

Algorytmy macierzowe - zadanie nr 3 - Formaty macierzy rzadkich

" Proszę wybrać swój ulubiony język programowania. Proszę wziąć największe macierze, które używali Państwo w zadaniu pierwszym (ewentualnie zmniejszyć p żeby stały się rzadsze). Proszę przekonwertować je do stosownego formatu i przemnożyć przez siebie.

6. Pierwsza i druga macierz CSR format "

Marcin Hawryluk, Norbert Wolniak
grupa: piątek 12:50B

```
In [1]: import numpy as np
from time import time
import pandas as pd
import matplotlib.pyplot as plt
import os
from time import time
```

Generowanie macierzy

Macierze użyte w ćwiczeniu są typu FEM i zostały przez nas wygenerowane za pomocą funkcji massmatrix w środowisku Octave. Największa z macierzy ma wymiary 441x441.

```
In [2]: def read_matrix(file_name):
    with open(file_name, 'r') as file:
        for line in file:
            if line.strip() == '':
                continue
            if line[0] == '#':
                if line[2:6] == "rows":
                    _, _, size = line.split()
                    size = int(size)
                    matrix = np.zeros((size, size))
            else:
                row, col, val = line.split(' ')
                matrix[int(row)-1, int(col)-1] = val

    return matrix
```

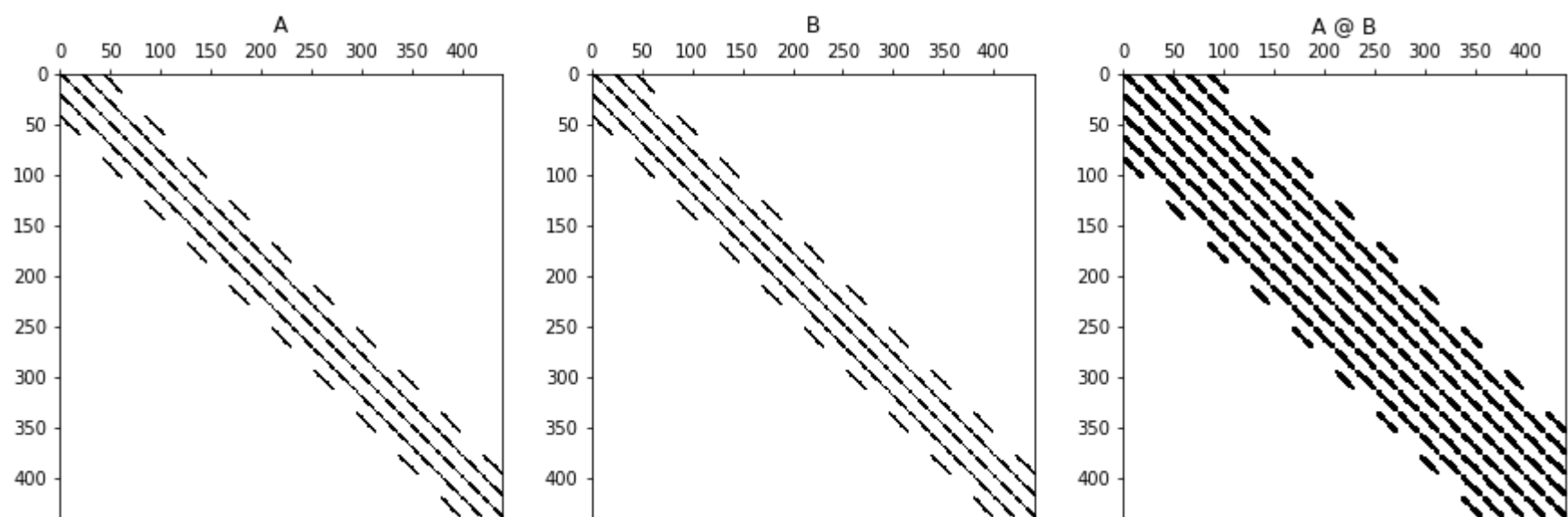
```
In [3]: # 441 x 441 matrix
A = read_matrix("matrices/fem_10.txt")
B = A.copy()
```

```
In [4]: fig, ax = plt.subplots(1, 3, figsize=(15, 5))

ax[0].spy(A)
ax[1].spy(B)
ax[2].spy(A @ B)

ax[0].set_title('A')
ax[1].set_title('B')
ax[2].set_title('A @ B')

plt.show()
```



Jak możemy zauważyć z powyższej wizualizacji, mnożone macierze są rzadkie. Wynik mnożenia zawiera już więcej niezerowych wartości, chociaż nadal jest to macierz rzadka.

```
In [5]: matrices = []
for file in os.listdir('matrices'):
    matrices.append(read_matrix('matrices/' + file))
```

Mnożenie macierzy z zadania 1

Za pomocą procedur napisanych w ramach ćwiczenia nr 1, dokonujemy pomiaru czasu mnożenia w sytuacji, w której nie wykorzystujemy faktu, że macierze są rzadkie.

```
In [6]: def mm_ipj(matrix_a, matrix_b):
    (m, n), k = matrix_a.shape, matrix_b.shape[1]
    matrix_c = np.zeros((m, k), dtype=np.float64)
    for i in range(m):
        for p in range(n):
            for j in range(k):
                matrix_c[i, j] += matrix_a[i, p] * matrix_b[p, j]
    return matrix_c
```

```
In [7]: start = time()
mm_ipj(A, B)
end = time()
print("Czas mnożenia:", end - start, "s")
```

Czas mnożenia: 39.08433198928833 s

Format CSR

Poniższa funkcja służy do konwersji zadanej macierzy do formatu Compressed Sparse Row.

```
In [8]: def convert_to_csr(matrix):
    m, n = matrix.shape
    ICL = []
    VAL = []
    ROWPTR = []
    counter = 0

    for i in range(n): # rows
        ROWPTR.append(counter)
        for j in range(m): # columns
            val_ij = matrix[i, j]
            if abs(val_ij) < 1e-8:
                continue
            ICL.append(j)
            VAL.append(val_ij)
            counter += 1

    ROWPTR.append(counter)

    return ICL, VAL, ROWPTR
```

```
In [9]: A_csr = convert_to_csr(A)
B_csr = convert_to_csr(B)
```

Mnożenie macierzy CSR

Poniższa procedura matmul_CSR służy do mnożenia dwóch macierzy przekazanych w formacie CSR. Zwracana macierz C również jest w tej postaci.

Z uwagi na specyfikę formatu CSR, w którym dużo łatwiej iterować najpierw po wierszach, ważna była odpowiednia kolejność odwołań do wartości. Przechodzimy więc zatem po kolejnych wierszach macierzy A i dla każdego pola z tego wiersza znajdujemy odpowiedni wiersz w macierzy B. W celu osiągnięcia formatu CSR wyniku - bez konwersji wynikowej macierzy gęstej - korzystamy ze struktury słownika (tablicy z haszowaniem).

```
In [10]: def get_ICL_VAL_from_dict(C_ROW):
    C_ROW_sorted = sorted(C_ROW.items(), key=lambda item : item[0]) # sorting by key (column index)
    ICL_C = [x[0] for x in C_ROW_sorted] # column indexes list
    VAL_C = [x[1] for x in C_ROW_sorted] # values list
    return ICL_C, VAL_C

def matmul_CSR(A, B):
    ICL_A, VAL_A, ROWPTR_A = A
    ICL_B, VAL_B, ROWPTR_B = B
    ICL_C, VAL_C, ROWPTR_C = [], [], []

    elements_counter = 0

    for i in range(len(ROWPTR_A) - 1): # rows in A
        row_a_start_idx = ROWPTR_A[i]
        row_a_end_idx = ROWPTR_A[i+1]
        C_ROW = dict() # dict to store results of row_A * rows_B

        for j in range(row_a_start_idx, row_a_end_idx): # indexes of values in row A
            col_a = ICL_A[j]
            val_a = VAL_A[j]
            row_b_start_idx = ROWPTR_B[col_a] # column idx in A maps to row idx in B
            row_b_end_idx = ROWPTR_B[col_a + 1] # col_a + 1 to get range of value indexes in B row

            for p in range(row_b_start_idx, row_b_end_idx): # indexes of values in row B
                col_b = ICL_B[p]
                val_b = VAL_B[p]
                C_ROW[col_b] = C_ROW.get(col_b, 0) + (val_a * val_b) # update dict with val_a * val_b

        ICL, VAL = get_ICL_VAL_from_dict(C_ROW)
        ICL_C += ICL
        VAL_C += VAL
        ROWPTR_C.append(elements_counter)
        elements_counter += len(VAL)

    ROWPTR_C.append(elements_counter)
    return ICL_C, VAL_C, ROWPTR_C
```

```
In [11]: start = time()
matmul_CSR(A_csr, B_csr)
end = time()
print("Czas mnożenia:", end - start, "s")
```

Czas mnożenia: 0.03740406036376953 s

Test poprawności mnożenia CSR * CSR

```
In [12]: def print_CSR_matrix(A):
    ICL, VAL, ROWPTR = A
    VAL = VAL.copy()
    m_col = max(ICL) + 1
    for i in range(len(ROWPTR)-1):
        for p in range(m_col):
            if p in ICL[ROWPTR[i]:ROWPTR[i+1]]:
                print(VAL.pop(0), end="\t")
            else:
                print(0, end="\t")
    print()
```

```
In [13]: testA = np.array([[0,0,1],
                           [0,0,0],
                           [3,1,4]])
testB = np.array([[0,0,1],
                  [0,0,0],
                  [3,1,4]])

testA_csr = convert_to_csr(testA)
testB_csr = convert_to_csr(testB)

print("A w formacie CSR\n ICL : {} \n VAL : {}\n ROWPTR : {}".format(*testA))
```

A w formacie CSR
 ICL : [0 0 1]
 VAL : [0 0 0]
 ROWPTR : [3 1 4]

```
In [14]: print("Wynikowa macierz (zwykle mnożenie)\n")
print(testA @ testB)
```

Wynikowa macierz (zwykle mnożenie)

```
[[ 3  1  4]
 [ 0  0  0]
 [12  4 19]]
```

```
In [15]: testC = matmul_CSR(testA_csr, testB_csr)
print("Wynikowa macierz (mnożenie CSR)\n")
print_CSR_matrix(testC)
```

Wynikowa macierz (mnożenie CSR)

```
3      1      4
0      0      0
12     4     19
```

Pomiar czasów

```
In [16]: def compare_times(matrices):
times = {}

for matrix in matrices:
    size = matrix.shape[0]
    times[size] = {}
    start = time()
    mm_ipj(matrix, matrix)
    times[size]["matmul_ipj"] = time() - start

    start = time()
    matrix_csr = convert_to_csr(matrix)
    times[size]["conversion time"] = time() - start

    start = time()
    matmul_CSR(matrix_csr, matrix_csr)
    times[size]["matmul_csr"] = time() - start

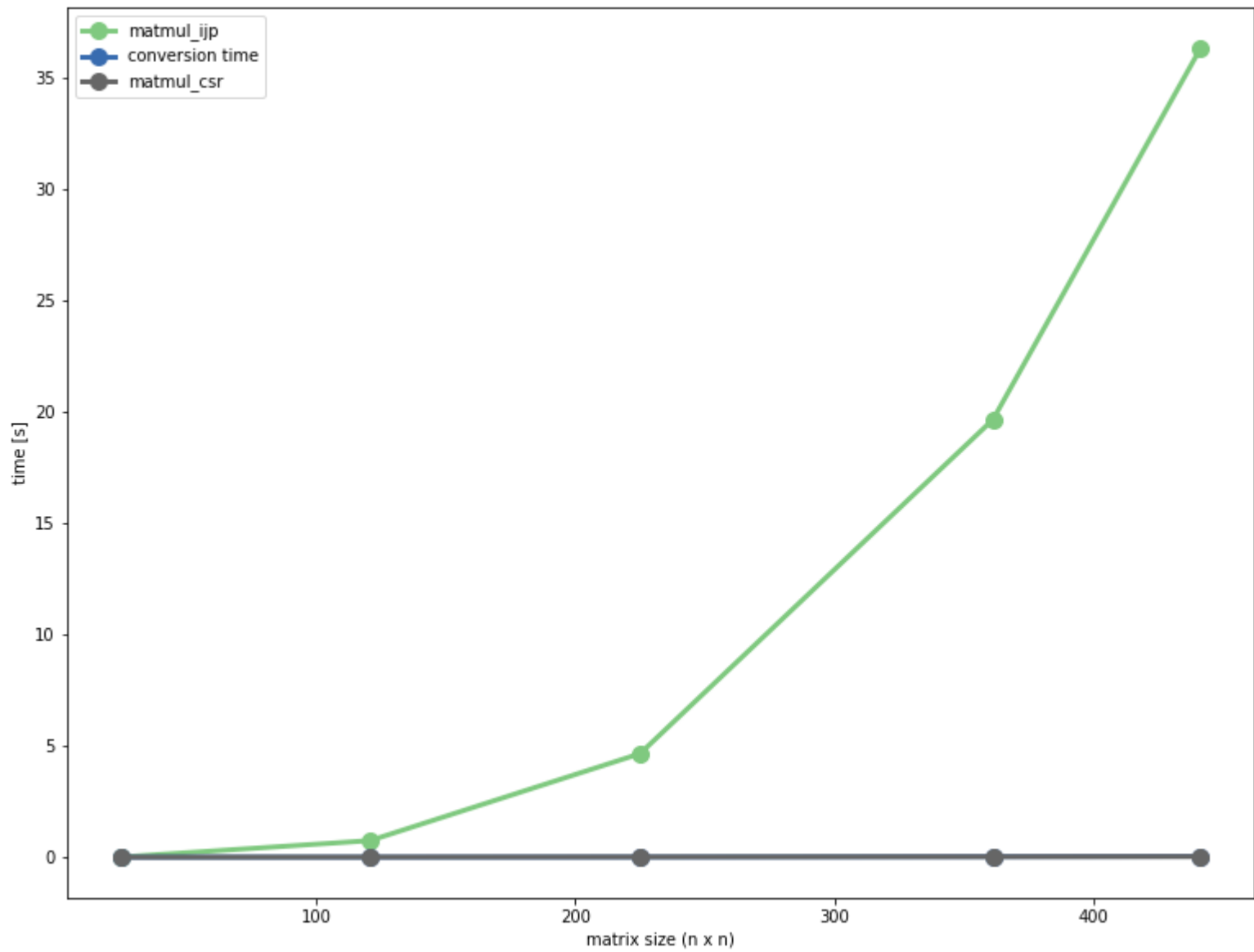
df = pd.DataFrame(times).T.sort_index()
df.plot(
    figsize=(13, 10),
    xlabel='matrix size (n x n)',
    ylabel='time [s]',
    colormap='Accent',
    linewidth=3,
    markersize=10,
    marker='o'
)

return df
```

```
In [17]: compare_times(matrices)
```

Out[17]:

	matmul_ijk	conversion time	matmul_csr
25	0.006938	0.000238	0.000848
121	0.742967	0.003431	0.006857
225	4.646366	0.011025	0.013627
361	19.636052	0.028084	0.022808
441	36.302584	0.042678	0.029824



Jak możemy zauważyć, czas mnożenia dla macierzy w formacie CSR jest wyraźnie mniejszy od czasu mnożenia dla macierzy gęstych. Dzięki specjalnemu formatowi i odpowiedniej procedurze byliśmy w stanie wyeliminować niepotrzebne operacje mnożenia i dodawania zer. Wymagało to trochę bardziej skomplikowanej procedury oraz dodatkowej konwersji, która wymaga tylko pojedynczego przejścia po macierzy. Stosowanie formatu CSR będzie dawać tym lepsze rezultaty, im stosunkowo mniej niezerowych wartości znajduje się w macierzy.

Wnioski

- W przypadku gdy większość elementów macierzy jest zerowa, jesteśmy w stanie zaoszczędzić pamięć, korzystając z różnych formatów reprezentacji macierzy rzadkich; jak również zmniejszyć czas operacji, takich jak np. mnożenie macierzy.
- Dla formatu CSR mamy możliwość szybkiego iterowania jedynie po wierszach. Odczyt po kolumnach wymaga dużej złożoności. W celu zoptymalizowania konkretnych procedur konieczne jest odpowiednie dobranie kolejności odwołań do wartości w macierzy.

M. Hawryluk, N. Wolniak. 2021