2\\ 4 & -3\end{pmatrix}$,

$C=\begin{pmatrix}2 & 0 & -1 & 3\\ 5 & 1 & 2 & 5\end{pmatrix}$,

$D=\begin{pmatrix}7 & 1 & 1 & -3\\ 4 & 1 & -2 & 0\end{pmatrix}$.

Решение

```
A=np.array([[1,0,-1],[2,3,-2]])
B=np.array([[5,1],[3,2],[4,-3]])
C=np.array([[2,0,-1,3],[5,1,2,5]])
D=np.array([[7,1,1,-3],[4,1,-2,0]])
Print(np.dot(A.transpose()+B,2*C-5*D))
[[-216 -39 0 156]
[-143 -30 49 113]
[ -43 0 -91 13]]
```

12. Вычислить $G = (A^T - 2B) \cdot (2C - 5D)$

$A=\begin{pmatrix} 1 & 0 & -1 \\ 2 & 3 & -2 \end{pmatrix}$, $B=\begin{pmatrix} 5 & 1 \\ 3 & 2 \\ 4 & -3 \end{pmatrix}$, $C=\begin{pmatrix} 2 & 0 & -1 & 3 \\ 5 & 1 & 2 & 5 \end{pmatrix}$,

$D=\begin{pmatrix} 7 & 1 & 1 & -3 \\ 4 & 1 & -2 & 0 \end{pmatrix}$.

Решение

```
import numpy as np
A=np.array([[1,0,-1],[2,3,-2]])
print (A)
[[ 1 0 -1]
[ 2 3 -2]]
B=np.array([[5,1],[3,2],[4,-3]])
print (B)
[[ 5 1]
[ 3 2]
[ 4 -3]]
C=np.array([[2,0,-1,3],[5,1,2,5]])
print (C)
[[ 2 0 -1 3]
[ 5 1 2 5]]
D=np.array([[7,1,1,-3],[4,1,-2,0]])
print (D)
[[ 7 1 1 -3]
```

[4 1 -2 0]]

print (np.dot(A.transpose()-2*B,2*C-5*D))

[[279 45 63 -189]

[196 33 28 -136]

[239 33 119 -149]]

13. Вычислить $(A - 2B^T) \cdot (C + 3D)$

$A = \begin{pmatrix} -3 & 0 \\ 2 & -1 \\ 1 & 1 \\ 0 & 3 \end{pmatrix}$, $B = \begin{pmatrix} 3 & 2 & 1 & -4 \\ 1 & 1 & -3 & 0 \end{pmatrix}$, $C = \begin{pmatrix} 4 & 0 & -3 \\ 2 & 5 & -1 \end{pmatrix}$,

$D = \begin{pmatrix} \mathbf{10} & \mathbf{5} & \mathbf{-1} \\ \mathbf{-2} & \mathbf{3} & \mathbf{-3} \end{pmatrix}$.

Решение

import numpy as np

A=np.array([[-3,0],[2,-1],[1,1],[0,3]])

Print(A)

[[-3 0]

[2 -1]

[1 1]

[0 3]]

B=np.array([[3,2,1,-4],[1,1,-3,0]])

print (B)

[[3 2 1 -4]

[1 1 -3 0]]

C=np.array([[4,0,-3],[2,5,-1]])

print ©

[[4 0 -3]

[2 5 -1]]

D=np.array([[10,5,-1],[-2,3,-3]])

print (D)

[[10 5 -1]

```
[-2 3 -3]]
G=np.dot(A-2*B.transpose(),C+3*D)
print (G)
[[-298 -163 74]
[ -56 -72 42]
[ -62 83 -64]
[ 260 162 -78]]
```

14. Вычислить значение многочлена $F(x) = 2x^3 - x + 5$ **от матрицы** $A = \begin{pmatrix} -1 & 0 \\ 2 & 2 \end{pmatrix}$.

Решение

Здесь мы вводим функции, а не выполняем действия с матрицами для того, чтобы еще раз обратить внимание на удобство и универсальность подпрограмм в Python. В качестве параметра (аргумента) x можно поставить любую матрицу того же размера, и программа для решения задачи не изменится. Отметим, что eye(2) — это единичная матрица размера 2, а eye(n) — единичная матрица размера n.

```
def f(x):
return 2*ln.matrix_power(x,3)-x+5*
A=np.array([[-1,0],[2,2]])
Print( f(A))
[[ 4. 0.]
[ 10. 19.]]
```

15. Вычислите определители матриц

Для вычисления определителя матрицы A используем ln.det(A).

$$A = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 3 & 4 \\ 3 & 1 & 2 & 4 \\ 4 & 3 & 2 & 1 \end{vmatrix}$$

Решение

```
import numpy as np
from numpy import linalg as ln
A=np.array([[1,2,3,4],[2,1,3,4],[3,1,2,4],[4,3,2,1]])
Det_A=ln.det(A)
print (Det_A)
```

-30.0

И еще два примера.

16.

$$B=\begin{vmatrix} 4 & 6 & -2 & 4 \\ 1 & 2 & -3 & 1 \\ 4 & -2 & 1 & 0 \\ 6 & 4 & 4 & 6 \end{vmatrix}$$

Решение

```
B=np.array([[4,6,-2,4],[1,2,-3,1],[4,-2,1,0],[6,4,4,6]])
Det_B=ln.det(B)
print (Det_B)
```

-144.0

17.

$$C=\begin{vmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 3 & 4 \\ 3 & 1 & 2 & 4 \\ 4 & 3 & 2 & 1 \end{vmatrix}$$

Решение

```
C=np.array([[1,2,3,4],[2,1,3,4],[3,1,2,4],[4,3,2,1]])
Det_C=ln.det(C)
print (Det_C)
```

-30.0

18. Вычислить обратную матрицу

Для вычисления обратной матрицы A используем ln.inv(A)

$$A = \begin{pmatrix} 1 & -1 & 1 \\ 2 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix}.$$

Решение

```
A=np.array([[1,-1,1],[2,1,1],[1,1,2]])
A_inv=ln.inv(A)
Print(A_inv)
[[ 0.2 0.6 -0.4]
[-0.6 0.2 0.2]
[ 0.2 -0.4 0.6]]
```

19. Решить матричное уравнение $\mathbf{A \cdot X = B}$

A= $\begin{pmatrix} 3 & 1 \\ -3 & 1 \end{pmatrix}$, B= $\begin{pmatrix} 9 & 5 \\ -3 & -1 \end{pmatrix}$

Используем def matrix_eq(A,B):

Поясним, что матрицу X мы находим, программируя произведение (np.dot) обратной к матрице A (ln.inv(A)) и матрицы B, то есть $X = A^{-1} \cdot B$.

Решение

```
A=np.array([[3,1],[-3,1]])
B=np.array([[9,5],[-3,-1]])
def matrix_eq(A,B):
return np.dot(ln.inv(A),B)
print (matrix_eq(A,B))
[[ 2. 1.]
[ 3. 2.]]
```

20. Найдите ранг матрицы

$$\begin{pmatrix} 0 & -1 & 3 & 0 & 2 \\ 2 & -4 & 1 & 5 & 3 \\ -4 & 5 & 7 & -10 & 0 \\ -2 & 1 & 8 & -5 & 3 \end{pmatrix}$$

Ранг матрицы находит функция ln.matrix_rank(A).

Решение

```
A=[[0,-1,3,0,2],[2,-4,1,5,3],[-4,5,7,-10,0],[-2,1,8,-5,3]]
A=np.array([[0,-1,3,0,2],[2,-4,1,5,3],
[-4,5,7,-10,0],[-2,1,8,-5,3]])
print (ln.matrix_rank(A))
```

2

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Вычислить *G*, если даны *A*, *B*, *C*, *D*.

21. $G = (A - 2B^T) \cdot (C + 3D)$, $A = \begin{pmatrix} 2 & -1 \\ 3 & 2 \\ 5 & -2 \end{pmatrix}$, $B = \begin{pmatrix} 3 & 2 & 1 & -4 \\ 1 & 1 & -3 & 0 \end{pmatrix}$, $C = \begin{pmatrix} 4 & 0 & -3 \\ 2 & 5 & -1 \end{pmatrix}$,

$D = \begin{pmatrix} 10 & 5 & -1 \\ -2 & 3 & -3 \end{pmatrix}$.

22. $G = (2A + B)^T \cdot (3C - 4D)$, $A = \begin{pmatrix} 2 & -1 \\ 3 & 2 \\ 5 & -2 \end{pmatrix}$, $B = \begin{pmatrix} -2 & -1 \\ 11 & -2 \\ 5 & 7 \end{pmatrix}$, $C = \begin{pmatrix} 1 & 3 & 0 & 4 \\ 0 & 2 & -3 & 5 \\ 5 & 3 & 10 & -2 \end{pmatrix}$,

$D = \begin{pmatrix} -2 & 5 & 1 & -3 \\ 1 & 1 & -5 & -4 \\ 2 & 6 & 11 & -1 \end{pmatrix}$.

23. $G=(3A-B)\cdot(2C+3D)^T$, $A=\begin{pmatrix}0&1&1&-1\\1&1&-2&-5\\2&2&-1&-1\end{pmatrix}$, $B=\begin{pmatrix}-1&2&1&-1\\2&4&-3&-2\\1&1&10&-1\end{pmatrix}$, $C=$

$\begin{pmatrix}4&2&1&-5\\3&1&-2&-1\\5&4&0&-4\end{pmatrix}$, $D=\begin{pmatrix}-2&1&3&-3\\1&1&-1&-2\\1&2&-3&-1\end{pmatrix}$.

24. $G=(2A+3B)\cdot\left(C-2D^T\right)$,

$A=\begin{pmatrix}2&1&-1\\0&3&2\\1&4&-3\\0&5&2\end{pmatrix}$, $B=\begin{pmatrix}0&1&-2\\1&1&1\\3&-2&2\\2&0&4\end{pmatrix}$, $C=\begin{pmatrix}3&-2\\1&4\\2&-3\end{pmatrix}$, $D=\begin{pmatrix}-3&2&-2\\-4&0&-1\end{pmatrix}$.

25. $G=2\cdot(A-B)\cdot(3C+2D)^T$,

$A=\begin{pmatrix}-2&1\\0&-3\\-1&4\\2&-1\end{pmatrix}$, $B=\begin{pmatrix}3&-2\\4&-1\\1&4\\5&-3\end{pmatrix}$, $C=\begin{pmatrix}0&-1\\3&-4\\5&-1\end{pmatrix}$, $D=\begin{pmatrix}-3&4\\1&2\\-2&3\end{pmatrix}$.

26. $G=3(A+B)^T\cdot(2C-D)$, $A=\begin{pmatrix}-2&3&-4\\-3&5&-2\end{pmatrix}$, $B=\begin{pmatrix}0&-3&-1\\2&0&-4\end{pmatrix}$,

$C=\begin{pmatrix}0&4&1&-3\\2&1&-1&5\end{pmatrix}$, $D=\begin{pmatrix}3&0&2&-2\\0&3&-3&0\end{pmatrix}$.

27. $G=5\left(A^T-2B\right)\cdot(C+2D)$, $A=\begin{pmatrix}0&1&-2\\3&2&1\\2&0&-2\\1&4&-1\end{pmatrix}$, $B=\begin{pmatrix}2&1&2&-2\\0&3&-1&-1\\4&1&0&-3\end{pmatrix}$, $C=\begin{pmatrix}-3&4\\-2&1\\-1&4\\-2&3\end{pmatrix}$, D

$=\begin{pmatrix}2&-2\\3&-4\\4&-3\\5&-4\end{pmatrix}$

Вычислить значение многочлена $f(x)$ от матрицы A.

28. $f(x)=2x^3+x+5$, $A=\begin{pmatrix}-1 & 0\\ 2 & 2\end{pmatrix}$.

29. $f(x)=x^3+2x-4$, $A=\begin{pmatrix}3 & 1\\ 0 & 1\end{pmatrix}$.

30. $f(x)=x^3+x^2+x+1$, $A=\begin{pmatrix}2 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 3\end{pmatrix}$.

31. $f(x)=2x^2+x-3$, $A=\begin{pmatrix}-1 & 0 & 0\\ -2 & 1 & 0\\ -3 & 0 & 1\end{pmatrix}$.

32. $f(x)=x^3-x^2-1$, $A=\begin{pmatrix}-2 & 1\\ 1 & 0\end{pmatrix}$.

33. $f(x)=x^2-x^3+1-x$, $A=\begin{pmatrix}0 & 0 & -1\\ 0 & 1 & 0\\ 1 & 0 & 0\end{pmatrix}$.

34. $f(x)=5x^2-x-5$, $A=\begin{pmatrix}1 & 0 & 0\\ 2 & 1 & 0\\ 3 & 2 & 1\end{pmatrix}$.

35. $f(x)=3x^4-x^2+3$, $A=\begin{pmatrix}-5 & -1 & 4 & 2 & -9 & 11\\ -1 & 5 & 4 & -6 & 8 & -6\\ 4 & 4 & -4 & 2 & -7 & -3\\ 2 & -6 & 2 & -8 & 5 & -2\\ -9 & 8 & -7 & 5 & -6 & 7\\ 11 & -6 & -3 & -2 & 7 & -8\end{pmatrix}$

Вычислить определитель.

36. $\begin{vmatrix} 3 & 5 & 7 & 2 \\ 7 & 6 & 3 & 7 \\ 5 & 4 & 3 & 5 \\ -5 & -6 & -5 & -4 \end{vmatrix}$.

37. $\begin{vmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix}$.

38. $\begin{vmatrix} 2 & 4 & 8 & 0 \\ 4 & 8 & 0 & 27 \\ 8 & 0 & 27 & 9 \\ 0 & 27 & 9 & 3 \end{vmatrix}$.

39. $\begin{vmatrix} -3 & 2 & 4 & 0 \\ -2 & 3 & 6 & 9 \\ -4 & 0 & 3 & 9 \\ -6 & 2 & 3 & 0 \end{vmatrix}$.

40. $\begin{vmatrix} 2 & -3 & 4 & 3 \\ 4 & -2 & 3 & 2 \\ 5 & -5 & 2 & 3 \\ 3 & -4 & 4 & 3 \end{vmatrix}$.

41. $\begin{vmatrix} 1 & 1 & 1 & 1 \\ 4 & 3 & 2 & 1 \\ 16 & 9 & 4 & 1 \\ 64 & 27 & 8 & 1 \end{vmatrix}$.

42. $\begin{vmatrix} 0 & 3 & 5 & 7 \\ -3 & 0 & 3 & 5 \\ -5 & -3 & 0 & 3 \\ -7 & -5 & -3 & 0 \end{vmatrix}$

Найти матрицу обратную к данной.

44. $A = \begin{pmatrix} -1 & -4 & 8 & 15 & -1 & 5 \\ -8 & 4 & -5 & 7 & -5 & 8 \\ 7 & 3 & -14 & 7 & 8 & -2 \\ 10 & -9 & 0 & -4 & -2 & -6 \\ -6 & -3 & -4 & -2 & 12 & 2 \\ -7 & 8 & 12 & -4 & -9 & 9 \end{pmatrix}$

45. $B = \begin{pmatrix} 3 & -5 & 9 & 3 & -1 & 0 \\ 5 & -3 & -5 & 0 & -5 & 13 \\ -6 & 10 & -4 & 15 & -2 & 0 \\ 15 & -6 & 9 & -9 & -2 & -6 \\ 8 & -3 & 5 & -5 & 8 & -1 \\ 0 & 8 & 6 & -4 & 12 & -7 \end{pmatrix}$

46. $A = \begin{pmatrix} -5 & -1 & 4 & 2 & -9 & 11 \\ -1 & 5 & 4 & -6 & 8 & -6 \\ 4 & 4 & -4 & 2 & -7 & -3 \\ 2 & -6 & 2 & -8 & 5 & -2 \\ -9 & 8 & -7 & 5 & -6 & 7 \\ 11 & -6 & -3 & -2 & 7 & -8 \end{pmatrix}$

47. $A = \begin{pmatrix} -1 & 0 & 8 & 6 & -1 & 5 \\ 8 & 4 & -5 & 4 & -5 & 11 \\ 3 & 6 & -14 & 7 & 8 & -2 \\ 10 & 4 & 0 & -4 & -2 & -6 \\ -6 & -3 & -4 & -2 & 2 & 13 \\ -7 & 8 & 12 & -4 & 9 & -5 \end{pmatrix}$

48. $B = \begin{pmatrix} 8 & -5 & 12 & -5 & -1 & 0 \\ 5 & -3 & 8 & 13 & -5 & 13 \\ -6 & 13 & -4 & 15 & -3 & 3 \\ 15 & -11 & 9 & -9 & -2 & -6 \\ 8 & -3 & 9 & -5 & 8 & -1 \\ 0 & 8 & 6 & -4 & 12 & 7 \end{pmatrix}$

$$49.\ A = \begin{pmatrix} 6 & 4 & 1 \\ 5 & 6 & 2 \\ 1 & 1 & -1 \end{pmatrix}.$$

$$50.\ A = \begin{pmatrix} -6 & -1 & 4 \\ -4 & -4 & 3 \\ 3 & -2 & -2 \end{pmatrix}.$$

51. Решить матричные уравнения и проверить результат, выполнив умножение матриц исходной задачи: $AX = B$, где

$$A = \begin{pmatrix} -1 & -4 & 8 & 15 & -1 & 5 \\ -8 & 4 & -5 & 7 & -5 & 8 \\ 7 & 3 & -14 & 7 & 8 & -2 \\ 10 & -9 & 0 & -4 & -2 & -6 \\ -6 & -3 & -4 & -2 & 12 & 2 \\ -7 & 8 & 12 & -4 & -9 & 9 \end{pmatrix} \text{ и } B = \begin{pmatrix} 3 & -5 & 9 & 3 & -1 & 0 \\ 5 & -3 & -5 & 0 & -5 & 13 \\ -6 & 10 & -4 & 15 & -2 & 0 \\ 15 & -6 & 9 & -9 & -2 & -6 \\ 8 & -3 & 5 & -5 & 8 & -1 \\ 0 & 8 & 6 & -4 & 12 & -7 \end{pmatrix}$$

Найти ранг матрицы:

$$52.\ B = \begin{pmatrix} 8 & -5 & 12 & -5 & -1 & 0 \\ 5 & -3 & 8 & 13 & -5 & 13 \\ -6 & 13 & -4 & 15 & -3 & 3 \\ 15 & -11 & 9 & -9 & -2 & -6 \\ 8 & -3 & 9 & -5 & 8 & -1 \\ 0 & 8 & 6 & -4 & 12 & 7 \end{pmatrix}$$

$$53.\ \begin{pmatrix} 1 & 3 & 7 & 2 & 5 \\ -1 & 0 & 4 & 8 & 3 \\ 3 & 6 & 10 & -4 & 7 \end{pmatrix}.$$

54. $\begin{pmatrix} 0 & 5 & -1 & 1 & 5 \\ 2 & 3 & 0 & 1 & 6 \\ -1 & -3 & 1 & 3 & 0 \\ 3 & -1 & 0 & 4 & 6 \end{pmatrix}$.

55.

$$\begin{pmatrix} 0 & 1 & -1 & 3 \\ 2 & 5 & 1 & 11 \\ 1 & 2 & 1 & 4 \end{pmatrix}.$$

56.

$$\begin{pmatrix} 0 & 6 & 3 & 5 & 1 \\ -3 & 2 & 4 & 1 & 0 \\ 5 & 1 & 4 & 3 & 2 \\ -3 & 8 & 7 & 6 & 1 \\ 1 & 0 & 3 & 4 & 0 \end{pmatrix}.$$

Определить максимальное число линейно независимых строк матрицы:

57.

$$\begin{pmatrix} 1 & 2 & 1 & 4 \\ 0 & 5 & -1 & 4 \\ -1 & 3 & 4 & 6 \end{pmatrix}.$$

58.

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & -1 & 1 \\ 1 & -1 & 2 \\ 3 & -6 & 5 \end{pmatrix}.$$

59.

$$\begin{pmatrix} 2 & 3 & 5 \\ 3 & 7 & 8 \\ 1 & -6 & 1 \\ 7 & -2 & 15 \end{pmatrix}.$$

60.

$$\begin{pmatrix} 1 & 1 & 1 & 6 \\ 2 & -1 & 1 & 3 \\ 1 & -1 & 2 & 5 \\ 3 & -6 & 5 & 6 \end{pmatrix}.$$

61.

$$A = \begin{pmatrix} -7 & -1 & -5 & 2 & 6 & -2 \\ -1 & -8 & 8 & 4 & -1 & -6 \\ -5 & 8 & 1 & 2 & 7 & -3 \\ 2 & 4 & 2 & 6 & 0 & -1 \\ 6 & -1 & 7 & 0 & -9 & 7 \\ -2 & -6 & -3 & -1 & 7 & 0 \end{pmatrix}.$$

СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ (СЛУ)

Напомним, что система из m линейных уравнений с n переменными имеет вид:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \end{cases},$$

где a_{ij} и b_j (для всех значений $i=1,2,...,m;\ j=1,2,...,n$) — это произвольные числа, называемые соответственно коэффициентами при переменных $x_1, x_2, ..., x_n$ и свободными членами. Матричная форма записи СЛУ имеет вид $A \cdot X = B$. Где

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$ — матрица из коэффициентов

при неизвестных, $X = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$ — матрица из неизвестных системы и $B = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{pmatrix}$ — матрица из свободных членов. Решением системы m линейных уравнений с n переменными называют упорядоченный набор чисел , при подстановке которых в каждое уравнение системы вместо соответствующих переменных получают верное равенство. Вообще, как доказывает Теорема Кронеккера-Капели, СЛУ могут иметь 1 решение, бесконечное множество решений и не иметь решения.

Сначала рассмотрим случай, когда система имеет единственное решение. Это значит, что матрица системы невырожденная, то есть ее определитель не равен нулю, и эту систему можно решать по правилу Крамера или методом обратной матрицы.

62. Решить СЛУ (случай единственного решения)

$$\begin{cases} 3x_1 - x_2 + x_3 = 5 \\ x_1 - x_2 - x_3 = 2 \\ 5x_1 - 3x_2 - x_3 = 10 \end{cases};$$

Решение

Для решения системы воспользуемся функцией **numpy.linalg.solve** модуля numpy (документация — http://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.solve.html). Функция принимает на вход 2 параметра:

1-й — матрица коэффициентов перед переменными

2-й — вектор свободных членов

```
A=np.array([[3,-1,1],[1,-1,-1],[5,-3,-1]])
B=np.array([3,11,8])
X=np.linalg.solve(A,B)
for t, x in zip(X, [‘x1=’,’x2=’, ‘x3=’]):
print (x,t)
x1= -1.2857142857142854
x2= -2.142857142857143
x3= 6.7142857142857135
```

Функция zip используется для вывода решение системы на экран в виде x1=..., x2=..., x3=...

Обратим внимание, что определитель основной матрицы системы не равен нулю, мы получили единственное решение, хотя и не целочисленное.

```
import numpy as np
from numpy import linalg as ln
A=np.array([[3,-1,1],[2,1,1],[1,1,2]])
Det_A=ln.det(A)
print (Det_A)
7.000000000000001
```

63. Решить СЛУ

$$\begin{cases} x_1 - x_2 + x_3 = 3, \\ 2x_1 + x_2 + x_3 = 11, \\ x_1 + x_2 + 2x_3 = 8. \end{cases}$$

Решение

```
A=np.array([[1,-1,1],[2,1,1],[1,1,2]])
B=np.array([3,11,8])
X=np.linalg.solve(A,B)
for t, x in zip(X, [‘x1=’,’x2=’, ‘x3=’]):
print (x,t)
x1= 4.0
x2= 2.0
x3= 1.0
```

Решение системы линейных уравнений AX=B методом обратной матрицы

Детальное изложение метода можно найти в учебнике [1] часть 1.3.3 Реализуем следующий алгоритм:

1. Вычислим определитель матрицы A. Если определитель равен нулю, то конец решения. Система имеет бесконечное множество решений.

2. При определителе отличном от нуля, через алгебраические дополнения находится обратная матрица A.

3. Вектор решения X={ $x_1, x_2, \ldots, x_n$ } получается умножением обратной матрицы на вектор результата B.

64. Решить СЛУ методом обратной матриц

Найдем обратную матрицу с помощью функции np.linalg.inv(a) и умножим ее на столбец свободных членов с помощью функции np.dot(a_inv, b)

$$\begin{cases} 2x_1 + x_2 - 2x_3 = -3 \\ x_1 - 2x_2 + x_3 = 5 \\ 3x_1 + x_2 - x_3 = 0 \end{cases}$$

```
import numpy as np
a = np.array([[2, 1, -2], [1, -2, 1], [3, 1, -1]])
print (“Матрица:\n”, a)
Матрица:
[[ 2 1 -2]
[ 1 -2 1]
[ 3 1 -1]]
b = np.array([-3, 5, 0])
det = np.linalg.det(a)
if det==0:
print («Определитель матрицы равен нулю, система имеет бесконечно много решений»)
else:
a_inv = np.linalg.inv(a)
X=np.dot(a_inv, b)
print («Решение системы:\n», X)
Решение системы:
[1. -1. 2.]
```

Решение системы линейных уравнений AX=B с использованием правила Крамера.

Если определитель матрицы системы не равен нулю, ее можно решить по правилу Крамера. Для этого находят определитель матрицы системы и делят на него определители, полученные заменой i-го столбца матрицы системы на столбец свободных членов. Эти отношения и будут составлять решение системы. Отметим, что оно будет единственным.

Детальное изложение метода можно найти в учебнике [1] пункт 1.3.4.

65. Решить СЛУ по правилу Крамера

$$\begin{cases} 2x_1 + x_2 - 2x_3 = -3 \\ x_1 - 2x_2 + x_3 = 5 \\ 3x_1 + x_2 - x_3 = 0 \end{cases}$$

Отметим, что ниже инструкция a_1=np.column_ stack((b,s_1,s_2) записывает вместо 1-го столбца матрицы a столбец свободных членов. Ниже, так же используется обращение к элементам массива, s_i=a[:,i] — срез, который записывает i-тый столбец матрицы a, причем первый нумеруется как нулевой и т.д.

```
det = np.linalg.det(a)
if det==0:
print( «Определитель матрицы равен нулю, система имеет бесконечно много решений»)
else:
s_0=a[:,0]
s_1=a[:,1]
s_2=a[:,2]
a_1=np.column_stack((b,s_1,s_2))
det_1 = np.linalg.det(a_1)
a_2=np.column_stack((s_0,b,s_2))
det_2 = np.linalg.det(a_2)
a_3=np.column_stack((s_0,s_1,b))
det_3 = np.linalg.det(a_3)
x_1=det_1/det
x_2=det_2/det
x_3=det_3/det
print( «Решение системы:», x_1,x_2,x_3)
```

Решение системы: 0.9999999999999996 -1.0 2.0000000000000004

Теперь рассмотрим пример системы, имеющей бесконечное множество решений.

Этапы решения СЛУ¶

1. Найти определитель матрицы системы det A.

2. Если он не равен нулю, вывести решение с помощью np.linalg.solve(A,B).

3. Иначе, сравниваем ранги расширенной и основной матриц системы и используем теорему Кронеккера- Капели, [1] пункт 1.3.4.

Ниже приведен возможный вариант решения задачи о нахождении решения системы линейных уравнений в общем виде. Отметим, что команда B.reshape(1,B.shape[0]) изменяет форму вектора свободных членов, который изначально представлен как массив-строка, в массив-столбец. Команда np.vstack((A,B1)) объединяет матрицу коэффициентов и столбец свободных членов в один двумерный массив для дальнейшей работы.

67. Решить СЛУ

$$\begin{cases} x_1 + 2x_2 - x_3 = 7 \\ 2x_1 - 3x_2 + x_3 = 3 \\ x_1 + x_2 - x_3 = 16 \end{cases}$$

Решение.

```
import numpy as np
A=np.array([[1,2,-1],[2,-3,1],[4,1,-1]])
B=np.array([7,3,16])
def solve_slu(A,B):
Det_A=np.linalg.det(A)
B1=B.reshape(1,B.shape[0])
AB=np.vstack((A,B1))
if Det_A!=0:
X=np.linalg.solve(A,B)
```

```
print ('EDINSVENNOE RESHENIE')
print( X)
if Det_A==0:
rank_A=np.linalg.matrix_rank(A)
rank_AB=np.linalg.matrix_rank(AB)
if rank_A==rank_AB:
print ('Сиситема имеет бесконечное множество решений, частное решение CHASTNOE'+X)
else: print ('Система не имеет решений')
return
solve_slu(A,B)
print (np.linalg.matrix_rank(A))
B1=B.reshape(1,B.shape[0])
AB=np.vstack((A,B1))
print (np.linalg.matrix_rank(AB))
```

Система не имеет решений

2

3

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Решить СЛУ

68. $\begin{cases} -6x_1 + 2x_2 + x_3 - 5x_4 + 2x_5 - 3x_6 - x_7 + 7x_8 = 49, \\ -4x_1 - 6x_3 - 5x_4 + x_5 + 4x_6 - 6x_7 - 2x_8 = 23, \\ 5x_1 - 5x_2 - x_3 - 3x_4 + 5x_5 + 2x_6 + 6x_7 + 3x_8 = -62, \\ -5x_1 + 3x_2 - 4x_3 + 5x_4 - 3x_5 - 8x_7 + x_8 = 74, \\ 9x_1 - 4x_3 - 3x_4 - 3x_5 + 4x_6 - 5x_7 - 3x_8 = -24, \\ -5x_1 - 5x_2 - 3x_3 + 7x_4 + 2x_5 - 5x_6 - 2x_7 - x_8 = 34, \\ -x_1 + x_2 + 5x_3 + 2x_5 - 10x_6 - x_7 - x_8 = 39, \\ 7x_1 - x_2 + 4x_3 + 3x_4 - 4x_5 + 8x_8 = -14 \end{cases}$

69. $\begin{cases} -5x_1 - 5x_2 + 11x_4 + 2x_5 - 9x_6 + 3x_7 - x_8 = 34, \\ -11x_1 - 4x_2 + 4x_3 - 5x_4 - 8x_5 + 4x_6 + 2x_7 + 8x_8 = 62, \\ -x_1 + 2x_3 - 7x_4 + 2x_5 - 4x_6 - 11x_8 = 53, \\ x_1 - 7x_2 + 13x_3 - 4x_4 + 2x_5 - x_6 - 4x_7 - 8x_8 = 93, \\ 2x_2 + 7x_3 - 4x_4 - 5x_6 - 6x_7 + 6x_8 = 32, \\ -3x_1 - x_2 + 5x_3 - x_6 + x_7 + 6x_8 = 44, \\ 7x_1 + 8x_3 - 4x_4 + x_5 - 5x_6 + 7x_7 - 3x_8 = 99, \\ -4x_1 - 3x_2 + 2x_4 + 9x_5 - 8x_6 + 3x_7 - 2x_8 = -75 \end{cases}$

70

$$\begin{cases} 2x_1 + 3x_2 - x_3 + x_4 = 5 \\ 3x_1 - x_2 + 2x_3 + x_4 = 1 \\ x_1 + 2x_2 + 3x_3 + 4x_4 = 6 \\ 6x_1 + 4x_2 + 4x_3 + 6x_4 = 1 \end{cases}$$

71. $\begin{cases} 2x_1 + x_2 + x_3 + x_4 = 1 \\ x_2 - x_3 + 2x_4 = 2 \\ 2x_1 + 2x_2 + 3x_4 = 3 \end{cases}$

72 $\begin{cases} x_1 + 2x_2 - x_3 = 7 \\ 2x_1 - 3x_2 + x_3 = 3 \\ 4x_1 + x_2 - x_3 = 16 \end{cases}$

73. $\begin{cases} x_1 + 2x_2 + 3x_3 - 2x_4 = 6 \\ 2x_1 + 4x_2 - 2x_3 - 3x_4 = 18 \\ 3x_1 + 2x_2 - x_3 + 2x_4 = 4 \\ 2x_1 - 3x_2 + 2x_3 + x_4 = -8 \end{cases}$

74. $\begin{cases} -6x_1 + 9x_2 + 3x_3 + 2x_4 = 4 \\ -2x_1 + 3x_2 + 5x_3 + 4x_4 = 2 \\ -4x_1 + 6x_2 + 4x_3 + 3x_4 = 3 \end{cases}$

75. $\begin{cases} 5x_1 - x_2 + 2x_3 + x_4 = 7 \\ 2x_1 + x_2 + 4x_3 - 2x_4 = 1 \\ x_1 - 3x_2 - 6x_3 + 5x_4 = 0 \end{cases}$

76. $\begin{cases} 3x_1 + 2x_2 - x_3 + x_4 = -4 \\ 2x_1 - x_2 + 3x_3 - x_4 = 7 \\ -x_1 + 3x_2 + 3x_3 + x_4 = 14 \\ 5x_1 - 3x_2 - x_3 - 2x_4 = -10 \end{cases}$

77. $\begin{cases} x + 2y + z = 8 \\ -2x + 3y - 3z = -5 \\ 3x - 4y + 5z = 10 \end{cases}$

78. $\begin{cases} 3x + 2y + z = -8 \\ 2x + 3y + z = -3 \\ 2x + y + 3z = -1 \end{cases}$

79. $\begin{cases} -6x - y + 4z = 1 \\ -4x - 4y + 3z = -3 \\ 3x - 2y - 2z = -3 \end{cases}$

80. $\begin{cases} 3x + 2y + z = 1 \\ 6x + 5y + 4z = -2 \\ 9x + 8y + 7z = 3 \end{cases}$

81. $\begin{cases} 2x - 3y - z = -6 \\ 3x + 4y + 3z = -5 \\ x + y + z + 2 = 0 \end{cases}$

82. $\begin{cases} 6x + 4y + z = -17 \\ 5x + 6y + 2z = -10 \\ x + y - z + 1 = 0 \end{cases}$

83. $\begin{cases} -2x + y + 6 = 0 \\ x - 2y - z = 5 \\ 3x + 4y - 2z = 13 \end{cases}$

ТЕМА 2. ЭЛЕМЕНТЫ АНАЛИТИЧЕСКОЙ ГЕОМЕТРИИ

ЛИНЕЙНЫЕ ПРОСТРАНСТВА. ВЕКТОРЫ НА ПЛОСКОСТИ И В ПРОСТРАНСТВЕ. ОПЕРАЦИИ НАД ВЕКТОРАМИ

¶

Вектором на плоскости и в пространстве называют множество сонаправленных отрезков, имеющих одинаковую длину. Обозначают векторы малыми латинскими буквами со стрелкой $\vec{a}$. Если даны координаты начала и конца вектора, то что бы найти координаты вектора нужно из координат конца вычесть координаты начала. Если $A(x_1; y_1)$, $B(x_2; y_2)$, то $\overrightarrow{AB}=(x_2-x_1; y_2-y_1)$, а длина вектора $\left|\overrightarrow{AB}\right|=|AB|=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$. Операции с векторами: сложение, вычитание, умножение на число, векторное, скалярное, смешанное произведение, рассмотрены ниже. Векторы линейно зависимые, если один из них можно представить, как линейную комбинацию других. Далее нам понадобятся знания понятий ортонормированного базиса и разложения вектора по базисным, которые можно посмотреть в [1].

84. Даны вектор *a*(5;1) и точка *A*(-2;3). Найти координаты точки *B*, такой, что $a=\overrightarrow{AB}$.

Решение

```
a=np.array([5,1])
A=np.array([-2,3])
B=a+A
print (B)
[3 4]
```

85. Найти вектор результат векторного произведения векторов $a=(1;3;-1)$ и $b=(2;-2;1)$ и его длину.

Векторное произведение двух векторов $\vec{a}=(a_1,a_2,a_3)$, $\vec{b}=(b_1,b_2,b_3)$ – это вектор $\vec{c}=\left[\vec{a}\times\vec{b}\right]=\left(\begin{vmatrix}a_2 & a_3\\ b_2 & b_3\end{vmatrix},\begin{vmatrix}a_3 & a_1\\ b_3 & b_1\end{vmatrix},\begin{vmatrix}a_1 & a_2\\ b_1 & b_2\end{vmatrix}\right)$, длина вектора равна корню квадратному из суммы квадратов координат этого вектора.

Введем координаты векторов в виде строчек матрицы *a*, *s_i=a[:,i]* –столбцы матрицы *a*: первый, второй и третий, найдем определители, это и есть координаты векторного произведения. Запишем вектор $\vec{c}$ через разложение по базисным ортам *i, j, k*. Найдем его длину через функцию norm(c).

Решение

```
a = np.array([[1, 3, -1],[2,-2,1]])
s_0=a[:,0]
s_1=a[:,1]
s_2=a[:,2]
a_1=np.column_stack((s_1,s_2))
det_1 = np.linalg.det(a_1)
a_2=np.column_stack((s_0,s_2))
det_2 = -np.linalg.det(a_2)
a_3=np.column_stack((s_0,s_1))
det_3 = np.linalg.det(a_3)
c=np.array([det_1,det_2,det_3])
print( “c=(“,det_1,”)i+(“,det_2,”) j+(“,det_3, “)k”
n=np.linalg.norm(c))
print( «Длина векторного произведения =», n
```

c=(1.0)i+(-3.0) j+(-8.0)k)

Длина векторного произведения = 8.60232526704

86. Определить, будут ли линейно зависимы системы векторов:

а) (1; 1; 4; 2), (−3; −1; 3; 4), (1; −1; −2; 4);

б) (1, −1, −2, 4), (0, 2, 6, −2), (−3, −1, 3, 4), (−1, 0, — 4, −7).

Решение.

a.T обозначает операцию транспонирования, что равносильно использованной выше функции transpose. Для примера а) рассмотрен метод без использования вспомогательной функции, для примера б) введена функция LinInd(a). Инструкция a.shape[1] определяет количество строк в матрице a, a.shape[0]- количество столбцов.

```
а).
np.array([[1,1,4,2],[-3,-1,3,4],[1,-1,-2,4]])
a=a.T
rang=np.linalg.matrix_rank(a)
k=a.shape[1]
if rang==a.shape[1]:
print( «Линейно независимы»)
else:
print («Линейно зависимы»)
Линейно независимы


б).
def LinInd(a):
a=a.T
rang=np.linalg.matrix_rank(a)
k=a.shape[1]
if rang==a.shape[1]:
print («Линейно независимы»)
else:
print(«Линейно зависимы»)
```

a=np.array([[1,-1,-2,4],
[0,2,6,-2],[-3,-1,3,4],[-1,0,-4,-7]])

LinInd(a)

Линейно зависимы

87. Определить является ли система векторов (−1, 0, — 4, −7), (1; 1; 4; 2), (−3; −1; 3; 4), (1, −1, −2, 4), (0, 2, 6, −2) линейно независимой (способ 2).

Решение. Определим, является ли система векторов a1, a2, a3, a4, a5 линейно зависимой или нет. Для этого составим линейную комбинацию: α1a1 + α2a2 + α3a3 + α4a4 + α5a5. Подставим числовые значения векторов a1, a2, a3, a4, a5 и приравняем линейную комбинацию к нулевому вектору (0,0,0,0). В результате мы получим однородную систему линейных уравнений. Если эта СЛОУ имеет одно решение, то векторы a1, a2, a3, a4, a5 линейно независимы. Если решений бесконечно много, то векторы линейно зависимы. СЛОУ можно решить многими способами. В настоящем примере реализован метод Гаусса. С синтаксисом циклических конструкций на Python мы знакомились выше. Детали реализации данного алгоритма предлагаем разобрать по ссылке https://doc.lagout.org/programmation/python/Numerical%20Methods%20in%20Engineering%20with%20Python%203%20%5BKiusalaas%202011-02-21%5D.pdf .

In [20]:
```
import numpy as np
def gaussElimin(a,b):
    n = len(b)
# Elimination Phase
    for k in range(0,n-1):
        for i in range(k+1,n):
            if a[i,k] != 0.0:
                lam = a [i,k]/a[k,k]
                a[i,k+1:n] = a[i,k+1:n] - lam*a[k,k+1:n]
                b[i] = b[i] - lam*b[k]
# Back substitution
    for k in range(n-1,-1,-1):
        b[k] = (b[k] - np.dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
    return b
a = np.array([[-1, 0, -4,-7], [1, 1, 4,2], [-3, -1, 3,4],[1,-2,-2,4],[0,2,6,-2]])
b = np.array([0, 0, 0,0])
gaussElimin(a,b)
```

В результате выполнения программы получаем нулевой вектор, значит система линейно зависима.

88. Доказать, что векторы (3; 4; 3), (−2; 3; 1), (4; −2; 3) образуют базис и найти координаты вектора (−17; 18; −7) в этом базисе.

Стандартную схему доказательства линейной независимости векторов мы разобрали. Здесь разберём упрощённый вариант проверки, касающийся квадратных матриц, для которых числовой характеристикой является не только ранг, но и определитель. Вспомним о том, что понижение ранга квадратной матрицы приводит к её вырожденности (т.е. определитель такой матрицы будет равен нулю). Поэтому для ответа на вопрос о линейной зависимости трёх векторов в трёхмерном пространстве можно просто подсчитать значение определителя, составленного из координат этих векторов. Если он не ноль, то вектора линейно не зависимые:

```
b = np.array([-17, 18, -7])
a = np.array([[3, 4, 3], [-2,3,1], [4,-2,3]])
a=a.T
det = np.linalg.det(a)
det
49.000000000000014
```

Следовательно, матрица является невырожденной и её ранг равен трём. Поэтому векторы линейно независимы, и образуют базис трёхмерного пространства. Полученная система линейных уравнений является неоднородной. Основная матрица этой системы имеет ранг, равный трём, что совпадает с количеством переменных в системе. Поэтому система является определённой, то есть имеет единственное решение, которое можно находить разными способами. Воспользуемся встроенной функцией linalg.solve библиотеки Numpy.

```
x = np.linalg.solve(a, b)
x
array([ 1., 2., -4.])
```

Ответ: вектор в базисе, состоящем из векторов (3; 4; 3), (−2; 3; 1), (4; −2; 3), имеет координаты (1; 2; −4).

89. Построить ортогональный базис, построенный на векторах $a_1=(3;4;3), a_2=(-2;3;1), a_3=(4;-2;3)$.

Замечание. Данный в задаче 88 базис из векторов $a_1=(3;4;3), a_2=(-2;3;1), a_3=(4;-2;3)$ не является ортогональным, так как его векторы не являются попарно ортогональными:

$(\boldsymbol{a}_1,\boldsymbol{a}_2)=3\cdot(-2)+4\cdot 3+3\cdot 1=9\neq 0, (\boldsymbol{a}_1,\boldsymbol{a}_3)\neq 0, (\boldsymbol{a}_2,\boldsymbol{a}_3)\neq 0$.

От системы a_i перейдем к системе векторов $\boldsymbol{a}_i^*$, в которой все векторы попарно ортогональны при всех i от 1 до 3 следующим образом. Возьмем $\boldsymbol{a}_1^*=\boldsymbol{a}_1$, $a_2^*=a_2+a_1^*\cdot\mu$, найдем μ, умножив скалярно обе части последнего на a_1, получим $\mu=-\frac{(a_1,a_2)}{(a_1,a_1)}$, рассуждая аналогично построим $a_3^*=a_3+a_1^*\cdot\mu+a_2^*\cdot\beta$, где $\mu=-\frac{(a_3,a_1)}{(a_1,a_1)}, \beta=-\frac{(a_3,a_2^*)}{(a_2^*,a_2^*)}$.

Процесс построения ортогонального базиса называют ортогонализацией.

Отметим, что система векторов $\boldsymbol{a}_i^*$ в коде обозначена как система b_i, mu — это мю μ. Можно проверить равенство нулю скалярных произведений.

Решение

```
import numpy as np
def mu(x,y):
return -np.dot(x,y)/np.dot(x,x)
a1 = np.array([3, 4, 3])
a2 = np.array([-2, 3, 1])
a3 = np.array([4, -2, 3])
b1=a1
print('Ортогональный базис›)
print(b1)
b2=a2+a1*mu(a1,a2)
print(b2)
b3=a3+a1*mu(a1,a3)+b2*mu(b2,a3)
print(b3)
```

```
print('Скалярные произведения')
k=np.dot(b1,b2)
print(k)
i=np.dot(b3,b2)
print(i)
f=np.dot(b1,b3)
print(f) [3 4 3]
```

Ортогональный базис

[3 4 3]

[-2.79411765 1.94117647 0.20588235]

[-0.62025316 -1.1164557 2.10886076]

Скалярные произведения

-6.661338147750939e-16

-1.3322676295501878e-15

0.0

Последние три числа можно считать равными нулю.

Можно этот базис еще и нормировать, то есть каждый вектор поделить на его длину, тогда получим ортонормированный базис. Предоставляем решить эту задачу читателям самостоятельно.

ПРЯМЫЕ НА ПЛОСКОСТИ В ПРОСТРАНСТВЕ

Прямая на плоскости задается линейным уравнением, то есть уравнением, в которое переменные x, y входят в степени не больше первой. Перечислим различные виды уравнений, задающих прямую линию на плоскости: уравнение прямой, проходящей через 2 точки $\frac{x-x_0}{x_1-x_0}=\frac{y-y_0}{y_1-y_0}$; уравнение прямой через угловой коэффициент $y = kx + b$; общее уравнение прямой $ax + by + c = 0$; вектор, перпендикулярный прямой, нормальный вектор прямой $\vec{n}=(a;b)$, вектор параллельный прямой, направляющий вектор $\vec{p}=(-b;a)$ прямой .

90. Написать уравнение прямой проходящей через две заданные точки $M_1(5,3)$, $M_2(-2,4)$. Представить это уравнение через угловой коэффициент, в общем виде, найти координаты направляющего и нормального векторов.

Решение

```
print("Координаты точки A(x1;y1):")
x1 = float(input("\tx1 = "))
y1 = float(input("\ty1 = "))

print("Координаты точки B(x2;y2):")
x2 = float(input("\tx2 = "))
y2 = float(input("\ty2 = "))

print("Уравнение прямой, проходящей через эти точки:")
k = (y1 — y2) / (x1 — x2)
b = y2 — k*x2
print(" y = %.2f*x + %.2f" % (k, b))
print("Угловой коэффициент равен"), round(k,2)
Координаты точки A(x1;y1):
x1 = 5
y1 = 3
Координаты точки B(x2;y2):
x2 = –2
y2 = 4
Уравнение прямой, проходящей через эти точки:
y = -0.14*x + 3.71
Угловой коэффициент равен
(None, -0.14)
print («Перенесем все в одну сторону и получим общее уравнение прямой линии: %.2f*x + y + (%.2f)=0,» % (-k, -b))
print(«Направляющий вектор имеет координаты («),round(-k,2),(«,»), 1,(«)»)
```

print(«Нормальный вектор имеет координаты («), 1 ,(«,»), round(k,2) ,(«)»)

Перенесем все в одну сторону и получим общее уравнение прямой линии: 0.14*x + y + (-3.71)=0,

Направляющий вектор имеет координаты (

Нормальный вектор имеет координаты (

(None, 1, ‘,’, -0.14, ‘)’)

91. Найти угловые коэффициенты прямых линий параллельных и перпендикулярных данной $2x + 4y - 3 = 0$.

Решение

Выразим из уравнения y, найдем угловой коэффициент прямой k. У прямых, параллельных данной, угловой коэффициент равен $k_1 = k$. А у перпендикулярных данной прямых угловой коэффициент равен $k_2 = -\frac{1}{k}$.

print(“Коэффициенты :»)

A = float(input(“A = “))

B = float(input(“B = “))

C = float(input(“C = “))

Коэффициенты:

A = 2

B = 4

C = -3

k1=-A/B

k2=-1/k1

print («Угловой коэффициент прямых, параллельных данной прямой равен»), k1

print («Угловой коэффициент прямых, перпендикулярных данной прямой равен»), k2

Угловой коэффициент прямых, параллельных данной прямой равен -0.5

Угловой коэффициент прямых, перпендикулярных данной прямой равен 2.0.

РЕШИТЬ ЗАДАЧИ САМОСТОЯТЕЛЬНО

92. Проверить принадлежность данных точек одной прямой: а) A (3; 4; 1), B(1; 0; −1) и C(−2;−6; −4); б) A(3; 2; −1), B(5; −1; 4) и C(1;−7; 12). (*Указание*: использовать коллинеарность векторов, лежащих на одной прямой).

93. Даны векторы $\vec{a} = (-3; 4; -1)$, $\vec{b} = (-1; 2; 3)$ и $\vec{c} = (-4; -2; 1)$. Найти вектор $x = 4\vec{a} + 3\vec{b} - 2\vec{c}$ и вектор $\vec{y} = -5\vec{a} + 4\vec{b} + \vec{c}$. (*Указание*: выполнение действий с векторами сводится к выполнению этих же действий с соответствующими координатами).

94. Известно, что $|\vec{a}| = 5$, $|\vec{b}| = 6$. Найти скалярное произведение векторов $\vec{a}$ и $\vec{b}$, если угол φ между ними равен: а) 45°; б) 60°; в) 135°.

95. Найти векторное произведение $\vec{a} \times \vec{b}$, если известно:

a. $\vec{a} = 2\vec{i} + 3\vec{j} + 5\vec{k}, \vec{b} = \vec{i} + 2\vec{j} + \vec{k}$

b. $\vec{a} = 5\vec{i} - 4\vec{j} + 7\vec{k}, \vec{b} = \vec{i} + \vec{j} - 2\vec{k}$

c. $\vec{a} = (-1; 2; 4)$, $\vec{b} = (2; -1; -4)$

d. $\vec{a} = (7; -9; 3)$, $\vec{b} = (-1; 4; 0)$.

96. Вычислить площадь параллелограмма, построенного на векторах $\vec{a}$ и $\vec{b}$ как на сторонах, если $\vec{a} = (3; -2; 6)$, $\vec{b} = (6; 3; -2)$. (*Указание*: воспользоваться геометрическим смыслом модуля вектора векторного произведения).

97. Вычислить площадь треугольника с вершинами в точках A (−3; −2; −4), B (−1; −4; −7) и C (1; −2; 2). (*Указание*: площадь треугольника ABC равна половине площади параллелограмма, построенного на векторах AB и AC).

98. Даны векторы $\vec{a} = \vec{i} + 2\vec{j} - \vec{k}$ и $\vec{b} = 3\vec{i} - \vec{j} - 2\vec{k}$. Найти векторное произведение векторов $\vec{x}$ и $\vec{y}$, если $\vec{x} = 2\vec{a} - \vec{b}$, $\vec{y} = \vec{a} + 2\vec{b}$.

99. Найти смешанное произведение $\vec{a}\cdot\vec{b}\cdot\vec{c}$ векторов, если $\vec{a}=(1;\,1;\,2)$, $\vec{b}=(1;\,-2;\,3)$ и $\vec{c}=(2;\,1;\,1)$.

100. Проверить, будут ли векторы $\vec{a}=(1;\,2;\,2)$, $\vec{b}=(2;\,5;\,7)$ и $\vec{c}=(1;\,1;\,-1)$ компланарными.

101. Проверить, принадлежат ли точки A(3; 5; −4), B(1; −1; −3), C(7; 2; −6) и D(−1; 3; −2) одной плоскости. (*Указание*: используя данные точки, определить три вектора с общим началом и проверить для них условие компланарности).

102. Вычислить объём параллелепипеда, построенного на векторах $\vec{a}=(3;\,2;\,1)$, $\vec{b}=(1;\,0;\,-1)$ и $\vec{c}=(1;\,-2;\,1)$.

103. Вычислить высоту параллелепипеда, построенного на векторах $\vec{a}=\vec{i}+2\vec{j}-3\vec{k}$, $\vec{b}=-\vec{i}+\vec{j}+2\vec{k}$ и $\vec{c}=2\vec{i}-\vec{j}-\vec{k}$. За основание параллелепипеда взят параллелограмм, построенный на векторах $\vec{a}$ и $\vec{b}$.

104. Для системы векторов $\vec{a}_1=(1;-1;\,2;\,1)$, $\vec{a}_2=(2;\,1;\,-1;\,3)$, $\vec{a}_3=(6;\,0;\,2;\,8)$, $\vec{a}_4=(-1;\,-2;\,3;\,-2)$, $\vec{a}_5=(3;\,0;\,1;\,4)$ выделить максимальную линейно независимую подсистему и представить оставшиеся векторы в виде линейной комбинации из линейно независимых векторов.

105. Построить ортогональный базис, в который включён вектор $\vec{a}_1=(2;-2;\,1)$, и полученный базис заменить ортонормированным.

106. Дополнить данную систему векторов $\vec{a}_1=(6;-8;\,5;\,-10)$, $\vec{a}_2=(10;\,5;\,8;\,6)$ до ортогонального базиса и полученный базис заменить ортонормированным.

107. Доказать, что векторы $\vec{a}_1=(-1;\,2;\,1)$, $\vec{a}_2=(2;\,1;\,-2)$; $\vec{a}_3=(1;\,1;\,1)$ образуют базис и найти координаты вектора $\vec{x}=(-3;\,4;\,5)$ в этом базисе.

108. В «новом» базисе из векторов $\vec{a}_1=\vec{e}_1+2\vec{e}_2+3\vec{e}_3$; $\vec{a}_2=\vec{e}_1+\vec{e}_2+\vec{e}_3$; $\vec{a}_3=2\vec{e}_1+\vec{e}_2+\vec{e}_3$ вектор $\vec{x}$ имеет разложение: $\vec{x}=2\vec{a}_1+3\vec{a}_2-\vec{a}_3$. Найти координаты вектора $\vec{x}$ в «старом» базисе из векторов $\vec{e}_1,\vec{e}_2,\vec{e}_3$, а также матрицу перехода от базиса $\{\vec{e}_1,\vec{e}_2,\vec{e}_3\}$ к базису $\{\vec{a}_1,\vec{a}_2,\vec{a}_3\}$.

ЛИНЕЙНЫЕ ОПЕРАТОРЫ

Отображение φ называется линейным оператором, если оно удовлетворяет следующим условиям линейности:

1. $\varphi(X_1 + X_2) = \varphi(X_1) + \varphi(X_2)$,

2. $\varphi(kX) = k\varphi(X)$,

где X, X_1, X_2 любые векторы из V, k — любое число.

Произвольному n-мерному линейному оператору φ соответствует определенная квадратная матрица A порядка nxn.

λ — называется собственным значением линейного оператора, а X — собственным вектором этого оператора, соответствующим этому собственному значению, если

$$\varphi(\mathrm{X}) = \lambda \cdot \mathrm{X}$$

Собственные значения линейного оператора находятся из характеристического уравнения

$$\left| \boldsymbol{A} - \lambda \boldsymbol{E} \right| = 0 \ .$$

Собственные векторы можно найти, зная собственные значения.

Интересные практические применения в решении прикладных задач имеет задача приведения матрицы линейного оператора к диагональной форме. Если линейный оператор имеет различные собственные значения, то его матрицу можно привести к диагональному виду. Если собственные значения линейного оператора не все различны, то задача имеет решение при дополнительных условиях.

Для нахождения собственных значений используется команда np.linalg.eig(a)

109. Найти собственные значения и собственные векторы линейного оператора, заданного в некотором базисе трехмерного пространства матрицей.

$$\boldsymbol{a} = \begin{pmatrix} 0 & 1 & 0 \\ -4 & 4 & 0 \\ -2 & 1 & 2 \end{pmatrix}.$$

Решение.

```
a = np.array([[0, 1, 0], [-4,4,0],[-2,1,2]])
w = np.linalg.eig(a)
print (w)
```

(array([2., 2., 2.]), array([[0. , 0.42640143, 0.3554644],
[0. , 0.85280287, 0.71092881],
[1. , 0.30151134, 0.60681569]]))

110. В трехмерном пространстве отображение φ задано в некотором базисе матрицей A. Привести матрицу линейного оператора φ к диагональной форме, если это возможно.

$$A = \begin{pmatrix} -1 & 3 & -1 \\ -3 & 5 & -1 \\ -3 & 3 & 1 \end{pmatrix}$$

Решение.

Воспользуемся тем, что у матрицы, приведенной к диагональному виду на диагонали стоят собственные значения. Найдем собственные значения и составим матрицу, с собственными значениями на диагонали. Z = np.diag(w[0]) значит, что в нашем примере матрица Z будет состоять из собственных значений матрицы A по диагонали diag, а остальные элементы равны нулю.

```
A = np.array([[-1, 3, -1],[-3,5,-1],[-3,3,1]])
w = np.linalg.eig(A)
Z = np.diag(w[0])
print(Z)
```

[[1. 0. 0.]
[0. 2. 0.]
[0. 0. 2.]]

КВАДРАТИЧНЫЕ ФОРМЫ

Рассмотрим случай двух переменных. Квадратичной формой от двух переменных x_1, x_2 называется многочлен вида

$$F = a_{11} \cdot x_1^2 + 2a_{12} \cdot x_1 \cdot x_2 + a_{22} \cdot x_2^2$$

Матрица

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

называется матрицей квадратичной формы

Каноническая вид квадратичной формы — это форма, в которой отсутствует член с произведением $x_1 \cdot x_2$, то есть

$$F = a_1 \cdot x_1^{'2} + a_2 \cdot x_2^{'2}.$$

Любая квадратичная форма может быть приведена к каноническому виду. Для приведения квадратичной формы к каноническому виду можно, например, использовать метод Лагранжа [1].

Следующее уравнение называется характеристическим уравнением квадратичной формы

$$\begin{vmatrix} a_{11} - \lambda & a_{12} \\ a_{12} & a_{22} - \lambda \end{vmatrix} = 0,$$

а его решения λ_1, λ_2 собственными значениями матрицы квадратичной формы.

Квадратичная форма называется положительно (отрицательно) определенной, если при всех значениях переменных значение самой квадратичной формы положительные (отрицательные). Для того чтобы квадратичная форма была положительно (отрицательно) определенной необходимо и достаточно, чтобы все собственные значения её матрицы были положительные (отрицательные). Или критерий Сильвестра: для того чтобы квадратичная форма была положительно (отрицательно) определенной необхо-

димо и достаточно, чтобы все главные миноры её матрицы были положительные (знаки главных миноров чередовались, начиная со знака минус).

Рассмотрим решение задач.

111. Привести к каноническому виду квадратичную форму

$$\boldsymbol{F}\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right) = \boldsymbol{x}_1^2 - 3\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 + \boldsymbol{x}_2^2$$

Решение.

Выпишем матрицу данной квадратичной формы, найдем корни характеристического уравнения, то есть собственные значения матрицы квадратичной формы, полученные значения будут новыми коэффициентами. Канонический вид квадратичной формы определен неоднозначно, так как зависит от последовательности выбора ведущих переменных

```
a = np.array([[1, -1.5],[-1.5,1]])
w = np.linalg.eig(a)[0]
w
array([ 2.5, -0.5])
```

Ответ: $\boldsymbol{F} = -0.5\boldsymbol{x}_1^{'2} + 2.5\boldsymbol{x}_2^{'2}$.

112. Привести к каноническому виду квадратичную форму $5\boldsymbol{x}_1^2 + 4\boldsymbol{x}_1\boldsymbol{x}_2 + 2\boldsymbol{x}_2^2$.

Решение.

```
A= np.array([[5, 2],[2,2]])
w = np.linalg.eig(a)[0]
w
array([ 6., 1.])
```

Ответ: $\boldsymbol{F} = \boldsymbol{x}_1^{'2} + 6 \cdot \boldsymbol{x}_2^{'2}$

113. Доказать, что квадратичная форма $F = -5x_1^{\,2} + 6x_1 \cdot x_2 - 13x_2^{\,2}$ **является отрицательно определенной.**

Решение.

1 способ.

Можно решить задачу так. Запишем матрицу квадратичной формы, и найдем собственные значения. Если оба они отрицательные, то квадратичная форма отрицательно определена.

```
a = np.array([[-5, 3],[3,-13]])
w = np.linalg.eig(a)[0]
w
print (w «Оба с.з. отрицательные, значит квадратичная форма отрицательно определена»)
[ -4. -14.] Оба с.з. отрицательные, значит квадратичная форма отрицательно
определена
```

2 способ.

Решим задачу, используя критерий Сильвестра.

```
print ('$|a_{11}|=',a[0,0],'<0')
print ( 'det A=', np.linalg.det(a),'>0.
Знаки миноров чередуются, начиная с минуса, значит согласно критерию Сильвестра, квадратичная форма отрицательно определена'
$|a_{11}|= -5 <0)
```

det A= 56.0 >0. Знаки миноров чередуются, значит, согласно критерию Сильвестра, квадратичная форма отрицательно определена.

КРИВЫЕ ВТОРОГО ПОРЯДКА

Уравнение кривой второго порядка имеет вид

$$a_{11}x^2 + 2a_{12}xy + a_{22}y^2 + a_1x + a_2y + a_0 = 0,$$

где коэффициенты произвольные действительные числа. К кривым второго порядка относятся эллипс, гипербола и парабола.

Канонические уравнения:

эллипс $\frac{x}{a^2}+\frac{y}{b^2}=1$;

гипербола $\frac{x}{a^2}-\frac{y}{b^2}=1$;

парабола $y = 2px$.

Чтобы привести общее уравнение кривой второго порядка к каноническому виду, нужно произвести преобразования поворота и параллельного переноса. Формулы преобразования координат и объяснения процедуры приведения уравнения кривой второго порядка к каноническому виду подробно рассмотрены в [1].

К основным характеристикам кривых второго порядка относятся: большая и малая полуоси y эллипса a и b, действительная и мнимая полуоси y гиперболы a и b, эксцентриситет e, фокусное расстояние c.

Величины $\Delta_1 = a_{11} + a_{22}, \Delta_2 = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}, \Delta_3 = \begin{vmatrix} a_{11} & a_{12} & a_1 \\ a_{12} & a_{22} & a_2 \\ a_1 & a_2 & a_0 \end{vmatrix}$

не меняются при любом преобразовании координат общего уравнения кривой второго порядка и называются *инвариантами* общего уравнения кривой второго порядка относительно преобразования прямоугольных координат.

Каждому уравнению

$$a_{11}x^2 + 2a_{12}xy + a_{22}y^2 + a_1x + a_2y + a_0 = 0,$$

соответствует квадратичная форма

$$F = a_{11}x^2 + 2a_{12}xy + a_{22}y^2.$$

Кривая второго порядка будет иметь эллиптический, гиперболический или параболический тип, если выражение $\Delta_2 = \lambda_1 \cdot \lambda_2 = a_{11} \cdot a_{22} - a_{12}^2$ соответственно больше, меньше или равно нулю. λ_1, λ_2 — корни характеристического многочлена

$$\begin{vmatrix} a_{11} - \lambda & a_{12} \\ a_{12} & a_{22} - \lambda \end{vmatrix} = 0.$$

116. Найти параметры a, b, c, e эллипса, заданного уравнением $\frac{x^2}{16}+\frac{y^2}{9}=4$. Построить график.

Параметры эллипса находятся по формулам

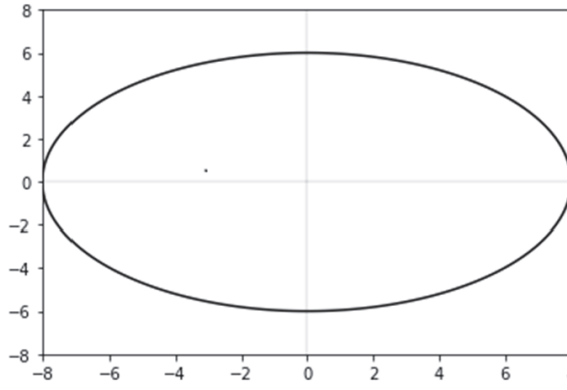
$$a=\sqrt{a^2}, b=\sqrt{b^2}, c^2=a^2-b^2, e=\frac{c}{a}.$$

Сначала введем коэффициенты *a, b, r*. Где *r* = 4.

Решение.

```
print(“Введите коэффициенты :»)
a = float(input(“a = “))
b = float(input(“b = “))
r = float(input(“r = “))
Введите коэффициенты :
a = 16
b = 9
r = 4
import math
def koe(a,b,r):
a1=math.sqrt(a*r)
b1=math.sqrt(b*r)
c=math.sqrt(a1*a1-b1*b1)
e=c/a1
print (‘a=’,round(a1,2),’b=’,round(b1,2),’c=’,round(c,2
),’e= ‘,round(e,2)
koe(a,b,r))
a= 16.97 b= 14.7 c= 8.49 e= 0.5
```

Для построения графика эллипса используем код, предложенный на сайте http://qaru.site/questions/504426/how-to-plot-an-ellipse-by-its-equation-on-python.



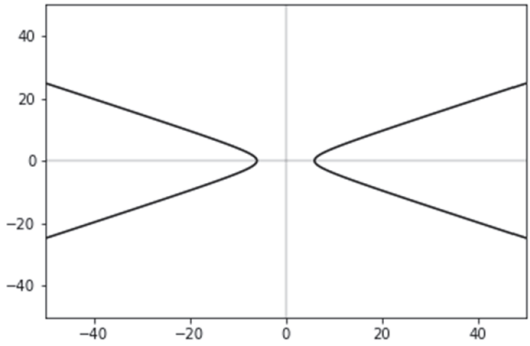
117. Найти параметры гиперболы *a, b, c, ε*, заданной уравнением $x^2 - 4y^2 = 36$, построить график.

Решение.

```
a=1
b=0.25
r=36
def kog(a,b,r):
a1=math.sqrt(a*r)
b1=math.sqrt(b*r)
c=math.sqrt(a1*a1+b1*b1)
e=c/a1
print ‘a=’,round(a1,2),’b=’,round(b1,2),’c=’,round(c,2)
,’e= ‘,round(e,2)
kog(a,b,r)
x = np.linspace(-50, 50, 400)
y = np.linspace(-50, 50, 400)
x, y = np.meshgrid(x, y)
def axes():
plt.axhline(0, alpha=.1)
plt.axvline(0, alpha=.1)
```

```
axes()
a = 6.
b = 3.
axes()
plt.contour(x, y,(x**2/a**2 — y**2/b**2), [1], colors='k')
plt.show()
```

a= 6.0 b= 3.0 c= 6.71 e= 1.12



118. Парабола с вершиной в начале координат проходит через точку A(9;3) и симметрично относительно оси *Ox*. Написать её каноническое уравнение, построить ее график.

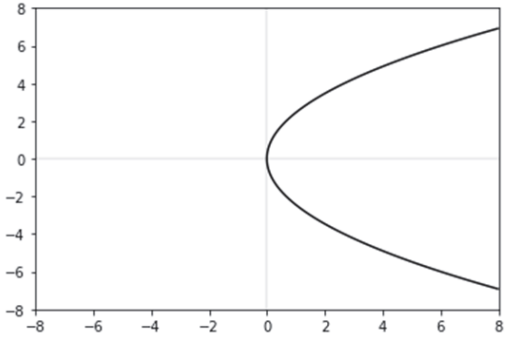
Решение.

Подставим координаты точки А в уравнение параболы $y^2 = 2px$

```
x_0=9
y_0=3
p=y_0^2/(2*x_0)
mpl.rcParams['lines.color'] = 'k'
mpl.rcParams['axes.prop_cycle'] = mpl.cycler('color', ['k'])
print ('Уравние параболы имеет вид: y^2=',2*p,'x')
```

```
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rcParams['lines.color'] = 'k'
mpl.rcParams['axes.prop_cycle'] = mpl.cycler('color', ['k'])
x = np.linspace(-8, 8, 400)
y = np.linspace(-8, 8, 400)
x, y = np.meshgrid(x, y)
def axes():
plt.axhline(0, alpha=.1)
plt.axvline(0, alpha=.1)
axes()
plt.contour(x, y, (y**2 — 2*p*x), [0], colors='k')
plt.show()
```

Уравнение параболы: $y^2 = 6x$, график имеет вид

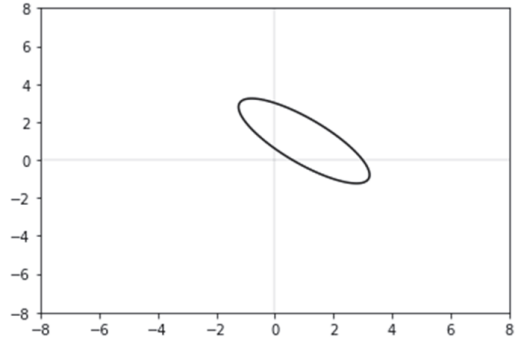


119. Привести уравнение кривой второго порядка к каноническому виду $5x^2 + 8xy + 5y^2 - 18x - 18y + 9 = 0$. Построить график.

Решение.

```
mport numpy as np
import matplotlib as mpl
```

```
import matplotlib.pyplot as plt
a, b, c, d, e, f = 5, 8, 5, -18, -18, 9
#
axes()
plt.contour(x, y,(a*x**2 + b*x*y + c*y**2 + d*x + e*y + f), [0], colors='k')
plt.show()
```



Мы видим, что данная кривая второго порядка является эллипсом. Действительно, выпишем определитель квадратичной формы без младших членов, которые отвечают за поворот эллипса

```
a = np.array([[5, 4],[4,5]])
k=np.linalg.det(a)
print (k)
w = np.linalg.eig(a)[0]
w
9.0
array([ 9., 1.])
```

Определитель равен 9>0, значит кривая имеет эллиптический тип. Собственные значения матрицы равны 9 и 1. Значит квадратичная форма $F = 5x^2 + 8xy + 5y^2$ в новой системе координат $x'Oy$ запишется в виде $F = \lambda_1 \cdot x'^2 + \lambda_2 \cdot y'^2 = 9x'^2 + y'^2$.

```
#from sympy import *
x = Symbol('x')
print (solveset(1/tan(2*x) , x, domain=S.Reals))
```

Мы использовали библиотеку sympy для решения тригонометрического уравнения $tan(2\varphi) = \frac{\{a_{\{11\}} - a_{\{22\}}\}}{2a_{\{12\}}} = 0$. Получили, что $\varphi = \pi/4$. Формулы преобразования координат примут вид.

```
k=math.cos(math.pi/4)
k1=math.sin(math.pi/4)
(0.7071067811865476, 0.7071067811865475)
```

Подставив в уравнение эллипса и выделив полный квадрат, получим

$$\{x''\}^2 + \frac{\{y''^2\}}{9} = 1.$$

Это каноническое уравнение в новой системе координат с центром в точке (1,2), оси координат повернуты на угол $\varphi = \pi/4$.

ПЛОСКОСТИ И ПРЯМЫЕ В ПРОСТРАНСТВЕ

Рассмотрим трехмерное векторное пространство. Через любые 3 точки можно провести одну и только одну плоскость. Любое линейное уравнение $Ax + By + Cz + D = 0$, где не все коэффициенты равны нулю, задает плоскость.

Виды уравнений плоскости в пространстве:

$$A(x - x_0) + B(y - y_0) + C(z - z_0) = 0 -$$

уравнение плоскости, проходящей через точку $M_0 = (x_0, y_0, z_0)$ и перпендикулярная вектору $\vec{n} = \{A, B, C\}$;

$Ax + By + Cz + D = 0 -$ общее уравнение плоскости;

$\frac{x}{a}+\frac{y}{b}+\frac{z}{c}=0$ – уравнение плоскости в «отрезках»;

$\begin{vmatrix} x-x_1 & y-y_1 & z-z_1 \\ x_2-x_1 & y_2-y_1 & z_2-z_1 \\ x_3-x_2 & y_3-y_2 & z_3-z_2 \end{vmatrix}=0$ – уравнение плоскости, проходящей через три заданные точки

$M_0=(x_0,y_0,z_0),\ M_1=(x_1,y_1,z_1),M_2=(x_2,y_2,z_2)$

имеет вид

Расстояние от точки $(x_1;y_1;z_1)$ до плоскости $Ax+By+Cz+D=0$ равно

$$d=\frac{|Ax_1+By_1+Cz_1+D|}{\sqrt{A^2+B^2+C^2}}.$$

Условие параллельности двух плоскостей. Две плоскости, заданные уравнениями $A_1x+B_1y+C_1z+D_1=0,\ A_2x+B_2y+C_2z+D_2=0$ параллельны, если координаты нормальных векторов этих плоскостей пропорциональны, то есть $\frac{A_1}{A_2}=\frac{B_1}{B_2}=\frac{C_1}{C_2}$. Если $\frac{A_1}{A_2}=\frac{B_1}{B_2}=\frac{C_1}{C_2}=\frac{D_1}{D_2}$, то плоскости совпадают.

Две плоскости, заданные уравнениями $A_1x+B_1y+C_1z+D_1=0,\ A_2x+B_2y+C_2z+D_2=0$ перпендикулярны (ортогональны), если перпендикулярны нормальные вектора этих плоскостей, то есть скалярное произведение нормальных векторов равно нулю $A_1\cdot A_2+B_1\cdot B_2+B_1\cdot B_2=0$.

Прямую в пространстве можно определить, как пересечение двух плоскостей, таким образом уравнение прямой в пространстве представляется как система, состоящая из уравнений двух плоскостей

$$\begin{cases} A_1x+B_1y+C_1z+D_1=0, \\ A_2x+B_2y+C_2z+D_2=0. \end{cases}$$

Это общее уравнение прямой линии в пространстве.

Каноническим уравнением прямой в пространстве

$$\frac{x - x_0}{m} = \frac{y - y_0}{n} = \frac{z - z_0}{q} -$$

уравнение прямой проходящей через точку параллельно вектору .

Рассмотрим задачи с решениями.

120. Записать уравнение плоскости, проходящей через точку (-1;5;1) перпендикулярно вектору {2;-4;3}.

Решение.

Подставим координаты точки и вектора в уравнение $A(x-x_0)+B(y-y_0)+C(z-z_0)=0$.

rint(«Введите координаты точки :»)

x = float(input(“x = “))

y = float(input(“y = “))

z = float(input(“z = “))

print(“Введите координаты вектора :»)

a = float(input(“a = “))

b = float(input(“b = “))

c = float(input(“c = “))

A=a

B=b

C=c

D=-a*x-b*y-c*z

print ‘Уравнение плоскости:’, A,’x+’, B ,’y+’,C,’z+’,D,’=0’

Введите координаты точки:

x = -1

y = 5

z = 1

Введите координаты вектора:

a = 2

b = -4

c = 3

Уравнение плоскости: 2.0 x+ -4.0 y+ 3.0 z+ 19.0 =0

121. Найти расстояние от точки М (1;0; -2) до плоскости 2x-y+2z-4=0.

Решение.

```
print(«Введите координаты точки М:»)
x = float(input(“x = “))
y = float(input(“y = “))
z = float(input(“z = “))
print(“Введите коэффициенты уравнения плоскости :»)
a = float(input(“a = “))
b = float(input(“b = “))
c = float(input(“c = “))
d = float(input(“d = “))
dist = math.fabs(a*x+b*y+c*z+d)/math.sqrt(a^2+b^2+c^2)
print (‘Расстояние от точки до плоскости равно:’, dist)
a = 2
b = -1
c = 2
d = -4
x = 1
y = 0
z = -2
```

Расстояние от точки до плоскости равно: 2.4494897427831783

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Найти собственные значения и собственные векторы линейного оператора, заданного в некотором базисе трехмерного пространства матрицей.

122. $\begin{pmatrix} 5 & 6 & -3 \\ -1 & 0 & 1 \\ 1 & 2 & -1 \end{pmatrix}$ 123. $\begin{pmatrix} 2 & -1 & 2 \\ 5 & -3 & 3 \\ -1 & 0 & -2 \end{pmatrix}$ 124. $\begin{pmatrix} 0 & 2 & 1 \\ -2 & 0 & 3 \\ -1 & -3 & 0 \end{pmatrix}$

125. $\begin{pmatrix} 3 & 1 & 0 \\ -4 & -1 & 0 \\ 4 & -8 & -2 \end{pmatrix}$ 122. $\begin{pmatrix} 5 & 6 & -3 \\ -1 & 0 & 1 \\ 1 & 2 & -1 \end{pmatrix}$ 126. $\begin{pmatrix} 2 & 5 & -6 \\ 4 & 6 & -9 \\ 3 & 6 & -8 \end{pmatrix}$

127. $\begin{pmatrix} -1 & 0 & 1 \\ 1 & 2 & -1 \\ 5 & 6 & -3 \end{pmatrix}$ 128. $\begin{pmatrix} -1 & 2 & 2 \\ -3 & 3 & 5 \\ 0 & -2 & -1 \end{pmatrix}$ 129. $\begin{pmatrix} 5 & 6 & -2 \\ 6 & 9 & -4 \\ 6 & 8 & -3 \end{pmatrix}$

В трехмерном пространстве отображение задано в некотором базисе матрицей. Привести матрицу линейного оператора к диагональной форме, если это возможно.

130. $\begin{pmatrix} 1 & 2 & 0 \\ 0 & 2 & 0 \\ -2 & -2 & -1 \end{pmatrix}$ 131. $\begin{pmatrix} 4 & 6 & 0 \\ -3 & -5 & 0 \\ -3 & -6 & 1 \end{pmatrix}$ 132. $\begin{pmatrix} 13 & 16 & 16 \\ -5 & -7 & -6 \\ -6 & -8 & -7 \end{pmatrix}$

133. $\begin{pmatrix} 3 & 0 & 8 \\ 3 & -1 & 6 \\ -2 & -0 & -5 \end{pmatrix}$ 134. $\begin{pmatrix} -4 & 2 & 10 \\ -4 & 3 & 7 \\ -3 & 1 & 7 \end{pmatrix}$ 135. $\begin{pmatrix} 7 & 12 & -2 \\ 3 & -4 & 0 \\ -2 & 0 & -2 \end{pmatrix}$

136. $\begin{pmatrix} -2 & 8 & 6 \\ -4 & 10 & 6 \\ 4 & -8 & -4 \end{pmatrix}$

ЛИТЕРАТУРА

1. Высшая математика: учебник и практикум для академического бакалавриата / М. Б. Хрипунова [и др.] ; под общ. ред. М. Б. Хрипуновой, И. И. Цыганок. — М. : Издательство Юрайт, 2017. — 478 с. — (Серия : Бакалавр. Академический курс).

2. *А. С. Балджы, М. Б. Хрипунова, Л. А.Шмелева*.Математическое моделирование в экономике и менеджменте на языке R — М.: Научный Консультант, 2016. — 59 с.

3. *Кремер, Н. Ш.* Эконометрика: учебник / Б. А. Путко, Н. Ш. Кремер — 3-е изд., перераб. и доп. — М. : ЮНИТИ-ДАНА, 2012 .— 329 с. — (Золотой фонд российских учебников) .— Под ред. Н. Ш. Кремера

4. *Доусон М.* Программируем на Python. — СПб.: Питер, 2014. — 416 с.

5. *Лутц М.* Изучаем Python, 4-е издание. — Пер. с англ. — СПб.: Символ-Плюс, 2011. — 1280 с.

6. *Лутц М.* Программирование на Python, том I, 4-е издание. — Пер. с англ. — СПб.: Символ-Плюс, 2011. — 992 с.

7. *Лутц М.* Программирование на Python, том II, 4-е издание. — Пер. с англ. — СПб.: Символ-Плюс, 2011. — 992 с.

8. *Прохоренок Н. А.* Python 3 и PyQt. Разработка приложений. — СПб.: БХВ-Петербург, 2012. — 704 с.

9. *Пилгрим М.* Погружение в Python 3 (Dive into Python 3 на русском)

10. *Прохоренок Н. А.* Самое необходимое. — СПб.: БХВ-Петербург, 2011. — 416 с.

11. *Хахаев И. А.* Практикум по алгоритмизации и программированию на Python. — М.: Альт Линукс, 2010. — 126 с. (Библиотека ALT Linux).

12. *Шабанов П. А.* Научная графика в python [Электронный ресурс]. URL: https://github.com/whitehorn/Scientific_graphics_in_python (31.08.2015).

БАЛДЖЫ Анна Сергеевна
ХРИПУНОВА Марина Борисовна
АЛЕКСАНДРОВА Ирина Александровна

МАТЕМАТИКА НА PYTHON
ЧАСТЬ I

ЭЛЕМЕНТЫ ЛИНЕЙНОЙ АЛГЕБРЫ И АНАЛИТИЧЕСКОЙ ГЕОМЕТРИИ

Учебно-методическое пособие

Компьютерная верстка *Яскульской И. Б.*
Дизайн обложки *Середы Т. В.*

Издательство «Прометей»
115035 Москва, ул. Садовническая, д. 72, стр. 1, офис 6
Тел./факс: +7 (495) 799-54-29
E-mail: info@prometej.su

Подписано в печать 10.05.2018.
Формат 60x84/16. Объем 4,75 п. л.
Тираж 500 экз. Заказ № 735.

ISBN 978-5-907003-86-6