Pierwszy Projekt Labowy

Metody Numeryczne 2022/2023, grupy 1 i 7

`Termin oddania: do 8 listopada 2022 włącznie`

Niech $d, n \in \mathbb{N}$ oraz $1 \leqslant d < n$. Powiemy, że macierz kwadratowa $A = [a_{ij}]_{i,j \leqslant n}$ jest *macierzą d-trójdiagonalną*, jeśli $a_{ij} = 0$ dla $|i - j| \notin \{0, d\}$. Poniżej znajduje się przykład macierzy 2-trójdiagonalnej.

$$\begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 3 & 0 & 4 \\ 5 & 0 & 6 & 0 \\ 0 & \pi & 0 & 7 \end{bmatrix}.$$

Naddiagonalą macierzy d-trójdiagonalnej $A = [a_{ij}]_{i,j \leqslant n}$ jest ciąg $(a_{i,i+d})_{1 \leqslant i \leqslant n-d}$, a *poddiagonalą* ciąg $(a_{i+d,i})_{1 \leqslant i \leqslant n-d}$; standardowo, *diagonala* to ciąg $(a_{i,i})_{1 \leqslant i \leqslant n}$. W powyższym przykładzie naddiagonala to ciąg $(2, 4)$, a poddiagonala to ciąg $(5, \pi)$; oczywiście, diagonalą jest ciąg $(1, 3, 6, 7)$.

Stwórz w Pythonie klasę `dTridiag`, która reprezentuje macierz d-trójdiagonalną i implementuje następujące metody:

- `__init__(self, a:np.ndarray, b:np.ndarray, c:np.ndarray) -> dTridiag`
  konstruktor d-trójdiagonalnej macierzy A, której diagonalę stanowi wektor `a`, naddiagonala to wektor `b`, a poddiagonala to wektor `c`. Wszystkie trzy wektory reprezentowane są przez `np.ndarray` (tu i później `np` jest skrótem do `numpy`).
- `dTridiag.dot(self, v: np.ndarray) -> np.ndarray`
  metoda odpowiadająca za mnożenie A z prawej strony przez zadany wektor `v`
- `dTridiag.solve(self, y: np.ndarray) -> np.ndarray`
  metoda odpowiadająca za rozwiązywanie równania $Ax =$ `y`, gdzie `y` jest zadanym wektorem

Powyższe metody **nie** muszą sprawdzać poprawności argumentów (np. tego, że `np.size(a)>=np.size(b)==np.size(c)` w konstruktorze lub zgodności wymiarów w metodach `dot` i `solve`).
[Uzupełnienie 24 X] Zakładamy ponadto, że skądinąd wiadomo iż macierze A, z którymi mamy do czynienia, są nieosobliwe i mają rozkład LU.

W implementacji można korzystać wyłącznie z funkcji i klas wbudowanych w Pythona oraz tych dostępnych w pakiecie `numpy` (oraz, oczywiście, własnych). Rzecz jasna, wyżej opisane metody nie muszą być jedynymi implementowanymi przez przedstawioną klasę.

Wymagania pamięciowe i obliczeniowe

Obiekt klasy `dTridiag` powinien zajmować $O(n)$ pamięci.
Każda udostępniana przez niego metoda powinna wykorzystywać $O(n)$ pamięci oraz $O(n)$ operacji zmiennoprzecinkowych.

Sposób oceniania:

Za zadanie można otrzymać maksymalnie 5 punktów. Zaczynając od `ocena=5`...

1. ...jeśli metoda `dTridiag.dot` nie prowadzi do poprawnych wyników (co weryfikowane będzie z wykorzystaniem losowych wektorów `a`, `b`, `c`, `v`), `ocena=ocena-3`.
2. ...jeśli metoda `dTridiag.solve` nie prowadzi do poprawnych wyników (co weryfikowane będzie z wykorzystaniem losowych wektorów `a`, `b`, `c`, `y`), `ocena=ocena-3`.
3. ...za każde przekroczenie opisanych wyżej wymagań pamięciowo-obliczeniowych, `ocena=ocena-1.5`.
4. `ocena = max(ocena, 0)`

$\boxed{5.}$ …jeśli gdziekolwiek w kodzie pojawia się `np.linalg.inv`, `ocena=-10`.

Dodatkowo, sprawdzający zastrzega sobie możliwość co-najwyżej-dwukrotnego wykorzystania instrukcji `ocena=ocena-0.5` (z powodów, które będzie musiał przedstawić w komentarzu do rozwiązania) oraz dowolnie-krotnego zastosowania instrukcji `ocena=ocena+0.5` (jeśli na przykład coś w kodzie szczególnie mu się spodoba).

Format rozwiązania:

Plik o nazwie odpowiadającej szablonowi `nazwisko_imie-dTridiag.py`, zawierający implementację klasy `dTridiag`. Ponadto w pliku `nazwisko_imie-dTridiag-komentarz.pdf` należy przedstawić (choćby kilkuzdaniowy) komentarz do rozwiązania, uzasadniający poprawność implementacji oraz spełnianie wymagań pamięciowo-obliczeniowych.