Pierwszy Projekt Labowy

Metody Numeryczne 2024Z

`Termin oddania: do 23 grudnia 2024 włącznie`

– Ogry są... jak cebula!
– Śmierdzą?!
– Tak... nie!
– Bo się od nich płacze?
– Nie!
– Aaa. Jak się je zostawi na słońcu, to brązowieją i rosną im włoski?
– Nie! Warstwy! Cebula ma warstwy, ogry mają warstwy – cebula ma warstwy.
Dociera!? Ogry mają warstwy.

rozmowa Shreka i Osła.

Niech $n, m \in \mathbb{N}$. Powiemy, że macierz $\mathbf{A}$ o wymiarach $nm \times nm$ jest *cebulowa*, jeśli jest następującej postaci:

$$\mathbf{A} = \begin{bmatrix} A_1 & A_1 & A_1 & \dots & A_1 & A_1 \\ A_1 & A_2 & A_2 & \dots & A_2 & A_2 \\ A_1 & A_2 & A_3 & \dots & A_3 & A_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ A_1 & A_2 & A_3 & \dots & A_{n-1} & A_{n-1} \\ A_1 & A_2 & A_3 & \dots & A_{n-1} & A_n \end{bmatrix}, \qquad (🧅)$$

przy czym każda z macierzy A_i ($1 \leqslant i \leqslant n$) jest macierzą kwadratową $m \times m$. Precyzyjniej rzecz ujmując, podmacierz $\mathbf{A}$ utworzona przez wiersze $(k-1)m+1, (k-1)m+2, \dots, km$ i kolumny $(l-1)m+1, (l-1)m+2, \dots, lm$ (gdzie $1 \leqslant k, l \leqslant n$) to $A_{\min\{k,l\}}$.

Stwórz w Pythonie klasę `OnionMatrix`, która reprezentuje macierz cebulową i implementuje następujące metody:

- `__init__(self, blocks: list[numpy.ndarray]) -> OnionMatrix`
  konstruktor macierzy cebulowej $\mathbf{A}$, określonej wzorem (🧅). `blocks` to lista długości n zawierająca macierze A_i tych samych wymiarów, z których każda reprezentowana jest jako `numpy.ndarray`
- `OnionMatrix.multiply(self, v: numpy.ndarray) -> numpy.ndarray`
  metoda odpowiadająca za mnożenie $\mathbf{A}$ z prawej strony przez wektor v długości nm
- `OnionMatrix.solve(self, b: numpy.ndarray) -> numpy.ndarray`
  metoda odpowiadająca za rozwiązywanie równania $\mathbf{A}x = b$ (b jest wektorem długości nm)

W implementacji można korzystać wyłącznie z funkcji i klas wbudowanych w Pythona oraz tych dostępnych w pakietach `numpy`, `scipy` i `math` (w szczególności można korzystać z zaimplementowanych w tych pakietach funkcji rozwiązujących układy równań liniowych). W implementacji nie trzeba sprawdzać poprawności danych wejściowych (w szczególności w implementacji metody `solve` nie trzeba weryfikować, czy macierz $\mathbf{A}$ jest nieosobliwa). Oczywiście, wyżej opisane metody nie muszą być jedynymi implementowanymi przez przedstawioną klasę.

Sposób oceniania:

Za zadanie można otrzymać od 0 do 8 punktów. Ocena jest obliczana w następujący sposób:

- po pierwsze, sprawdzane jest, czy przy $m = n = 3$, losowym wypełnieniu macierzy i losowym wektorze wejściowym, zastosowanie metod daje w wyniku to, czego się od nich oczekuje. Jeśli tak jest, rozwiązanie otrzymuje co najmniej 1 punkt, w przeciwnym wypadku na pewno otrzymuje 0 punktów.
- po drugie, sprawdzana jest szybkość działania tych metod przy $n = 1000$, $m = 10$, losowym wypełnieniu macierzy oraz losowych wektorach wejściowych. Testy będą przeprowadzane na maszynie `students`.

- Jeśli dana metoda działa w czasie mniejszym niż 0,01s, do oceny dodaje się 3 punkty dla metody `multiply` i 4 punkty dla metody `solve`.
- Jeśli dana metoda działa w czasie nie mniejszym niż 0,01s, ale mniejszym niż 0,05s, do oceny dodaje się 1 punkt dla metody `multiply` i 2 punkty dla metody `solve`.
- Jeśli dana metoda działa w czasie nie mniejszym niż 0,05s, ocena nie zostaje zwiększona.

Do porównania brana jest wartość maksymalna z 10 prób; uwzględniamy również czas potrzebny na skonstruowanie obiektu klasy `OnionMatrix`. W ramach tego punktu będzie również sprawdzana poprawność wyników, i jeśli będą one błędne, punkty (za metodę dającą błędne wyniki) nie będą przyznawane.

(W testach metoda benchmarkowa dawała $\sim$0,005s dla `multiply` oraz $\sim$0,007s dla `solve`.)

Kod służący do oceniania umieszczony jest w pliku `proj2_ocena.py`. W ramach `test_solve_time` oraz `test_multiply_time` następować będzie również sprawdzanie wyników poprzez porównanie z metodą benchmarkową.

Format rozwiązania:

Plik `nazwisko_imie-OnionMatrix.py` zawierający implementację opisanej w zadaniu klasy.