lab3

January 6, 2022

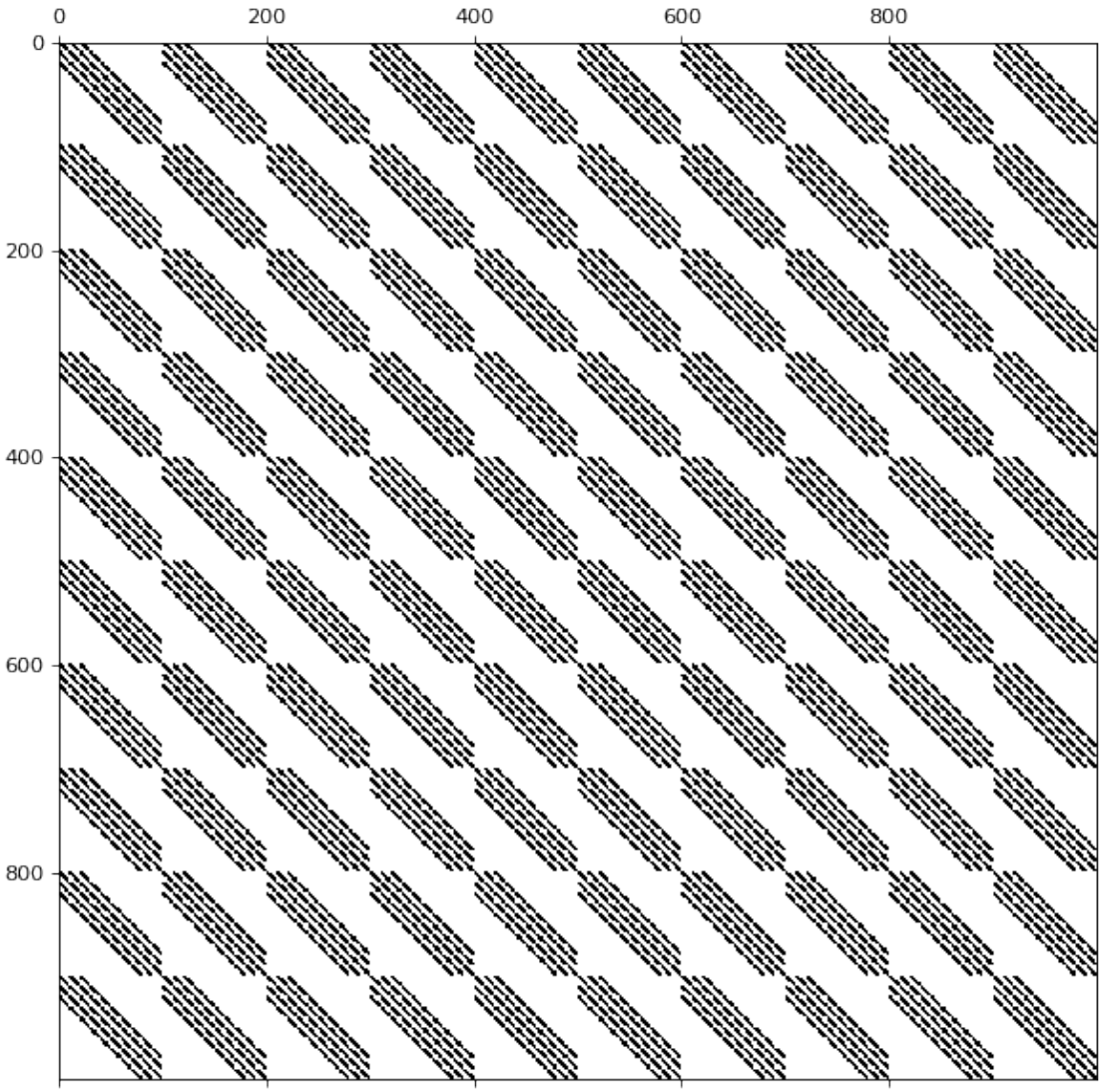
1 Laboratorium 3

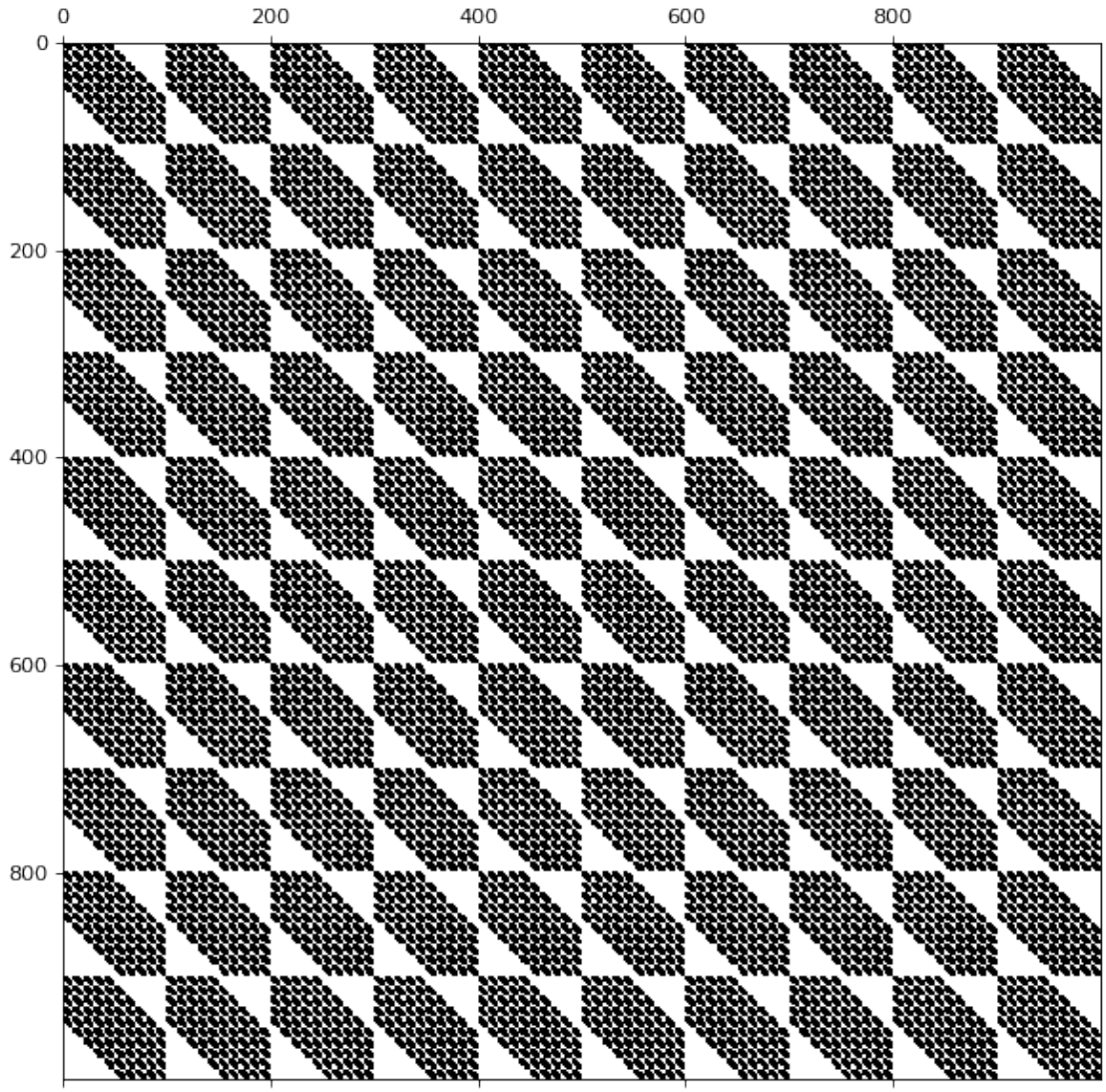
Wojciech Kłyszejko i Jan Stobnicki

Zadanie 6 - Mnożenie macierzy: CSR format

[8]:
```
import numpy as np
from scipy.sparse import csr_matrix
from time import time

def load_matrix(file_path):
    with open(file_path, "r") as file:
        return np.loadtxt(file, delimiter=",")

def enlarge_matrix(m, q):
    size = len(m) * q
    res = np.zeros([size, size])
    for i in range(size):
        for j in range(size):
            res[i, j] = m[i % len(m), j % len(m)]
    return res

def draw_nonzero_values(m):
    figure(figsize=(15, 9), dpi=80)
    plt.spy(m)
    plt.show()
```

[7]:
```
M1 = enlarge_matrix(load_matrix("matrices/iga8_2.csv"), 10)
M2 = enlarge_matrix(load_matrix("matrices/iga8_2.csv"), 10)

M3 = M1@M2

draw_nonzero_values(M2)
draw_nonzero_values(M3)
```


0
200
400
600
800
0
200
400
600
800



Algorytm mnożenia blokowego z zadania 1. Rozmiar bloku: 50 (najlepsze wyniki czasowe)

[17]:
```
def ijp_multiplication(m1, m2):
    m, n, k = len(m1), len(m2), len(m1[0])
    res = np.zeros([m, n])
    for i in range(m):
        for j in range(n):
            for p in range(k):
                res[i, j] += m1[i, p] * m2[p, j]
    return res

def block_matrix_multiplication(m1, m2, m_block_s, n_block_s, k_block_s):
    m, n, k = len(m1), len(m2), len(m1[0])
```

```
    res = np.zeros([m, n])
    for i in range(0, m, m_block_s):
        for j in range(0, n, n_block_s):
            for p in range(0, k, k_block_s):
                A_block = m1[i:i + m_block_s, p:p + k_block_s]
                B_block = m2[p:p + k_block_s, j:j + n_block_s]
                res[i:i + m_block_s, j:j + n_block_s] =␣
↪ijp_multiplication(A_block, B_block)
    return res
```

Przekształcenie macierzy do formatu CSR oraz mnożenie:

[29]:
```
class MyCSR:
    def __init__(self, indices, values, row_ptr):
        self.indices = indices
        self.values = values
        self.row_ptr = row_ptr

class CSRutil:
    @staticmethod
    def matrix_to_csr_format(A):
        nonzero_row, nonzero_col = A.nonzero()

        indices = nonzero_col
        values = A[nonzero_row, nonzero_col]

        tmp = [0 for i in range(np.shape(A)[0] + 1)]
        for e in nonzero_row:
            tmp[e + 1] += 1
        for i in range(1, len(tmp)):
            tmp[i] += tmp[i - 1]

        row_ptr = np.array(tmp)

        return MyCSR(indices, values, row_ptr)

    @staticmethod
    def mul_CSR_CSR(A,B):
        N = np.shape(A.row_ptr)[0] - 1
        C = np.zeros(shape= (N,N))

        for i in range(N):
            a_ind = [k for k in range(A.row_ptr[i], A.row_ptr[i+1])]
            a_cols = [A.indices[x] for x in a_ind]
            a_vals = [A.values[x] for x in a_ind]
            for a_col_ind in range(len(a_cols)):
                j = a_cols[a_col_ind]
```

```
                a_i_j = a_vals[a_col_ind]

                b_ind = [k for k in range(B.row_ptr[j], B.row_ptr[j+1])]
                b_cols = [B.indices[x] for x in b_ind]
                b_vals = [B.values[x] for x in b_ind]

                for b_col_ind in range(len(b_cols)):
                    C[i, b_cols[b_col_ind]] += a_i_j * b_vals[b_col_ind]

        return C
```

Porównanie czasu wykonywania obu algorytmów mnożenia:

[30]:
```
def timetest(m1, m2, mul_type, block_size=None):
    start_time = time()
    C = None
    if mul_type == "block":
        C = block_matrix_multiplication(m1, m2, block_size, block_size,␣
 ↪block_size)
    elif mul_type == "numpy":
        C = m1 @ m2
    else:
        # csr_multiplication
        m1_csr = CSRutil.matrix_to_csr_format(m1)
        m2_csr = CSRutil.matrix_to_csr_format(m2)
        C = CSRutil.mul_CSR_CSR(m1_csr, m2_csr)
    res = time() - start_time
    return res


def get_average_mul_time(m1, m2, mul_type, trial_number, block_size=None):
    total_time = 0
    for i in range(trial_number):
        test_i = timetest(m1, m2, mul_type, block_size)
        print(test_i)
        total_time += test_i
    res = total_time / trial_number
    print("Avg time for "+ str(mul_type) + " is: " + str(res) + " seconds")
    return res
```

[19]:
```
get_average_mul_time(M1, M2, "block", 3, 50)
```

```
421.89492106437683
405.7970869541168
405.24875688552856
Avg time for block is: 410.9802549680074 seconds
```

[19]:
```
410.9802549680074
```

[25]:
```
get_average_mul_time(M1, M2, "numpy", 3)
```

```
0.050360679626464844
0.0419156551361084
0.058332204818725586
Avg time for numpy is: 0.05020284652709961 seconds
```

[25]: 0.05020284652709961

[31]:
```
get_average_mul_time(M1, M2, "csr", 3)
```

```
20.60311985015869
21.174890995025635
20.159261465072632
Avg time for csr is: 20.64575743675232 seconds
```

[31]: 20.64575743675232

[ ]: