ІНФОРМАТИКА ТА ПРОГРАМУВАННЯ

Тема 20. Наукові обчислення

Наукові обчислення

- **Науковими обчисленнями** будемо називати застосування програмування до розв’язання наукових задач.
- Наукові обчислення мають свою специфіку.
- Зокрема, це наявність великих обсягів числових даних, які треба обробляти з достатньою швидкістю.
- Звичайно, наукові обчислення передбачають також програмування введення/виведення, обробку рядків та даних інших типів.
- Але ці задачі є, як правило, допоміжними.

Наукові обчислення.2

- У Python для наукових обчислень використовують пакет NumPy (numpy).
- NumPy є скороченням від Numerical Python, тобто, «Числовий Python».
- Окрім цього, разом з NumPy використовують також пакет для графічної візуалізації наукових даних Matplotlib (matplotlib).
- Нарешті, для специфічних наукових задач застосовують оснований на NumPy пакет SciPy, або Scientific Python, тобто «Науковий Python».
- У даній темі ми обмежимось розглядом основних можливостей пакетів numpy та matplotlib.

Встановлення numpy та matplotlib

- Пакети numpy та matplotlib не входять до стандартної поставки Python, їх треба встановлювати окремо.
- На жаль, процес встановлення не є тривіальним, відрізняється для різних операційних систем та може забрати певний час.
- Для встановлення numpy та matplotlib у системах сімейства Microsoft Windows (Windows 7, Windows 8, Windows 10) простіше використати готові «бінарні» інсталяції.
- Вони розраховані на роботу у середовищі Python 3.4. Посилання на відповідні пакети зібрані у файлі
  - http://web.cs.wpi.edu/~cs1004/a14/Resources/Windows/SettingUpPython_Windows.docx
- Встановлення numpy та matplotlib вимагає дотримання певних угод при встановленні Python 3.4:
  1. Треба встановлювати 32-розрядну версію Python 3.4 (а не 64-розрядну).
  2. Python 3.4 треба встановлювати тільки для поточного користувача (для себе), а не для всіх користувачів.

Встановлення numpy та matplotlib.2

- За умови дотримання зазначених вище угод, встановлення numpy та matplotlib полягає у запуску відповідних завантажених .exe файлів та слідуванні інтерактивним підказкам.
- Якщо Python 3.4 був встановлений без дотримання зазначених вище угод, його потрібно перевстановити.
- Треба зазначити, що пакет matplotlib залежить від декількох додаткових пакетів (перелік цих пакетів є у файлі за посиланням вище), які встановлюють аналогічно.
- Інсталяцію numpy та matplotlib для операційних систем Mac OS X та Linux описано у файлі
  - http://web.cs.wpi.edu/~cs1004/a14/Resources/Macintosh/SettingUpPython_Macintosh-Linux.pdf
- Для цієї інсталяції застосовують пакет з Python з назвою pip (pip3 для Python 3.4).
- pip – це скорочення від Python Installation Program – Програма встановлення Python.

Імпорт numpy та matplotlib

- Для використання у програмі пакети numpy та matplotlib імпортують командами:

```
import numpy as np
import matplotlib.pyplot as plt
```

- Скорочені імена np та plt не є обов’язковими, але ці назви традиційно використовують у більшості програм, що працюють з numpy та matplotlib.
- Звичайно matplotlib імпортують тільки тоді, коли потрібне графічне відображення результатів.
- Також з matplotlib, як випливає з команди імпорту, частіше імпортують тільки модуль pyplot, хоча у цьому пакеті є і інші модулі.

Масиви у NumPy

- Для збереження даних NumPy використовує масиви.
- Ми вже розглядали реалізацію масивів на базі списків Python у темі «Списки».
- Практично всі задачі наукових обчислень можна розв’язувати із застосуванням таких масивів.
- На жаль, у масивів на базі списків є один «мінус»: програмний код, створений з використанням списків, виконується доволі повільно.
- Це не дається взнаки при відносно невеликих обсягах даних. Але коли ці дані складаються з сотень тисяч або мільйонів одиниць, уповільнення може стати критичним.
- Інший мінус – великий обсяг пам’яті, який займають елементи масиву на базі списку.
- Ідея створення NumPy полягала у тому, щоб запропонувати більш прості однорідні масиви, які можна швидко обробляти завдяки програмним бібліотекам, складеним у мовах програмування C та Fortran.
- Також ці масиви потребують суттєво меншого об’єму пам’яті комп’ютера.

Масиви у NumPy.2

- Є багато способів створити масив у numpy. Перший спосіб - виконати команду

a = np.array(t)

  - де a – ім’я масиву, t – вираз типу, що ітерується, частіше за все, - список.
- Наприклад,

a = np.array([2, 4, 6])

- створює одновимірний масив з 3 цілими елементами: 2, 4, 6.
- Інший спосіб створення масиву у numpy – завдання діапазону аналогічно раніше розглянутому об’єкту range(i, j, k).

x = np.arange(i, j, k)

- утворює масив, починаючи з i, до j виключно з кроком k.
- Але, на відміну від об’єкту range, значення k може бути й дійсним числом.

Масиви у NumPy.3

- Наприклад,

x = np.arange(0.0, 2.0, 0.1)

- створює масив з елементами, починаючи від 0.0 до 1.9 через 0.1.
- Кількість елементів масиву при використанні дійсного кроку може варіюватись через те, що операції над дійсними числами виконуються наближено.
- Для створення масиву дійсних чисел з n елементів, які рівномірно розподілені на відрізку [a, b] використовують

x = np.linspace(a, b, n)

- Масиви у numpy – це об’єкти класу ndarray.

Тип елементів масиву у numpy

- Усі елементи масиву у numpy є однотипними.
- Типи елементів масивів у numpy можуть бути:
  - дійсними (float, float32, float64)
  - цілими (int, int 32, int 64, int128)
  - комплексними (complex, complex64, complex 128)
  - бульовими (bool, bool8)
  - іншими
- Для певного типу можна вказувати кількість біт, що виділяються для даних цього типу (8, 32, 64, 128).
- Дані цілого типу, на відміну від стандартних цілих у Python, обмежені. Обмеження залежить від кількості біт, що виділяються.
- Для того, щоб визначити тип елементів масиву, використовують атрибут dtype.
- Наприклад,

a.dtype

- повертає поточний тип елементів масиву a.

Тип елементів масиву у numpy.2

- Тип елементів встановлюється при створенні масиву. Якщо вказано елементи масиву, то тип визначається цими елементами.
- Наприклад, якщо всі елементи цілі, то тип також буде цілим.

```
a = np.array([2, 4, 6])
```

- Якщо є хоча б одне дійсне число, то тип буде дійсним.

```
a = np.array([2, 4.0, 6])
```

- Тип елементів можна явно вказати при створенні масиву, застосувавши ключовий параметр dtype.
- Наприклад,

```
a = np.array([2, 4, 6], dtype = np.float)
```

- створює масив з елементами дійсного типу.
- Значеннями параметру dtype можуть бути стандартні константи numpy (np.float, np.int тощо) або рядки (‘float’, ‘int’ тощо).
- Типом елементів масиву за угодою є дійсний тип (float64).

Приклад

- Обчислення інтервалу значень вектору для списків та масивів numpy.
- Інтервал – це різниця між максимальним та мінімальним значенням.
- Порівняємо час обчислення інтервалу для списку та масиву numpy.
- Створимо масив у вигляді списку з випадкових величин між 0 та 1.
- Створимо масив numpy на базі цього списку та обчислимо інтервал значень для списку та масиву.
- У першій версії ми порівнюємо час виконання за суб’єктивними відчуттями.
- У другій версії для порівняння використаємо декоратор @benchmark, який був побудований у темі «Декоратори».
- Порівняння швидкодії показує, що інтервал для масиву numpy обчислюється у 15-20 разів швидше, ніж інтервал для списку.

Створення масивів нулів та одиниць

- Часто виникає ситуація, коли розмір майбутнього масиву відомий, а сам масив заповнюється пізніше.
- У таких випадках використовують функції для створення масиву нулів або одиниць.
- Для створення масиву з n нулів:

```
a = np.zeros(n)
```

- Для створення масиву з n одиниць:

```
a = np.ones(n)
```

- Типом елементів у такому масиві буде дійсний тип.
- Можна змінити тип даних за угодою, використавши ключовий параметр dtype.
- Кількість елементів масиву numpy можна повернути за допомогою поля size:

```
a.size
```

Доступ до елементів масивів numpy

- Для доступу до елемента масиву numpy, як і для списків, використовують квадратні дужки, у яких вказують індекс елемента.
- Індекси починаються з 0.
- Тобто, a[1] повертає другий елемент масиву a.

Приклад: табулювання функції (версія 1)

- Отримати масив значень функції f(x) на відрізку [a, b] у n точках.
- Розв’язує цю задачу функція tabulate(f, a, b, n), яка використовує допоміжну функцію gety(f, x) для отримання масиву значень функції f для заданого масиву точок x.
- Програма будує масив значень функції для власної функції fun та для стандартної функції sin з модуля math.
- Для великої кількості точок (> 100 000) помітно, що побудова масиву відбувається із затримкою, і ця затримка більша для власної функції fun.

Виконання операцій над масивами numpy

- Дії над масивами numpy можна виконувати поелементно. Але у numpy є й можливість виконати операцію над усім масивом.
- Наприклад, для масивів x, x1, x2

x*2

- множить кожен елемент x на 2, а

x1 + x2

- обчислює масив, елементами якого будуть суми відповідних елементів (звичайно, при цьому x1, x2 повинні мати однаковий розмір).
- Тобто,

```
y = x1 + x2      ≡      for i in range(x1.size):
                            y[i] = x1[i] + x2[i]
```

- Те ж саме справедливо для всіх арифметичних операцій +, -, *, /, **, //, %.
- numpy містить також ряд математичних функцій, відомих нам за модулем math, які можуть виконувати відповідну дію для всіх елементів масиву.
- Наприклад, np.sin(x) за масивом x обчислює масив, що складається із значень функції sin відповідних елементів x.

Векторизація

- **Векторизація** – це побудова програмного коду, який працює над цілими масивами numpy, замість обробки окремих елементів.
- Векторизація базується на використанні арифметичних операцій та функцій, що можуть сприймати в якості параметрів як прості змінні, так і масиви.
- Векторизація дозволяє суттєво пришвидшити виконання програм, що працюють з великими обсягами даних.

Приклад: табулювання функції (версія 2)

- Отримати масив значень функції f(x) на відрізку [a, b] у n точках з використанням векторизації.
- Розв’язує цю задачу функція tabulate(f, a, b, n), яка використовує допоміжну функцію gety(f, x) для отримання масиву значень функції f для заданого масиву точок x.
- При цьому, функція gety стає тривіальною.
- Але gety розрахована на те, що функцію f вже векторизовано.
- Якщо на вхід потрапить функція, що не підтримує векторизацію, станеться помилка.
- Для векторизованих функцій їх табулювання здійснюється більш, ніж на порядок швидше у порівнянні з версією 1 програми.

Примусова векторизація функцій

- Якщо функція не є векторизованою, у numpy можна її векторизувати примусово. Для цього застосовують функцію vectorize.

np.vectorize(f)

- модифікує функцію f так, щоб вона приймала в якості параметрів звичайні змінні та масиви numpy.
- У подальших прикладах у цій темі ми будемо векторизувати наші програми для суттєвого підвищення їх швидкодії.

Приклад: табулювання функції (версія 3)

- Отримати масив значень функції f(x) на відрізку [a, b] у n точках з використанням векторизації.
- Забезпечити також табулювання для невекторизованих функцій.
- У порівнянні з попередніми версіями змінюється лише функція gety(f, x) для отримання масиву значень функції f для заданого масиву точок x.
- У цій функції ми додаємо блок обробки виключень try-except.
- Якщо при спробі обчислити векторизований варіант виникає помилка, ми у блоці обробки виконуємо невекторизований варіант коду, аналогічний версії 1 цього прикладу.

Зображення результатів обчислень у графічному вигляді

- Для зображення результатів наукових обчислень у графічному вигляді застосовують пакет matplotlib.
- Цей пакет має потужні засоби для зображення графіків функцій, діаграм, малюнків.
- Девізом розробників matplotlib є «намагатися робити прості речі легкими, а складні, - можливими».
- І дійсно, побудувати графік у matplotlib нескладно.
- matplotlib базується на numpy.
- Масиви numpy є джерелом даних для побудови графіків matplotlib.
- matplotlib надає функціональний та об’єктно орієнтований інтерфейс.
- Тобто, можна використовувати функції matplotlib, не звертаючи уваги на ієрархію класів, хоча для більш складних графічних зображень треба мати уявлення щодо основних класів matplotlib.
- matplotlib містить декілька модулів, з яких частіше використовують модуль pyplot.

Зображення результатів обчислень у графічному вигляді.2

- Як було вже зазначено, pyplot, як правило, імпортують командою

```
import matplotlib.pyplot as plt
```

- Після імпорту, маючи точки у масиві x, а значення функції у точках у масиві y, побудувати графік можна буквально двома командами:

```
plt.plot(x, y)
plt.show()
```

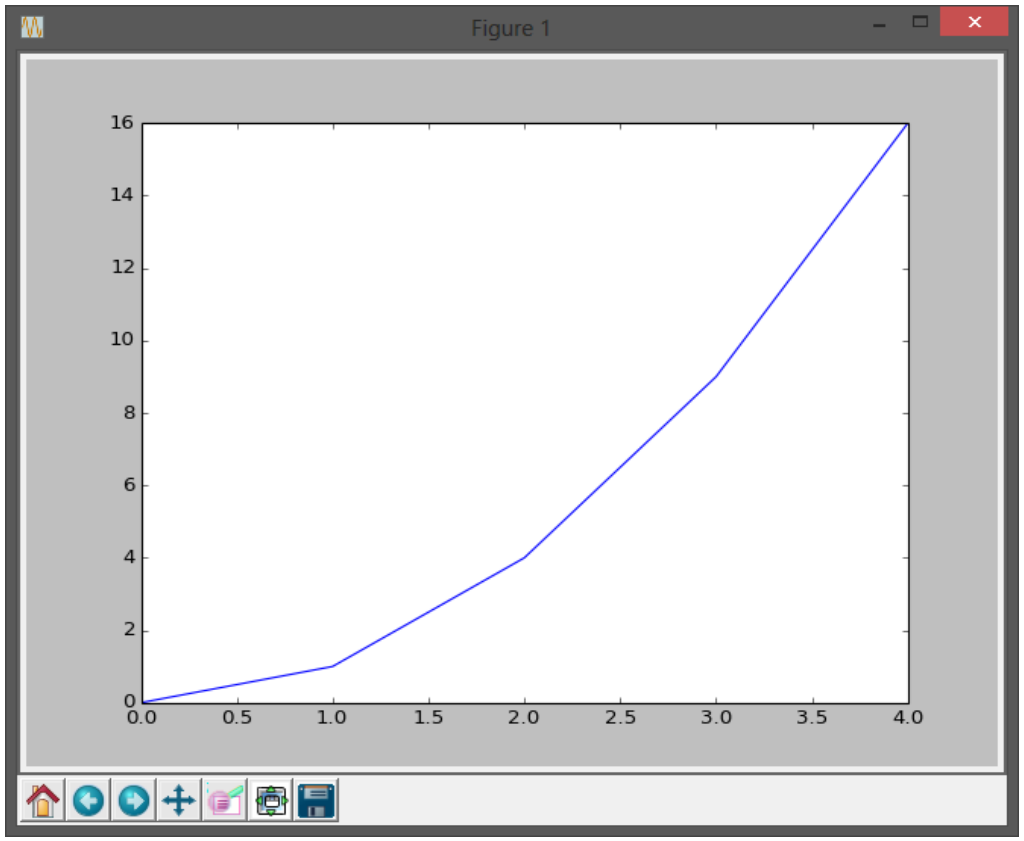
- Команда plot готує графік для зображення, а show, - показує інтерактивне вікно з графіком та кнопками для керування.
- Наприклад, послідовність команд

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(5)
y = x**2
plt.plot(x, y)
plt.show()
```

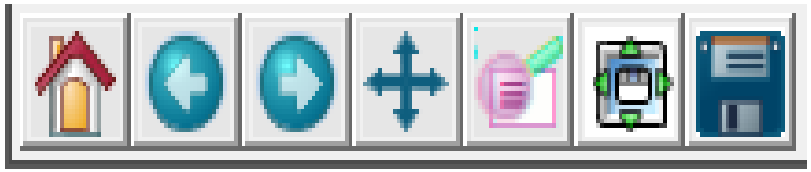
- показує вікно з графіком функції $y = x^2$ у 5 точках: 0, 1, 2, 3, 4.

Зображення результатів обчислень у графічному вигляді.3

- Головну частину вікна займає сам графік. При цьому, matplotlib самостійно підбирає масштаб осей так, щоб увесь графік розмістився у вікні.



Зображення результатів обчислень у графічному вигляді.3



- Призначення кнопок керування (зліва направо):
  - перехід до початкового зображення (при зміні зображень у вікні)
  - попереднє зображення (при зміні зображень у вікні)
  - наступне зображення (при зміні зображень у вікні)
  - пересування та масштабування осей за допомогою «миші»
  - укрупнення зображення до вибраного прямокутника
  - налаштування підмалюнків
  - збереження зображення у файл

Приклад: Зображення графіків функцій (версія 1)

- Зобразити графік функції f(x), для якої є дані у n точках на відрізку [a, b].
- Для зображення графіку використаємо раніше створені підпрограми табулювання функції.
- Побудуємо графікі для функції fun та стандартної функції sin.
- Побудову здійснює функція plotfunc1.
- Кожний графік з’являється у окремому вікні, причому поки ми не закриємо перше вікно, друге не з’явиться.
- Така поведінка не завжди бажана, оскільки у випадку декількох графіків, для їх аналізу треба переглядати ці графіки одночасно.

Приклад: Зображення графіків функцій (версія 2)

- Зобразити графіки декількох функцій, для яких є дані у n точках на відрізку [a, b].
- Побудову здійснює функція plotfunc2(a, b, n, *f), яка приймає в якості параметрів границі відрізку та кількість точок, а також змінну кількість функцій.
- plotfunc2 табулює першу функцію зі списку параметрів, а для інших отримує їх значення по вже готовому масиву x.
- Програма використовує plotfunc2 для зображення графіків функцій fun та sin(x)/x, яку довизначено у точці 0.

Зображення заголовку, легенди, міток осей та фіксація осей

- На графіках часто потрібно зобразити додаткову інформацію: заголовок, легенду, мітки осей.
- Часом треба також зафіксувати проміжки, на яких будуть зображені графіки, щоб уникнути автоматичного масштабування, яке робить matplotlib.
- Функції, які дозволяють це виконати, зібрані у таблиці

Функція	Опис
plt.title(s)	Зображує заголовок рисунку s. s - рядок
plt.axis([xmin,xmax,ymin,ymax])	Фіксує осі x та y на відрізках від xmin до xmax та від ymin до ymax
plt.xlabel(sx)	Зображує мітку sx осі x. sx - рядок
plt.ylabel(sy)	Зображує мітку sy осі y. sy - рядок
plt.legend(lglist)	Зображує легенду по кожному графіку на рисунку. lglist – список, що містить рядки для виведення у якості легенди.

Вибір кольору та стилю ліній

- Колір та стиль лінії можуть бути задані, як параметри plot, безпосередньо після даних у рядку, який містить символи для стилю та кольору.
- Деякі символи для стилю та кольору ліній наведені у таблицях нижче

Символ(и) стилю	Опис
'-'	суцільна лінія
'--'	пунктир
'-.'	пунктир з точками
':'	лінія точками
'o'	маркер кола
'v'	маркер трикутника вершиною донизу
's'	маркер квадрата
'p'	маркер п’ятикутника
'*'	маркер зірочки

Вибір кольору та стилю ліній.2

Символ кольору	Колір
‘b’	синій
‘g’	зелений
‘r’	червоний
‘c’	блакитний
‘m’	пурпурний
‘y’	жовтий
‘k’	чорний
‘w’	білий

- Наприклад, щоб задати зображення графіку пурпурними точками, треба вказати рядок ‘:m’:

```
plt.plot(x, y, ':m')
```

Приклад: Зображення графіків функцій (версія 3)

- Зобразити графіки декількох функцій, для яких є дані у n точках на відрізку [a, b].
- Зобразити заголовок, легенду, мітки осей. Вибрати різні стилі ліній та кольори.
- Закріпити вісь x на відрізку [0, 7] та вісь y на відрізку [0, 2].
- Побудову здійснює функція plotfunc3(a, b, n, *f) та основна програма.

Зображення декількох підграфіків на одному рисунку

- Замість того, щоб зображувати декілька функцій на одному графіку, інколи виникає необхідність зобразити поруч декілька графіків, але кожен у своїй координатній системі.
- Будемо називати такі графікі **підграфіками**. У термінах matplotlib підграфіки називаються subplots.
- Кожен підграфік зображується так само як і окремий графік зі своїми даними, легендою, осями тощо.
- Усі підграфіки на одному рисунку утворюють сітку.
- У простому випадку це матриця mxn.
- Для того, щоб почати працювати з k-м підграфіком, треба виконати команду

plt.subplot(m, n, k)

  - де m – кількість підграфіків по вертикалі, n – кількість підграфіків по горизонталі, k – загальний порядковий номер підграфіку від 1 до mxn.

Зображення декількох підграфіків на одному рисунку.2

- Наприклад, якщо m = 3, n = 2, то підграфікі будуть розташовані на рисунку так:

1	2
3	4
5	6

- Тоді

plt.subplot(3, 2, 5)

- означає підграфік у 3 рядку та 1 стовпчику.

Перенесення осей

- matplotlib за угодою зображує осі координат та засічки на осях зліва, справа, знизу та згори графіку.
- При цьому позначки на осях зображуються зліва та знизу.
- У деяких випадках осі доцільно розташувати у початку координат, щоб графік набув звичного вигляду.
- Для того, щоб зрозуміти, як цього досягти, нам потрібно коротко розглянути частину об'єктної моделі matplotlib.
- matplotlib для кожного рисунку має головний об'єкт класу figure.
- Цей об'єкт містить декілька об'єктів-осей, класу axes.
- Кожний об'єкт класу axes, у свою чергу містить самі осі а також дані функцій, графік яких побудований у цих осях.
- Щоб повернути поточний (активний) об'єкт класу axes, треба використати функцію gca():

```
ax = plt.gca()
```

- Самі лінії, що зображують осі, називаються spines, а засічки, - ticks.

Перенесення осей.2

- Загалом переміщення осей у початок координат можна виконати за допомогою такої функції:

```
def movespinesticks():
    '''Перемістити осі у нульову позицію '''
    ax = plt.gca()  #отримати поточний об'єкт класу axes
    # зробити праву та верхню осі невидимими:
    ax.spines['top'].set_color('none')
    ax.spines['right'].set_color('none')
    # перенести нижню вісь у позицію y=0:
    ax.xaxis.set_ticks_position('bottom')
    ax.spines['bottom'].set_position(('data',0))
    # перенести ліву вісь у позицію x == 0:
    ax.yaxis.set_ticks_position('left')
    ax.spines['left'].set_position(('data',0))
```

Приклад: Зображення графіків функцій (версія 4)

- Зобразити графіки декількох функцій, для яких є дані у n точках на відрізку [a, b], на окремих підграфіках.
- Зобразити заголовок, легенду, мітки осей.
- Вибрати різні стилі ліній та кольори.
- Перенести осі у нульову позицію для кожного підграфіку.
- Побудову здійснює функція plotfunc4(a, b, n, *f) та основна програма.
- Підграфіки розміщуються вертикально у один стовпчик.
- Перенесення осей здійснюється функцією movespinesticks.

Приклад: Зображення графіків функцій (версія 4).2

- У функції plotfunc4 трохи по-іншому встановлюється легенда.
- Надписи легенди передаються у кожний підграфік у функції plot за допомогою ключового параметру label.
- Сама ж легенда

```
plt.plot(x, y, style, label = ff.__doc__)
#створити графік з легендою
plt.legend(loc = 'best')
#встановити легенду
```

Багатовимірні масиви

- До цього часу ми розглядали тільки одновимірні масиви. Але у numpy є і багатовимірні масиви.
- Створити багатовимірний масив можна з відповідного масиву на базі списку або перетворивши одновимірний масив у багатовимірний.
- Наприклад,

```
a = np.array([[1, 2, 3],[4, 5, 6]])
```

- створює двовимірний масив розміром 2x3.
- Як і раніше, a.size повертає кількість елементів всього масиву.
- В той же час, для багатовимірних масивів len(a) повертає тільки кількість елементів по першому індексу.

Багатовимірні масиви.2

- Для роботи з багатовимірними масивами numpy надає ряд додаткових засобів.

a.ndim

- повертає кількість вимірів масиву (для нашого прикладу - 2)

a.shape

- повертає/змінює розмір масиву для кожного виміру і водночас показує кількість вимірів (для нашого прикладу повертає кортеж (2, 3)).
- За допомогою shape можна перетворити одновимірний масив на багатовимірний.
- Наприклад, послідовність команд

b = np.array([1, 2, 3, 4, 5, 6])

b.shape = (2, 3)

- створює такий же масив 2x3 та з тими ж елементами, що й раніше створений масив a.
- Для багатовимірних масивів визначені ті ж арифметичні операції та функції, що й для одновимірних.

Індексація багатовимірних масивів

- Для індексації багатовимірних масивів numpy можна використовувати той же синтаксис, що й для багатовимірних масивів на базі списків.
- Наприклад, b[1][2] для раніше створеного масиву b повертає третій елемент другого рядка (індексація починається з 0).
- Але для масивів numpy визначений також інший синтаксис індексації елементів, через кому: b[1, 2].
- Останній спосіб є більш ефективним, оскільки b[1][2] спочатку створює тимчасовий масив для b[1], а потім бере з нього третій елемент.
- Аналогічно можемо присвоїти довільному елементу масиву нове значення. Наприклад, b[1, 2] = 10.

Вирізки індексів та вибір підмасивів

- У масивах numpy є можливість задавати вирізки аналогічно тому, як це робиться для рядків або списків.
- Вирізка вибирає частину масиву (підмасив).
- Наприклад, для масиву

a = np.array([[1, 2, 3],[4, 5, 6]])

- a[1, 1:3] повертає масив, що складається з елементів 5, 6.
- a[:, 1] повертає другий стовпчик матриці (масив з елементів 2, 5).
- a[-1, :] повертає останній рядок матриці (4, 5, 6).
- Вирізки дозволяють не тільки повертати, але й змінювати частину масиву однією командою.
- Наприклад,
- a[:, 1] = 10 присвоює всім елементам другого стовпчика значення 10.

Індексні масиви

- Для «точкової» вибірки декількох елементів масиву застосовують індексні масиви.
- Індексний масив – це одновимірний масив numpy, який складається з цілих чисел.
- Далі цей індексний масив (масиви) можна вказати у квадратних дужках замість індексу, після чого будуть відібрані елементи масиву з відповідними індексами.
- Наприклад, після

```
c = np.array([1, 5, 6, 23, 17, 88])
idx = np.array([0, 1, 1, 4, 2])
```

- застосування

```
c[idx]
```

- поверне масив [1, 5, 5, 17, 6]
- Індекси у індексному масиві можуть йти у довільному порядку та входити декілька разів.
- Для багатовимірних масивів індексні масиви можна комбінувати з вирізками для різних індексів.

Відношення та бульові операції для масивів numpy

- Над масивами numpy можна виконувати не тільки арифметичні, але й бульові операції, а також обчислювати відношення.
- Відношення позначаються стандартним чином.
- Наприклад, для визначеного вище масиву c відношення c >= 20 повертає масив бульових величин такого ж розміру, що й c.
- На місцях елементів масиву c більших 20, у новому масиві будуть стояти величини True, а на інших місцях, - False ([False, False, False, True, False, True]).
- Так само поелементно обчислюються відношення між двома масивами.
- Бульові операції, що допускають векторизацію, тобто, можуть використовуватись для масивів, позначаються спеціальним чином:.
- np.logical_and(g1, g2) – кон'юнкція умов g1, g2, які можуть містити масиви;
- np.logical_or(g1, g2) – диз'юнкція умов g1, g2, які можуть містити масиви.
- np.logical_not(g1) – заперечення умови g1, яка може містити масиви.

Бульова індексація для масивів numpy

- Бульові вирази у numpy слугують також для індексації масивів.
- Якщо замість індексу вказати бульовий вираз, то будуть відібрані ті елементи масиву, для яких значення цього виразу є істинними.
- Наприклад, після створення масиву

```
c = np.array([1, 5, 6, 23, 17, 88])
```

- вираз

```
c[c >= 20]
```

- поверне масив тих елементів c, які більші або рівні 20

Функції all, any та sum у numpy

- У numpy, як і у стандартному Python, є функції all, any та sum.
- Ці функції можуть приймати в якості параметрів масиви.
- Функція np.all(x) є істинною (True) тоді, коли усі елементи x є істинними (як розуміють істинність у Python).
- Функція np.any(x) є істинною (True) тоді, коли хоча б один елемент x є істинним.
- Функція np.sum(x) повертає суму всіх елементів x.
- Для багатовимірних масивів можна вибрати один вимір, по якому застосовуються ці функції.
- Тоді вони повертають не один результат, а масив результатів.
- Вибір виміру здійснюється ключовим параметром axis (вісь), який може набувати значень від 0 до (m-1), де m – кількість вимірів масиву.

Функції all, any та sum у numpy.2

- Якщо axis = 0, це означає перебір усіх значень першого індексу масиву.
- Якщо axis = 1, це означає перебір усіх значень другого індексу, тощо.
- Тобто, для двовимірного масиву axis = 0 – це застосування відповідної функції для всіх стовпчиків матриці.
- Наприклад, після

```
a = np.array([[1, 2, 3],[4, 5, 6])
```

- застосування

```
np.sum(a, axis = 0)
```

- повертає масив сум елементів стовпчиків матриці a.
- Функцію np.sum також часто використовують для підрахунку кількісті істинних елементів у масиві.
- Справа в тому, що np.sum трактує кожний істинний елемент як 1, а хибний, - як 0.

Векторно-матричні операції

- У numpy визначено ряд векторно-матричних операцій. Основні з них наведені у таблиці нижче.

Дія	Опис
np.dot(a, b)	Повертає добуток матриць a, b або добуток вектору на матрицю, або добуток матриці на вектор, або добуток двох векторів.
np.transpose(a)	Повертає транспоновану матрицю по відношенню до матриці a
np.eye(n)	Повертає одиничну матрицю nxn
np.diag(a)	Повертає одновимірний масив діагональних елементів матриці a
np.fill_diagonal(a, x)	Заповнює діагональні елементи матриці a значенням x

Поширення масивів

- Поширення масивів – ще одна потужна функціональність numpy.
- Поширення (broadcasting) означає підлаштування одного масиву під інший при виконанні операцій над масивами.
- Саме завдяки поширенню є можливість виконувати арифметичні операції, наприклад, над масивом та скаляром.
- Якщо розмір одного масиву по деякому індексу дорівнює 1, то цей масив поширюється (віртуально дублюється) так, щоб його розмір по цьому індексу відповідав розміру іншого масиву.
- Наприклад, для масивів a та b

```
a = np.array([[1, 2, 3],[4, 5, 6]])
b = np.array([5, 10, 15])
```

- їх добуток

```
a * b
```

- повертає масив

```
[[ 5, 20, 45],
 [20, 50, 90]]
```

- Тобто, через поширення кожний рядок матриці a поелементно множиться на елементи вектора b.

Зміна розмірності та/або розміру масивів

- Раніше було розглянуто зміну розмірності масиву за допомогою поля shape.
- Змінити розмірність також можна за допомогою np.newaxis.
- Наприклад,

b[:, np.newaxis]

- повертає вектор-стовпчик або матрицю 3x1.
- Такі перетворення розмірності часто використовують для подальшого поширення масиву у потрібному напрямку.
- Декілька масивів можна об’єднати в один.

np.vstack((a, b))

- об’єднує 2 масиви, приписуючи масив b «під» масивом a.

np.hstack((a, b))

- об’єднує 2 масиви, приписуючи масив b «праворуч» від масиву a.
- Для об’єднання масиви повинні мати рівні розміри за відповідним індексом.

Приклад: Розв'язування системи лінійних алгебраїчних рівнянь методом Якобі (методом простої ітерації) (версія 1)

- Розв'язати систему лінійних алгебраїчних рівнянь методом Якобі (методом простої ітерації).
- Система лінійних алгебраїчних рівнянь має вигляд:

$$Ax = b$$

- де A – матриця коефіцієнтів, x – вектор невідомих, b – вектор-стовпчик вільних членів.
- Для розв'язування таких систем існує багато методів. Ми розглянемо один з ітераційних методів. А саме, - метод простої ітерації (метод Якобі).
- Ідея цього методу полягає у перетворенні системи та обчисленні наближеного значення x за допомогою рекурентного співвідношення

$$x^{(k+1)} = g - Cx^{(k)}$$

  - де $x^{(k+1)}$, $x^{(k)}$ - значення x на (k+1) та (k) кроці, C – перетворена матриця A, g – перетворений вектор b.

Приклад: Розв'язування системи лінійних алгебраїчних рівнянь методом Якобі (методом простої ітерації) (версія 1).2

- Для перетворення кожен рядок матриці *A* та відповідний елемент вектору *b* ділимо на діагональний елемент a_{ii}.
- Після цього у трансформованій матриці діагональні елементи (одиниці) замінюємо нулями.
- Таким чином отримуємо матрицю *C* та вектор *g*.
- В якості $x^{(0)}$ беруть або нульовий вектор, або стовпчик вільних членів *g*.
- Умова завершення обчислень – достатня близькість двох послідовних наближень.
- *Треба відмітити, що цей метод працює та збігається (ітераційний процес завершується), коли матриця є діагонально-переважаючою.*
- *Тобто, модулі діагональних елементів набагато більші за інші елементи матриці.*
- *Більш точно, достатньою умовою збіжності є*

$$\forall i \left|a_{ii}\right| \geq \sum_{i \neq j} \left|a_{ij}\right|, \exists i : \left|a_{ii}\right| > \sum_{i \neq j} \left|a_{ij}\right|$$

Приклад: Розв'язування системи лінійних алгебраїчних рівнянь методом Якобі (методом простої ітерації) (версія 1).3

- Програма ініціалізує матрицю A та вектор b, друкує систему та виконує обчислення x за наведеним вище рекурентним співвідношенням.
- Для перевірки умови завершення обчислень використовується функція np.allclose, яка перевіряє, чи всі модулі різниць відповідних елементів двох послідовних наближень не перевищують значення atol.
- Відразу намітимо напрями покращання цієї програми:
  1. Коефіцієнти матриці та вектор вільних членів бажано вводити ззовні.
  2. Не перевіряється умова збіжності.
  3. Бажано здійснити векторизацію коду, позбавившись циклу у обчисленні чергового наближення.

Введення та виведення масивів numpy у текстовий файл

- Масиви numpy доволі просто зберігати у текстовий файл та читати з текстового файлу.
- Для збереження у текстовий файл використовується функція np.savetxt. У простому випадку ця функція виглядає так:

np.savetxt(filename, x)

  - де filename – ім'я файлу, x – масив для збереження.
- Двовимірні масиви зберігаються наступним чином: кожен рядок масиву – у окремому рядку файлу; окремі елементи відділяються одним або декількома пропусками (‘ ’).
- Для читання масиву з текстового файлу треба виконати команду:

x = np.loadtxt(filename)

Приклад: Розв'язування системи лінійних алгебраїчних рівнянь методом Якобі (методом простої ітерації) (версія 2)

- Розв'язати систему лінійних алгебраїчних рівнянь методом Якобі (методом простої ітерації).
- Виконати читання системи з текстового файлу.
- Здійснити також перевірку збіжності метода для заданої системи.
- У програмі виділено 3 функції:
  - print_system – показує систему на екрані;
  - check_convergence – перевіряє достатню умову збіжності;
  - jacobi – виконує пошук розв'язку.
- Основна програма читає систему з файлу та знаходить розв'язок.
- Передбачається, що у текстовому файлі один рядок системи записаний у одному рядку файлу.
- Спочатку – коефіцієнти a_{ij}, потім - b_i.
- Тому після читання програма за допомогою вирізок ділить масив ab на матрицю a та вектор b.

Приклад: Розв'язування системи лінійних алгебраїчних рівнянь методом Якобі (методом простої ітерації) (версія 3)

- Розв'язати систему лінійних алгебраїчних рівнянь методом Якобі (методом простої ітерації).
- Виконати читання системи з текстового файлу.
- Здійснити також перевірку збіжності метода для заданої системи.
- Векторизувати програмний код.
- У порівнянні з версією 2 у версії 3 виконано векторизацію функцій check_convergence та jacobi.
- Для векторизації використано розглянуті вище можливості обробки масивів numpy.
- Застосовано більш надійний критерій завершення обчислень (розрахунок суми модулів різниці двох послідовних наближень).
- Розв'язок порівнюється з розв'язком цієї ж системи за допомогою стандартної функції numpy для розв'язування систем лінійних алгебраїчних рівнянь: np.linalg.solve(a,b).

Генерація випадкових чисел та їх використання

- Ми вже знайомились з генерацією випадкових чисел у Python, яка реалізована у модулі random.
- У наукових обчисленнях випадкові числа використовують у низці застосувань для розв’язання різноманітних задач.
- Оскільки у більшості цих задач потрібна генерація великих масивів випадкових чисел, у numpy є свій модуль, який також називається random і багато в чому повторює функціональність random із стандартного Python.

Генерація випадкових чисел та їх використання.2

- У таблиці нижче зібрано декілька функцій np.random.

Дія	Опис
np.random.uniform(low, high, size)	Повертає масив з size рівномірно розподілених випадкових чисел у напівінтервалі [low, high)
np.random.rand(d0,…, dk)	Повертає (багатовимірний) масив, заповнений випадковими числами з напівінтервалу [0, 1). d0,…,dk – кількість елементів по кожному з вимірів
np.random.random_integers(low, high, size)	Повертає масив з size рівномірно розподілених випадкових цілих чисел у інтервалі [low, high]
np.random.shuffle(a)	Випадковим чином перемішує масив a. Якщо масив багатовимірний, то перемішує тільки за першим індексом.
np.random.choice(a, size, replace=True)	Випадковим чином вибирає з одновимірного масиву a size елементів та повертає масив розміру size. Параметр replace вказує, чи допускається повторний вибір одного й того ж елементу.

Приклад: Кидання монети (версія 1)

- Промоделювати кидання монети та прослідкувати, як змінюється із збільшенням числа випробувань різниця між кількістю «орлів» та «решок» а також співвідношення між орлами та решками.
- Зобразити результати на графіку.
- Для моделювання будемо вважати, що результатом 1 кидання монети може бути число 1 (орел) або 0 (решка).
- Проведемо n серій випробувань, у k-ій з яких буде 2^k кидань монети.
- Усі результати випробувань одразу збережемо у масиві flips.
- На базі масиву flips побудуємо масиви
  - numheads #кількість орлів
  - numtails #кількість решок
  - diffs #різниці між орлами та решками
  - ratios #відношення кількості орлів до кількості решок
- Зобразимо графіки різниці та відношення між орлами та решками у вигляді ліній на 2 підграфіках.
- Таке зображення неінформативне.

Приклад: Кидання монети (версія 2)

- Промоделювати кидання монети та прослідкувати, як змінюється із збільшенням числа випробувань різниця між кількістю «орлів» та «решок» а також співвідношення між орлами та решками.
- Зобразити результати точками на графіку.
- У порівнянні з версією 1, змінено тільки відображення підграфіків: вони зображені окремими точками.

Приклад: Кидання монети (версія 3)

- Промоделювати кидання монети та прослідкувати, як змінюється із збільшенням числа випробувань різниця між кількістю «орлів» та «решок» а також ймовірність випадання орлів.
- Зобразити результати точками на графіку.
- Використати логарифмічну шкалу по осі x.
- У порівнянні з версією 2, змінено відображення підграфіків, а також масив відношень (ratios) замінено масивом приблизних значень ймовірності (probs).

Метод Монте-Карло

- Приклад з киданням монети є ілюстрацією метода Монте-Карло.
- **Метод Монте-Карло** полягає у проведенні великої кількості випробувань з генерацією випадкових величин. На підставі результатів випробувань робиться висновок щодо певної гіпотези: обчислення ймовірності події, обчислення площі тощо.
- Точніше, нехай є деяка непевна подія, яка може бути результатом випробування з деякою ймовірністю.
- Тоді, якщо після проведення N випробувань така подія виникає M раз, то наближено ймовірність даної події складає M/N.
- Із збільшенням N точність наближення також зростає.

Ймовірність випадання «шісток»: задача Шевальє де Мера

- Один відомий французький гравець сімнадцятого століття, Шевальє де Мер, звернувся до свого друга Блеза Паскаля з таким запитанням щодо ставок на певний результат кидання костей.
- Він довгий час ставив на те, що при киданні однієї кості 4 рази хоча б один раз випаде «шістка», тобто, на горі буде поверхня з шістьма крапками.

Ймовірність випадання «шісток»: задача Шевальє де Мера.2

- При цьому Шевальє де Мер залишався у виграші.
- Він вирішив, що так само буде у виграші, якщо буде ставити на те, що при киданні двох костей 24 рази хоча б один раз випадуть дві шістки, але почав програвати гроші.
- Питання було в тому, яка мінімальна кількість кидань 2 костей зі ставкою на одночасне випадання двох шісток буде приносити виграш?
- Паскаль разом з Ферма розв’язали цю задачу, що, як визнається нині, поклало початок теорії ймовірності.

Приклад: кидання костей (версія 1)

- За допомогою методу Монте-Карло визначити ймовірність одночасного випадання ndice шісток при одночасному киданні ndice костей nroll раз.
- Перша версія програми використовує масиви на базі списків та генерацію випадкових чисел з модуля random.
- Програма показує наближену ймовірність випадання шісток на підставі TEST_NUM випробувань для кількості кидань від 1 до nroll.

Приклад: кидання костей (версія 2)

- За допомогою методу Монте-Карло визначити ймовірність одночасного випадання ndice шісток при одночасному киданні ndice костей nroll раз.
- Друга версія програми використовує масиви numpy та генерацію випадкових чисел з модуля np.random.
- Програма показує наближену ймовірність випадання шісток на підставі TEST_NUM випробувань для кількості кидань від 1 до nroll.

Зображення гістограм

- matplotlib має функціональність для зображення гістограм даних.
- Ця функціональність використовується у задачах статистики та економічних розрахунках.
- Зображення гістограми здійснюється за допомогою функції hist:

plt.hist(a, bins=n_bins, color=c)

  - де a – це одновимірний масив даних, bins – ключовий параметр, що вказує кількість діапазонів, color – ключовий параметр, що вказує колір гістограми.
- matplotlib ділить всі дані з a по bins діапазонах, рахує кількість результатів, що потрапили до і-го діапазону та зображує ці кількості у вигляді гістограми.

Приклад: кидання костей (версія 3)

- За допомогою методу Монте-Карло визначити ймовірність одночасного випадання ndice шісток при одночасному киданні ndice костей nroll раз.
- Зобразити гістограму розподілу сум з ndice костей у випробуваннях та гістограму кількостей одночасного випадання ndice костей у nroll киданнях.
- Третя версія програми відрізняється від другої тільки тим, що зображує гістограми при максимальній кількості кидань nroll.
- Для зображення використовується функція draw_histogram.

Інтегрування методом Монте-Карло

- Метод Монте-Карло використовується не тільки для обчислення ймовірності непевної події, але й для обчислення детермінованих величин шляхом проведення великої кількості випробувань.
- Одне з таких застосувань – обчислення визначеного інтегралу функції на відрізку.
- Для того, щоб обчислити визначений інтеграл функції $f(x)$ на відрізку $[a, b]$ методом Монте-Карло, проводять n випробувань, у яких отримують випадкові точки x_i, рівномірно розподілені на відрізку $[a, b]$.
- Після цього обчислюють суму

$$\sum_{i=1}^{n} f(x_i) * h, \partial e\, h = \frac{b-a}{n}$$

- яка і є наближенням інтегралу.

Приклад: обчислення визначеного інтегралу методом Монте-Карло

- Виконати обчислення визначеного інтегралу функції на відрізку методом Монте-Карло та порівняти з інтегруванням методом трапецій.
- Програма містить функції trap_integral та mc_integral, які здійснюють інтегрування заданої функції відповідно методом трапецій та методом Монте-Карло на відрізку [*a*, *b*] у n точках.
- В обох функціях використовуються масиви numpy.
- У функції mc_integral також використовується модуль np.random для генерації точок.

Обчислення площ фігур методом Монте-Карло

- Як розвиток обчислення інтегралу, метод Монте-Карло використовується також для обчислення площ складних фігур.
- Нехай ми маємо деяку фігуру, обмежену графіками 2 відомих функцій, що мають точки перетину.
- Нехай ми також можемо побудувати прямокутник, що повністю містить нашу фігуру.
- Тоді використання методу Монте-Карло для обчислення площі фігури полягає у наступному.
  - Генеруємо n випадкових точок, які рівномірно розподілені у нашому прямокутнику.
  - Для кожної точки перевіряємо, чи належить вона нашій фігурі.
  - Отримуємо кількість таких точок m.
  - Обчислюємо площу прямокутника S.
  - Обчислюємо наближено площу фігури за формулою: S*m/n.

Приклад: обчислення площі фігури між двома кривими

- Обчислити методом Монте-Карло площу фігури між кривими e**(1/x) та a*x**2+b*x+c при заданих a, b, c.
- Для того, щоб фігура була скінченною, значення параметра a повинно бути менше 0.
- Окрім цього, рівняння a*x**2+b*x+c = 0 повинно мати дійсні корені.
- Тоді охоплюючий прямокутник можна побудувати, взявши за xmin близьке до 0 число, а за xmax значення більшого кореня рівняння.
- За ymin можемо взяти 0, а за ymax, - максимум a*x**2+b*x+c.
- Таким чином, отримуємо охоплюючий прямокутник. Його будує функція get_box_bounds.

Приклад: обчислення площі фігури між двома кривими.2

- Функція mc_square безпосередньо обчислює площу фігури між f1 та f2 методом Моне-Карло.
- Кількість точок, що потрапляють у фігуру обчислюється за допомогою бульової індексації.
- Функція trinomial повертає функцію, що обчислює значення квадратного тричлена при заданих a, b, c.
- Функція exp1x обчислює e**(1/x).
- Функція plotf1f2 зображує графікі двох функцій, що утворюють фігуру, охоплюючого прямокутника, а також заливає область нашої фігури кольором.
- Заливка виконується функцією fill_between з matplotlib:

```
plt.fill_between(x, y1, y2, where = y1 <= y2, facecolor = 'cyan')
```

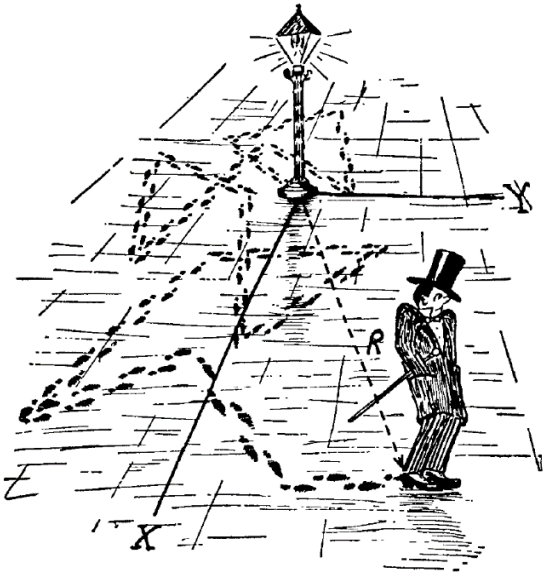
- Ключовий параметр where вказує умову заповнення площі між графіками.
- Функція movespinesticks зсуває осі у нульові позиції, була розглянута раніше.

Випадковий шлях

- Нехай є частинка, яка може рухатись вздовж осі *OX*. У кожен момент часу частинка може рухатись на 1 у додатному або від'ємному напрямку з рівною ймовірністю ½.
- Якщо у початковий момент часу частинка знаходиться у точці 0, то через N кроків вона перейде у точку k.
- Такий процес називається випадковим шляхом, або «ходою п'яниці».
- У двовимірному варіанті частинка може рівноймовірно рухатись на 1 по чотирьох напрямках: по осях *OX* та *OY*.

Випадковий шлях.2

- Випадковий шлях використовують для моделювання різноманітних процесів: від фізичних до економічних.
- Як правило, у моделях розглядають не одну, а багато частинок, які можуть бути на початку розташовані у одній позиції або ж у різних, у тому числі, випадкових, позиціях.



Приклад: моделювання поведінки молекул газу у прямокутній області (версія 1)

- Нехай молекули газу представлені частинками, що рухаються у двовимірному просторі згідно правил випадкового шляху.
- Є прямокутна область, поділена спочатку на дві прямокутних частини стінкою.
- Одна частина рівномірно заповнена молекулами, а інша, - порожня.
- Потім стінку знімають і молекули починають вільно рухатись по всій області.
- Змоделювати рух заданої кількості молекул впродовж заданої кількості кроків.
- Показати результат (стан моделі) через кожні m кроків.

Моделювання поведінки молекул газу. Реалізація

- Опишемо клас Drunkard2D, який моделює рух частинок у двовимірному просторі.
- Причому, область може бути обмежена прямокутником або необмежена.
- Клас має поля:
  - _n_d - кількість точок (частинок)
  - _is_limited - чи обмежена область
  - _bounds - границі області (прямокутник – кортеж (xmin, ymin, xmax, ymax))
  - _pos – поточні позиції всіх точок (частинок) – двовимірний масив numpy розміром (2, _n_d)
  - _dirs – двовимірний масив numpy можливих рухів для кожної частинки на деякому кроці
  - fig_count - номер рисунку для збереження у файлі

Моделювання поведінки молекул газу. Реалізація.2

- Клас також містить властивості
  - bounds – читати/змінити границі області
  - pos – читати поточні позиції всіх точок
- та методи
  - __init__ - конструктор
  - step - зробити один крок у моделюванні
  - msteps - зробити m кроків у моделюванні
  - plot - побудувати графік стану моделі без його зображення
  - show - побудувати та показати графік стану моделі
  - savefig - побудувати та зберегти графік стану моделі у файлі за допомогою функції plt.savefig з matplotlib.
- Внутрішній метод _push_into_bounds гарантує, що після кожного кроку всі точки знаходяться в межах області (якщо область обмежена).

Моделювання поведінки молекул газу. Реалізація.3

```
class Drunkard2D:
    '''Клас, що реалізує випадковий шлях у двовимірному просторі ("хода п'яниці")'''
    def __init__(self, num_drunkards, init_pos = None, is_limited = False, bounds = None):
        self._n_d = num_drunkards   #кількість точок
        self._is_limited = is_limited
#чи обмежена область
        self._bounds = bounds
#границі області (прямокутник)
         self._pos = init_pos       #позиції всіх точок
 if self._pos is None:
            #якщо позиції не задано, встановлюємо всі у (0,0)
            self._pos = np.zeros(self._n_d * 2)
            self._pos.shape = (2, self._n_d)
```

Моделювання поведінки молекул газу. Реалізація.4

```
        if self._is_limited:
            #якщо задано границі, встановлюємо всі точки у область
            xmin, ymin, xmax, ymax = self._bounds
            x = (xmin + xmax) // 2
            y = (ymin + ymax) // 2
            self._pos += np.array([[x], [y]])

        self._dirs = np.array([[-1, 0], [0, -1], [1, 0], [0, 1]]) #можливі рухи
        self._dirs = np.transpose(self._dirs) #зручніше мати
транспонований масив
        self.fig_count = 0  #номер рисунку

        plt.hold(False)   #кожного разу буде малювати нове зображення,
                          #а не доповнювати попереднє
```

Моделювання поведінки молекул газу. Реалізація.5

```
@property
def bounds(self):
    '''Властивість границі (читання).'''
    return self._bounds

@bounds.setter
def bounds(self, new_bounds):
    '''Властивість границі (встановлення).'''
    self._bounds = new_bounds

@property
def pos(self):
    '''Властивість позиції точок (тільки читання).'''
    return self._pos
```

Моделювання поведінки молекул газу. Реалізація.6

```
def _push_into_bounds(self):
    '''Повернути всі точки у межі області.'''
    xmin, ymin, xmax, ymax = self._bounds
    self.pos[0][self.pos[0] < xmin] = xmin + 1
    self.pos[0][self.pos[0] > xmax] = xmax - 1
    self.pos[1][self.pos[1] < ymin] = ymin + 1
    self.pos[1][self.pos[1] > ymax] = ymax - 1

def step(self):
    '''Зробити один крок у моделюванні.'''
    #масив індексів для подальшого формування масиву приростів
    ids = np.random.random_integers(0, 3, self._n_d)

    #масив приростів чергового кроку
    dxy = self._dirs[:,ids]
    self._pos += dxy
    if self._is_limited:
        self._push_into_bounds()
```

Моделювання поведінки молекул газу. Реалізація.7

```
def msteps(self, m):
    '''Зробити m кроків у моделюванні.'''
    for i in range(m):
        self.step()

def plot(self):
    '''Побудувати графік стану моделі.'''
    #set axes
    if self._bounds is None:
        xmin = ymin = -100
        xmax = ymax = 100
    else:
        xmin, ymin, xmax, ymax = self._bounds
    plt.plot(self._pos[0], self._pos[1], 'ob')
    plt.axis([xmin, xmax, ymin, ymax])
```

Моделювання поведінки молекул газу. Реалізація.8

```
def show(self):
    '''Побудувати та показати графік стану моделі.'''
    self.plot()
    plt.show()

def savefig(self, path):
    '''Побудувати та зберегти графік стану моделі у файлі.

    path - шлях до файлу, включаючи фінальний символ
    поділу каталогів ('/' або '\').
    Файл має ім'я відповідно масці
    tmpXXXXX.png, деXXXXX - номер рисунку
    '''
    self.fig_count += 1
    fname = "tmp{:0>5}.png".format(self.fig_count)
    self.plot()
    plt.savefig(path + fname)
```

Моделювання поведінки молекул газу. Реалізація.9

- Основна програма вводить кількість частинок, загальну кількість кроків, границі області та границю стінки.
- Рівномірно розподіляє частинки по області з урахуванням стінки, будує масив з початковими координатами частинок та створює об’єкт класу Drunkard2D.
- Далі здійснює моделювання і через кожних 100 кроків показує результат на графіку та зберігає графік у файл.

Приклад: моделювання поведінки молекул газу у прямокутній області (версія 2)

- Нехай молекули газу представлені частинками, що рухаються у двовимірному просторі згідно правил випадкового шляху.
- Є прямокутна область, поділена спочатку на дві прямокутних частини стінкою.
- Одна частина рівномірно заповнена молекулами, а інша, - порожня. Потім стінку знімають і молекули починають вільно рухатись по всій області.
- Змоделювати рух заданої кількості молекул впродовж заданої кількості кроків.
- Побудувати відео, яке показує результат (стан моделі) через кожні m кроків.

Приклад: моделювання поведінки молекул газу у прямокутній області (версія 2).2

- У версії 2 використовується той самий клас Drunkard2D, що й у версії 1.
- Відео будується зі збережених файлів зображень, які стають кадрами відео.
- Для побудови відео використовується програма mencoder – частина пакету mplayer, що вільно розповсюджується.
- Після завантаження з мережі цього пакету та розпакування архіву, треба вказати шлях до каталогу програми у змінній PATH.
- Запуск mencoder з програми у Python здійснюється за допомогою функції os.system зі стандартного модуля os.
- Ця функція виконує команду операційної системи, що міститься у рядку – параметрі.
- Після побудови відео зберігається у файлі anim.mpg, а всі файли зображень видаляються.
- Файл відео може бути переглянутий стандартним програвачем відео.

Приклад: моделювання поведінки молекул газу у прямокутній області (версія 3)

- Нехай молекули газу представлені частинками, що рухаються у двовимірному просторі згідно правил випадкового шляху. Є прямокутна область, поділена спочатку на дві прямокутних частини стінкою. Одна частина рівномірно заповнена молекулами, а інша, - порожня. Потім стінку знімають і молекули починають вільно рухатись по всій області. Змоделювати рух заданої кількості молекул впродовж заданої кількості кроків. Побудувати відео, яке показує результат (стан моделі) через кожні m кроків.
- Забезпечити програвання відео у уповільненому режимі.
- Версія 3 практично не відрізняється від версії 2.
- Але після побудови відео вводиться коефіцієнт уповільнення, а саме відео програється програмою mplayer.
- Після програвання відео файл anim.mpg видаляється.

Приклад: моделювання поведінки молекул газу у прямокутній області (версія 4)

- Нехай молекули газу представлені частинками, що рухаються у двовимірному просторі згідно правил випадкового шляху. Є прямокутна область, поділена спочатку на дві прямокутних частини стінкою. Одна частина рівномірно заповнена молекулами, а інша, - порожня. Потім стінку знімають і молекули починають вільно рухатись по всій області. Змоделювати рух заданої кількості молекул впродовж заданої кількості кроків.
- Побудувати анімацію у matplotlib, яка показує результат (стан моделі) через кожні m кроків.

Приклад: моделювання поведінки молекул газу у прямокутній області (версія 4).2

- Для побудови анімації використаємо ще один модуль matplotlib: animation.
- Цей модуль містить опис декількох класів для анімації.
- Ми використаємо клас FuncAnimation, який реалізує анімацію, виконуючи виклики функції для ініціалізації кадра та функції, яка повертає черговий кадр.
- У головній програмі це функції init та animate відповідно.
- Функція init викликається 1 раз та повертає ргафік, що буде використовуватись при кожній зміні кадру.
- Функція animate(i) викликається перед зображенням чергового кадру та повертає і-й кадр.
- Параметр frames встановлює кількість кадрів анімації, параметр interval визначає інтервал між кадрами у мілісекундах, параметр repeat визначає необхідність повторення анімації спочатку після її закінчення.

Резюме

- Ми розглянули:
  1. Наукові обчислення. Встановлення та імпорт numpy та matplotlib.
  2. Масиви numpy: створення масивів та типи елементів.
  3. Виконання операцій над масивами numpy. Векторизація
  4. Зображення результатів обчислень у графічному вигляді
  5. Вибір кольору та стилю ліній. Зображення декількох підграфіків на одному рисунку
  6. Багатовимірні масиви. Індексація багатовимірних масивів
  7. Вирізки індексів та вибір підмасивів. Індексні масиви
  8. Відношення та бульові операції для масивів numpy. Бульова індексація для масивів numpy
  9. Функції all, any та sum у numpy
  10. Векторно-матричні операції
  11. Поширення масивів. Зміна розмірності та/або розміру масивів
  12. Введення та виведення масивів numpy у текстовий файл
  13. Генерація випадкових чисел та їх використання
  14. Метод Монте-Карло. Інтегрування методом Монте-Карло. Обчислювання площ фігур методом Монте-Карло.
  15. Зображення гістограм
  16. Випадковий шлях у одновимірному та двовимірному просторі
  17. Побудова відео та анімації результатів обчислень

Де прочитати

1. Обвінцев О.В. Об’єктно-орієнтоване програмування. Курс на основі Python. Матеріали лекцій. – К., Основа, 2017
2. Langtangen H.P. - A Primer on Scientific Programming with Python, 2nd Edition – 2011
3. Beginning Python Visualization
4. John V Guttag - Introduction to Computation and Programming Using Python – 2013
5. Igor Milovanovic - Python Data Visualization Cookbook – 2013
6. http://matplotlib.org/contents.html
7. https://docs.scipy.org/doc/numpy/user/index.html
8. http://web.cs.wpi.edu/~cs1004/a14/Resources/Windows/SettingUpPython_Windows.docx
9. http://web.cs.wpi.edu/~cs1004/a14/Resources/Macintosh/SettingUpPython_Macintosh-Linux.pdf
10. http://www.python-course.eu/numerical_programming.php
11. http://www.labri.fr/perso/nrougier/teaching/numpy/numpy.html
12. http://cs231n.github.io/python-numpy-tutorial/#numpy-arrays
13. http://www.engr.ucsb.edu/~shell/che210d/numpy.pdf
14. https://www.youtube.com/watch?v=Aw98jGqjSMQ
15. http://scipy.github.io/old-wiki/pages/Tentative_NumPy_Tutorial.html#Universal_Functions
16. http://pythonworld.ru/numpy
17. https://en.wikipedia.org/wiki/Jacobi_method