Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки
Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота № 1

з дисципліни «Чисельні методи»

на тему

“Основи роботи в програмі MathCad”

Виконав:
студент гр. ІП-93
Завальнюк Максим

Викладач:
доц. Рибачук Л.В.

Київ – 2021

Зміст

1 Постановка задачі

Виконати всі вправи відповідно до програми роботи у MathCad та освоїти основи роботи в програмі MathCad.

2 Розв'язок

Я вирішив, що застосовувати чисельні методи у програмуванні буду за допомогою мови програмування Python, тому як математичний пакет я обрав NumPy, який є бібліотекою до Python. У даній лабораторній роботі моїм завданням було створення операцій у NumPy аналогічних до MathCad. Нижче представлені усі пояснення до прикладів.

Оскільки ніяких операцій із файлами у мене немає, то я одразу переходжу до Лістінгів, де проводяться операції з числами, матрицями, векторами, графіками.

Лістінг 1. Обчислення виразів.

```
>>> x = 12
>>> y = pow(x - 4, 2) + 5
>>> float(pow(x, y)) == np.power(12., 69.)
True
>>> y == np.sum([np.power(float(np.sum([x, -4])), 2.), 5])
True
```

Як видно, присвоєння у мові Python, а тому і в NumPy(тому що це бібліотека для Python) виглядає зовсім просто. Ми можемо присвоїти як і значення, так і вираз, обчислення. Функція pow() підносить у заданий степінь. На третьому рядку я порівнюю піднесення у степінь як це може зробити Python і піднесення у степінь, як це може NumPy(np). Результат – True(Істина). На п'ятому рядку я порівнюю наші прості обчислення у та обчислення за допомогою функцій тільки NumPy, тобто використання np.sum() замість оператора «+». Результат теж True.

Лістінг 2. Складання, віднімання, заперечення

```
>>> 3 + 4 - 5 == np.sum([3, 4, -5])
True
>>> -1 * -4 == np.multiply(-1, -4), -1 * -4 == np.negative(-4)
(True, True)
```

Тут знову показується сума із NumPy. Також перетворення негативного числа у додатне за допомогою множення на -1.

Лістінг 3. Ділення і множення

```
>>> 7 / 2 == np.divide(7, 2)
True
>>> 3 + 3 / 5 == np.sum([3, np.divide(3, 5)])
True
```

Тут ми можемо побачимо ділення із NumPy, а множення ще було застосоване у попередньому Лістінгу.

Лістінг 4. Факторіал та модуль

```
>>> math.factorial(7) == np.math.factorial(7)
True
>>> abs(-17) == np.abs(-17)
True
```

Порівнюємо факторіал та модуль чистого Python та NumPy.

Лістінг 5. Корені і ступінь

```
>>> math.sqrt(9) == np.sqrt(9)
True
>>> pow(8, (1 / 3)) == np.power(float(8), float(np.divide(1, 3)))
True
>>> math.e ** math.log(50, math.e) == np.power(np.e, float(np.log(50)))
True
```

Тут представлені дії з квадратним коренем(перший рядок), кубічний коренем(третій рядок) та логарифм(п’ятий рядок).

Лістінг 6. Дужки

```
>>> (1 + 3) * 4
16
```

Обчислення із дужками нічим не відрізняється у Python, і він добре з цим справляється.

Лістінг 7. Обчислювальні оператори

```
>>> main.get_sum(11, 2)
67
>>>
>>> main.get_product(11, 2)
79833600
```

Чистих таких функцій у NumPy немає, тому я написав їх самостійно, де сума та множення обраховується тільки за допомогою суми та множення NumPy. Вище представлені дані функції: $\sum_{i=2}^{n=11} i = 67$, $\prod_{i=2}^{n=11} i = 79833600$. Я їх розробив для того, щоб потім робити порівняння своїх результатів із ними.

Лістінг 8. Комплексне число та Лістінг 9. Функції та оператори для комплексних чисел.

```
>>> x = 6
>>> y = 2
>>> z = complex(x, y)
>>> z
(6+2j)
>>> z1 = 6 + 2j
>>> z2 = 21 * (cmath.e ** 0.7j)
>>> z.real
6.0
>>> z.imag
2.0
>>> np.iscomplex([z, z1, z2])
array([ True,  True,  True])
>>> z + z1, np.iscomplex(z + z1)
((12+4j), True)
```

У Python створення комплексних чисел не є важким, за допомогою complex(), і це можливо різними способами. На дев'ятому та одинадцятому показано як дістати реальну та уявну частини комплексного числа. У NumPy є метод iscomplex(), який перевіряє чи є число комплексним.

Лістінг 10. Операції над векторами і матрицями

```
>>> a = [[4, 6],
...      [5, 7]]
... a1 = np.matrix('4 6; 5 7')
>>> a == a1
matrix([[ True,  True],
        [ True,  True]])
>>> value = 3
>>> a1 = main.increase_matrix(a1, value)
>>> a == a1
matrix([[ True,  True],
        [ True,  True]])
>>> a * a1 == a1.dot(a1)
matrix([[ True,  True],
        [ True,  True]])
>>> a1.getT()
matrix([[12, 15],
        [18, 21]])
>>> np.dot(np.linalg.matrix_power(a1, -1), a1)
matrix([[1., 0.],
        [0., 1.]])
```

Отже, спочатку я створив матрицю через Python, а потім матрицю за допомогою NumPy та порівняв їх на п'ятому рядку. Далі я створив функцію, яка множить матрицю на число та знову порівняв з матрицею a(десятий рядок). На наступних рядках перевірив

множення матриці на матрицю(функція dot), отримав транспоновану(метод getT()), а також помножив матрицю на саму себе обернену.

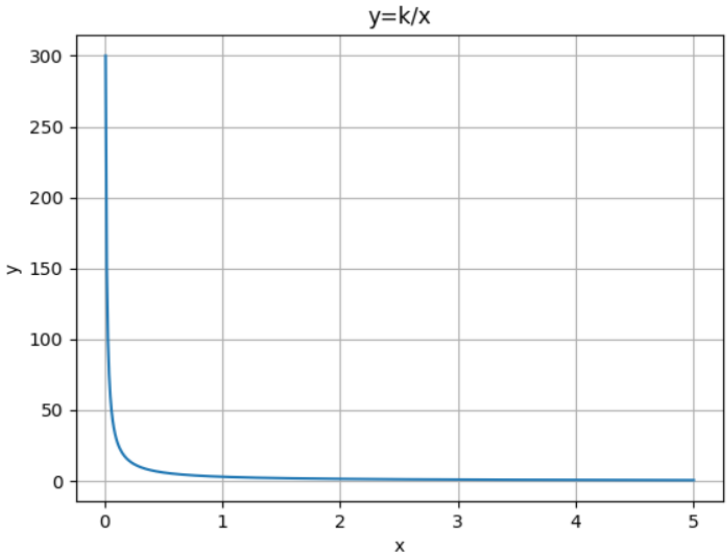
```
>>> a = np.array([1, 2, 3])
>>> b = np.array([3, 3, 2])
>>> np.vdot(a, b)
15
```

Також я створив два вектори($\bar{a}$ та $\bar{b}$) і вивів скалярний добуток векторів.

Робота з графікою

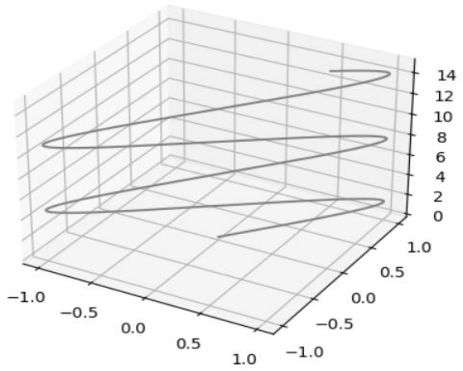
Безпосередньо NumPy не призначений для відображення графіки, тому існує бібліотека matplotlib, яка працює з даними від NumPy та відображає їх.

```
>>> k = 3
... t = np.arange(0.0, 5.01, 0.01)
... s = k / t
...
... fig, ax = plt.subplots()
... ax.plot(t, s)
...
... ax.set(xlabel='x', ylabel='y',
...        title='y=k/x')
... ax.grid()
...
... plt.show()
```



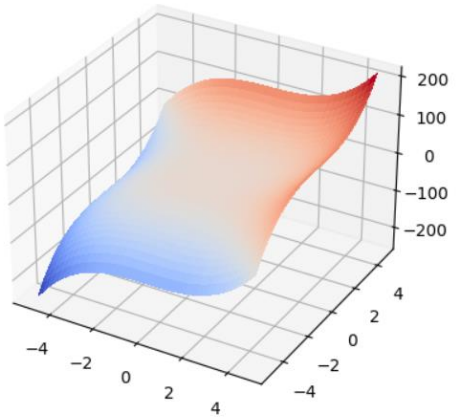
Я намагався відобразити функції y=k/x. Спочатку за допомогою NumPy я створив дані(другий рядок). Далі я додав x та y(у моєму випадку t та s) до графіку, і на останньому рядку відобразив це на площині.

```
>>> fig = plt.figure()
... ax = plt.axes(projection="3d")
...
... z_line = np.linspace(0, 15, 1000)
... x_line = np.sin(z_line)
... y_line = np.sin(z_line)
... ax.plot3D(x_line, y_line, z_line, 'gray'
... plt.show()
```



Також моїм завдання було створення тривимірних графіків. На щастя, ця бібліотека теж вміє таке відображати і результат ви можете бачити праворуч.

```
>>> fig = plt.figure()
... ax = fig.gca(projection='3d')
... # Make data.
... X = np.arange(-5, 5, 0.25)
... Y = np.arange(-5, 5, 0.25)
... X, Y = np.meshgrid(X, Y)
... Z = np.power(X, 3) + np.power(Y, 3)
... # Plot the surface.
... surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
...                        linewidth=0, antialiased=False)
... plt.show()
```



Ще один приклад тривимірної графіки.