Dopełnienie Schura

December 6, 2021

1 Dopełnienie Schur'a z użyciem eliminacji Gaussa

Maciej Skoczeń, Kacper Kafara

grupa wtorek (A) 17:50

1.1 Środowisko obliczeniowe

Procesor: Intel i7-9750H @ 2,6 GHz; 6 rdzeni fizycznych (12 log.)

1.2 Importy & typy

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import os
import re
import subprocess
import matplotlib.pyplot as plt

from timeit import default_timer
from pprint import pprint
from math import sqrt

Array = np.ndarray
```

1.3 Funkcje pomocnicze

[2]:
```
class Timer(object):
    def __init__(self):
        self._start_time = None
        self._stop_time = None

    def start(self):
        self._start_time = default_timer()

    def stop(self):
        self._stop_time = default_timer()
    @property
    def elapsed(self, val = None):
        if self._stop_time is None or self._start_time is None:
            return None
        elapsed = self._stop_time - self._start_time
        return elapsed

# mock impl
def is_int(value) -> bool:
    as_int = int(value)
    return value == as_int
```

1.3.1 Wczytywanie macierzy

wygenerowanej za pomocą dostarczonego skryptu `mass_matrix`

[3]:
```
def input_matrix(octave_matrix, n, m, q=1):
    result = np.zeros((n*q, m*q), dtype=np.double)

    for elem in octave_matrix:
        m = re.match(r"\s*\((\d+),\s*(\d+)\)\s*->\s*(\d+\.\d+)\s*", elem)
        if m is not None:
            x, y, value = m.groups()
        elif len(elem) > 0:
            coord, value = elem.strip().split(' -> ')
            value = float(value)
            x, y = coord.split(',')
            x, y = x[1:], y.strip()[:-1]
        else:
            continue

        for i in range(q):
            for j in range(q):
                result[i*n + int(x) - 1, j*n + int(y) - 1] = float(value)

    return result
```

[4]:
```
def load_octave_matrix(filename):
    with open(filename, "r") as file:
        return file.readlines()
```

[5]:
```
data_dir = "../../output"

def resolve_path(matrix_type, width, height = None, generate = False):
    if height is None: height = width
    path = f"{data_dir}/{matrix_type}-{width}x{height}.txt"
    if os.path.isfile(path): return path
    else:
        if not generate:
            raise FileNotFoundError(f"Matrix file {path} not found")

        # do generowania macierzy potrzebny jest direnv, ustawiona zmienna
        # środowiskowa:
        # SCRIPT_DIR=<path-to-scripts-dir>
        # albo na sztywno ustawiona ścieżka do skryptu (ale wtedy trzeba␣
 ↪zmodyfikować)
        # funkcję generate_matrix

        if width != height:
            raise ValueError("Can only generate square matrix")

        generate_matrix(matrix_type, width)

        if os.path.isfile(path): return path
        else:
            print(path)
            raise RuntimeError("Failed to generate matrix")


resolve_matrix = lambda matrix_type, n, m, q = 1: input_matrix(
    load_octave_matrix(resolve_path(matrix_type, n, m)), n, m, q
)

def resolve_matrix(matrix_type, n, m, q = 1, generate = False):
    return input_matrix(
        load_octave_matrix(resolve_path(matrix_type, n, m, generate =␣
 ↪generate)), n, m, q
    )

def generate_matrix(matrix_type, rank):
    if matrix_type not in {'iga', 'fem'}:
        raise ValueError(f"Invalid matrix type: {matrix_type}")

    if rank < 16 or not is_int(sqrt(rank)):
        raise ValueError(f"Invalid matrix rank: {rank}. Must be >= 16 and␣
 ↪sqrt(rank) must be of type integer.")

    rank_root = int(sqrt(rank))

    if matrix_type == 'fem':
        for p in range(2, 5):
            double_nxx = rank_root - p + 1
            if double_nxx % 2 == 0 and double_nxx // 2 >= 2:
                nxx = double_nxx // 2
                pxx = p
                break
        else:
            raise RuntimeError(f"Failed to determine nxx, pxx for rank: {rank}")
    else:
        for p in range(2, 5):
            nxx = rank_root - p
            if nxx >= 2:
                pxx = p
                break
        else:
            raise RuntimeError(f"Failed to determine nxx, pxx for rank: {rank}")

    cwd = os.getcwd()
    scripts_dir = os.getenv('SCRIPTS_DIR')
    os.chdir(scripts_dir)
    !./generate-matrix.sh cpp {matrix_type} {nxx} {pxx} 0
    os.chdir(cwd)
```

1.4 Eliminacja Gaussa

[6]:
```
def transform_matrix_gaussian_elim(
    A: Array,
    rows_to_transform: int,
    in_place: bool = False,
    timer: Timer = None
) -> Array:

    if not in_place: A = A.copy()

    if timer is not None:
        timer.start()

    n, _ = A.shape
    for i in range(0, min(n - 2, rows_to_transform)):
        A_i_i = A[i, i]
        for j in range(i + 1, n):
            factor = A[j, i] / A_i_i
            A[j, i] = 0
            for k in range(i + 1, n):
                A[j, k] -= factor * A[i, k]

    if timer is not None: timer.stop()
    if not in_place: return A
```

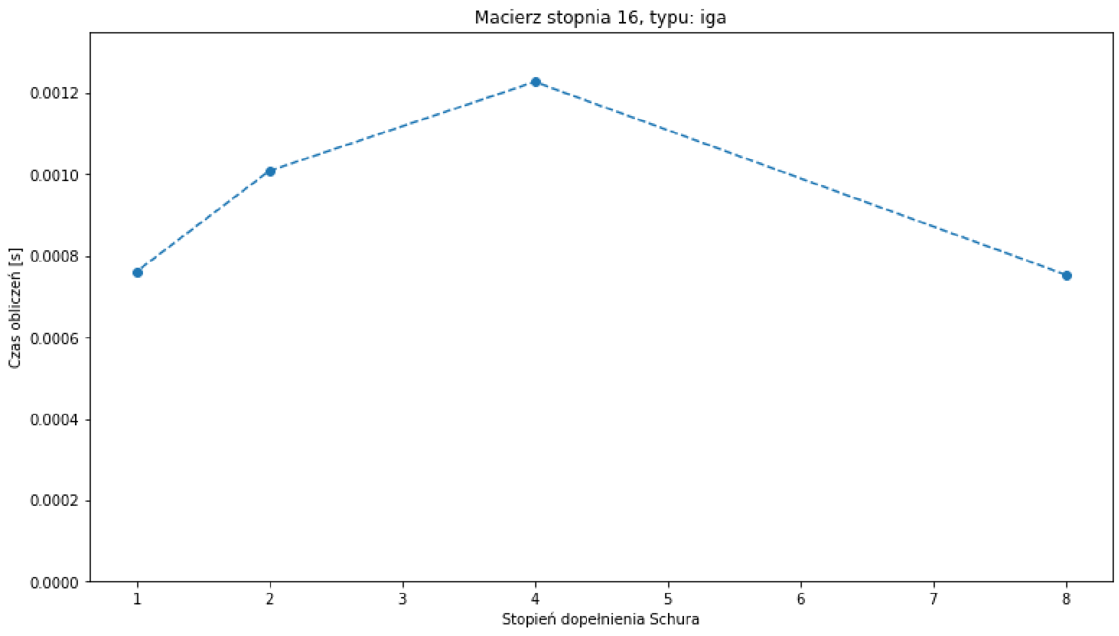
1.5 Dopełnienie Schur'a

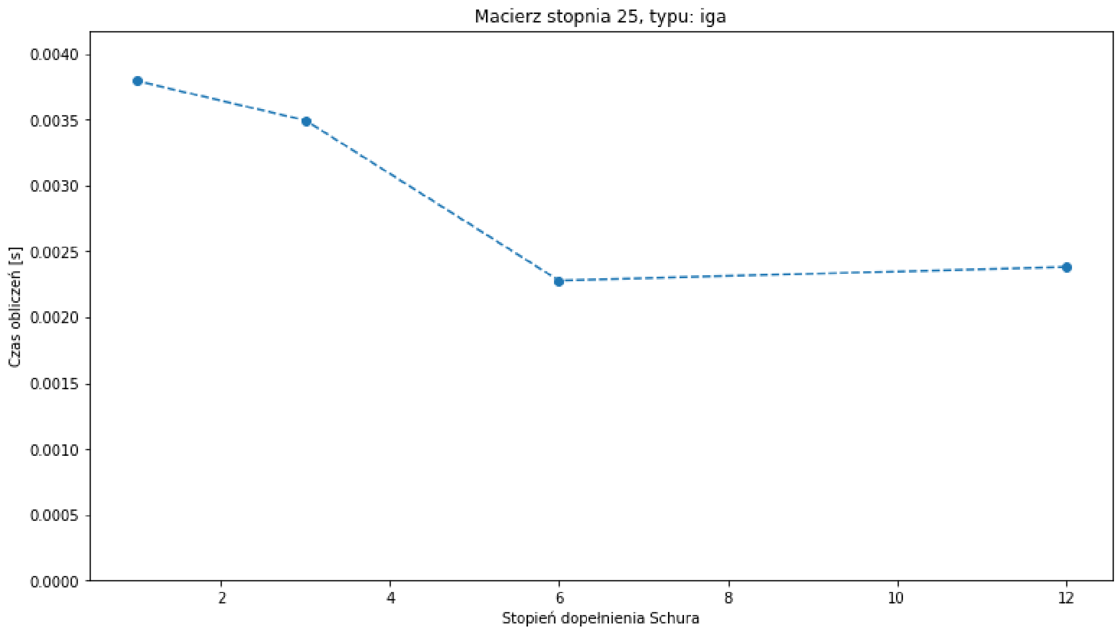
[7]:
```
def schur_complement(A: Array, complement_degree: int, timer: Timer = None) ->␣
↪Array:
    transformed = transform_matrix_gaussian_elim(A,
                                                 A.shape[0] - complement_degree,
                                                 in_place = False,
                                                 timer = timer)
    return transformed[A.shape[0] - complement_degree :, A.shape[1] -␣
↪complement_degree :]
```

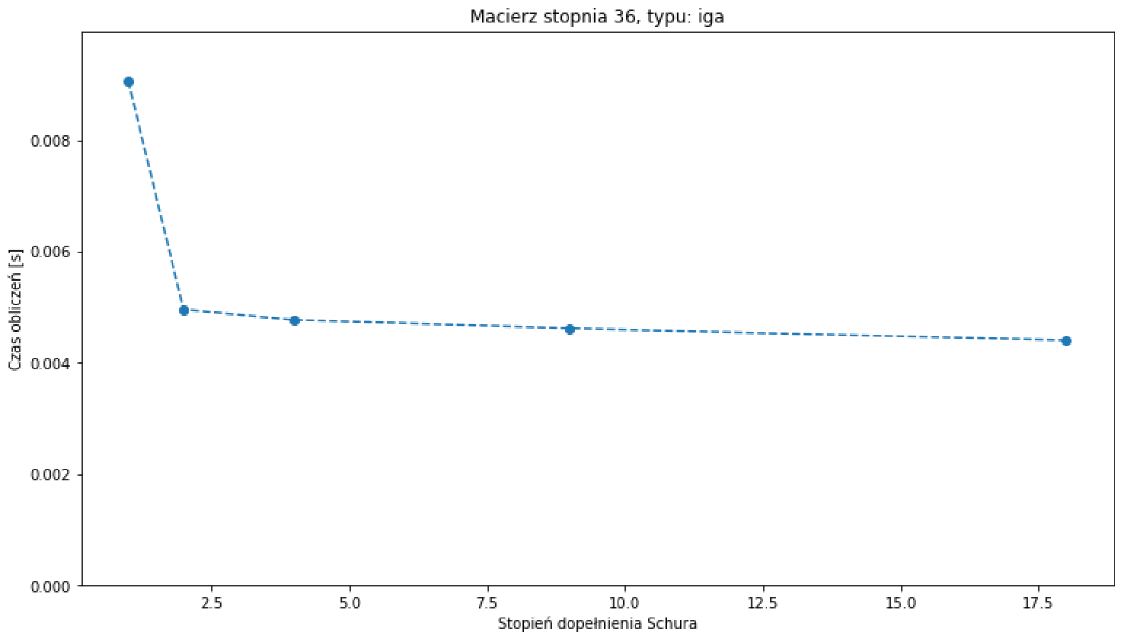
[ ]:
```
nxxs = {}
nxxs['iga'] = [i for i in range(2, 31)]
nxxs['fem'] = [i for i in range(2, 17)]

pxx = 2
rxx = 0
ranks = {}
ranks['iga'] = [(nxx + pxx) ** 2 for nxx in nxxs['iga']]
ranks['fem'] = [(2 * nxx + pxx - 1) ** 2 for nxx in nxxs['fem']]

matrixtypes = 'iga', 'fem'

main_timer = Timer()
exec_times = {
    'iga': {},
    'fem': {}
}
exec_ranks = {
    'iga': {},
    'fem': {},
}

padding = lambda n: n * ' '

for matrix_t in matrixtypes:
    matrices = ((resolve_matrix(matrix_t, rank, rank, generate=True), rank) for␣
↪rank in ranks[matrix_t])
    print('Computations for matrix type: ', matrix_t)
    for M, rank in matrices:
        print(padding(2) + 'Computations for rank', rank)

        exec_times[matrix_t][rank] = []
        exec_ranks[matrix_t][rank] = []
        rank_cp = rank

        while rank_cp >= 2:
            rank_cp //= 2
            print(padding(4) + 'Current rank:', rank_cp, end = '   ')

            schur_complement(M, rank_cp, timer = main_timer)

            exec_times[matrix_t][rank].append(main_timer.elapsed)
            exec_ranks[matrix_t][rank].append(rank_cp)

            print(f'{main_timer.elapsed:.5f}s')
```

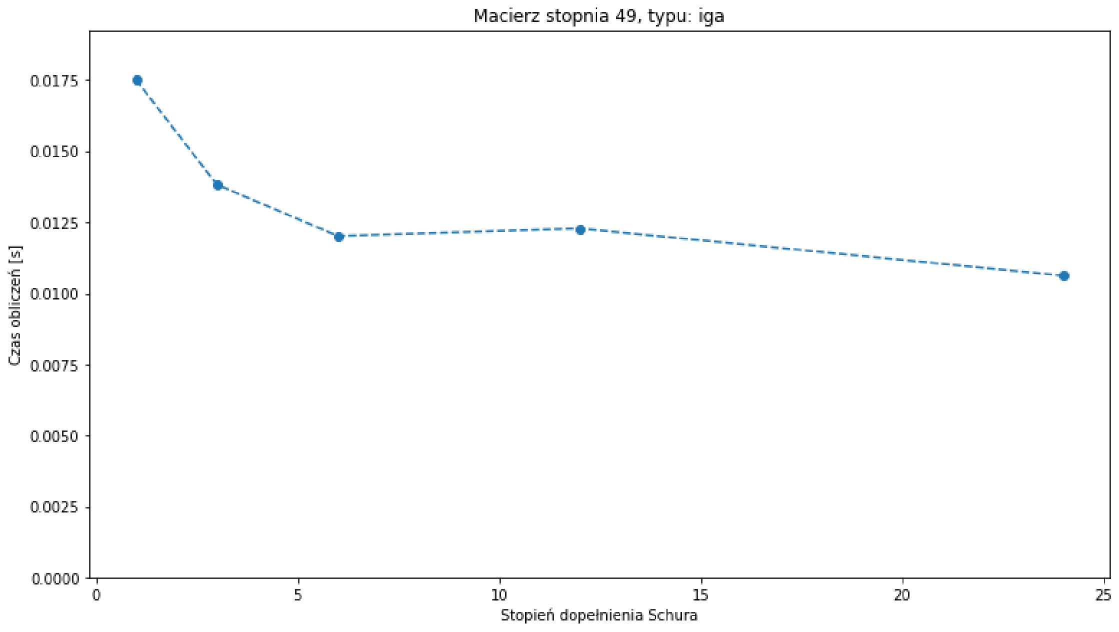
[23]:
```
%matplotlib inline

for matrix_t in matrixtypes:
    for rank in ranks[matrix_t]:
        _, ax = plt.subplots(figsize=(12.7, 7))

        max_y = max(exec_times[matrix_t][rank])
        max_y += 0.1 * max_y
        plt.ylim(0, max_y)

        ax.scatter(
            exec_ranks[matrix_t][rank],
            exec_times[matrix_t][rank],
            label=f'{matrix_t}'
        )

        ax.plot(
            exec_ranks[matrix_t][rank],
            exec_times[matrix_t][rank],
            linestyle='--'
        )

        ax.set(
            xlabel='Stopień dopełnienia Schura',
            ylabel='Czas obliczeń [s]',
            title=f'Macierz stopnia {rank}, typu: {matrix_t}'
        )
```

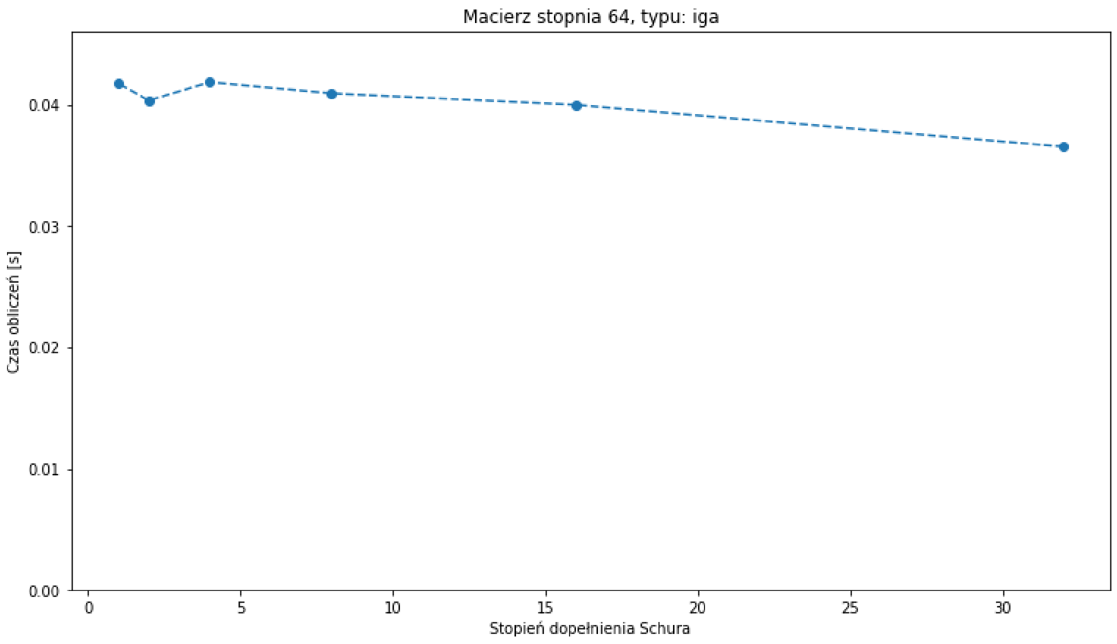
```
/tmp/ipykernel_186981/2887758238.py:5: RuntimeWarning: More than 20 figures have
been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  _, ax = plt.subplots(figsize=(12.7, 7))
```

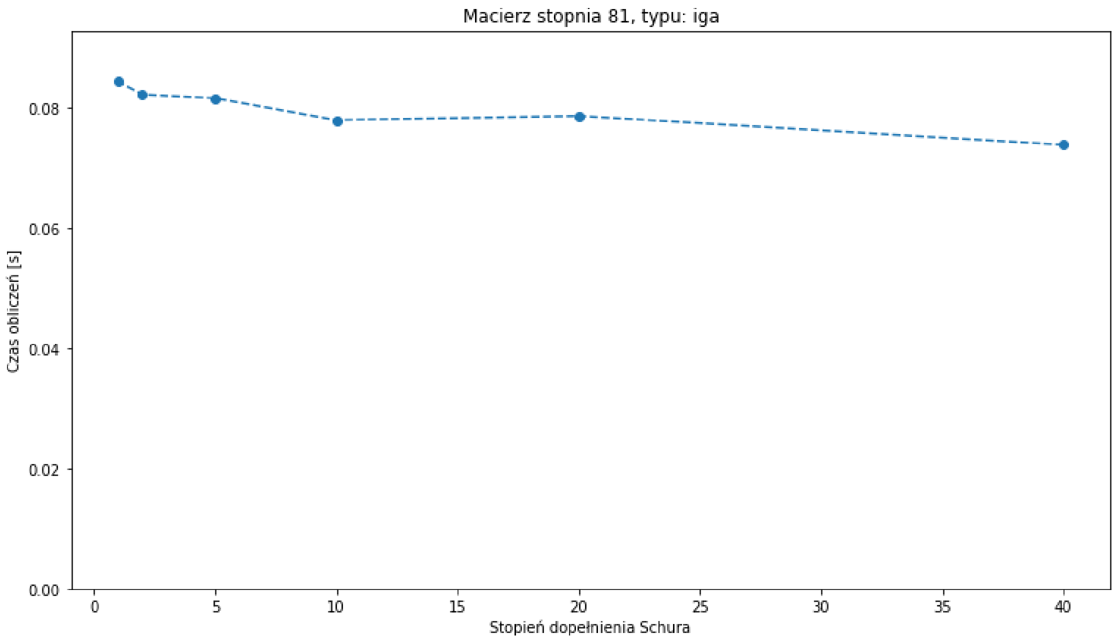

Macierz stopnia 16, typu: iga
Czas obliczeń [s]
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
0.0012
1
2
3
4
5
6
7
8
Stopień dopełnienia Schura

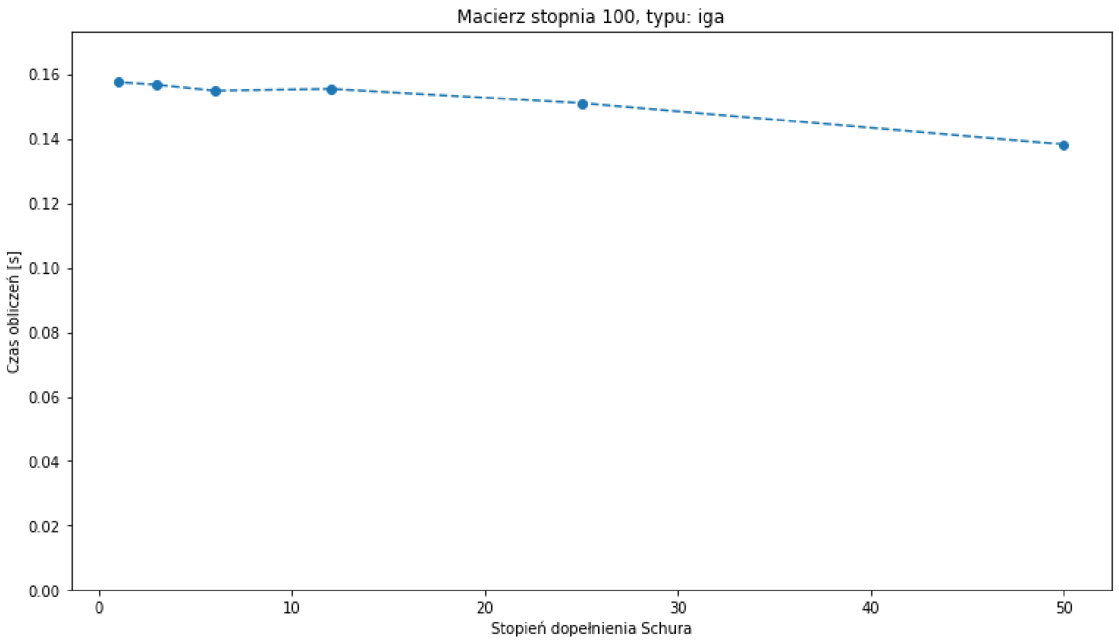

Macierz stopnia 25, typu: iga
Czas obliczeń [s]
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
0.0035
0.0040
2
4
6
8
10
12
Stopień dopełnienia Schura

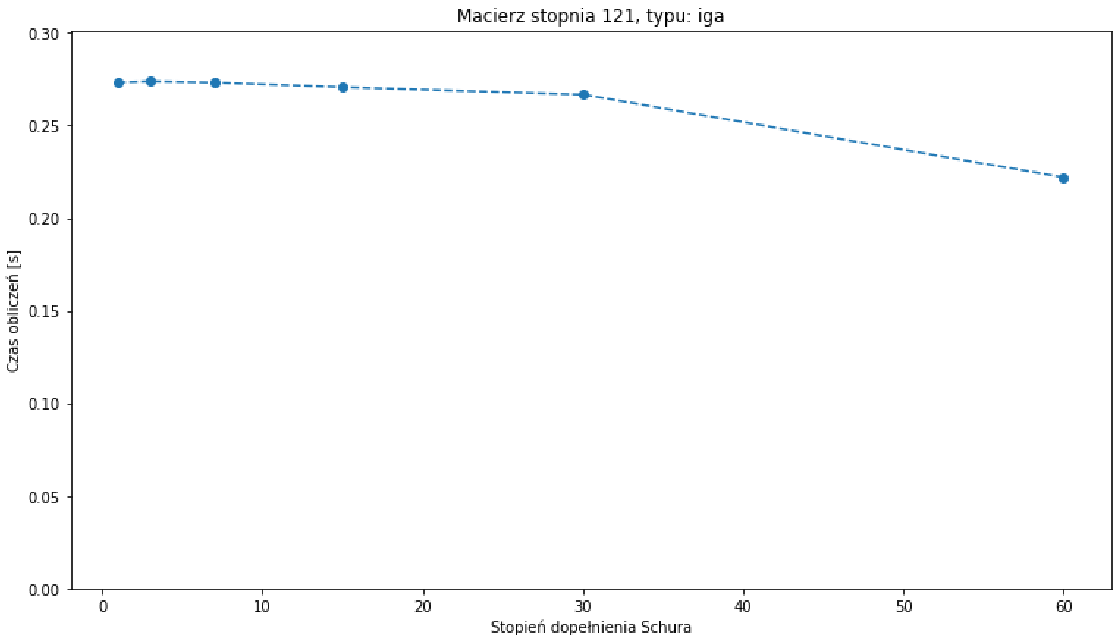

Macierz stopnia 36, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0.000
0.002
0.004
0.006
0.008
2.5
5.0
7.5
10.0
12.5
15.0
17.5

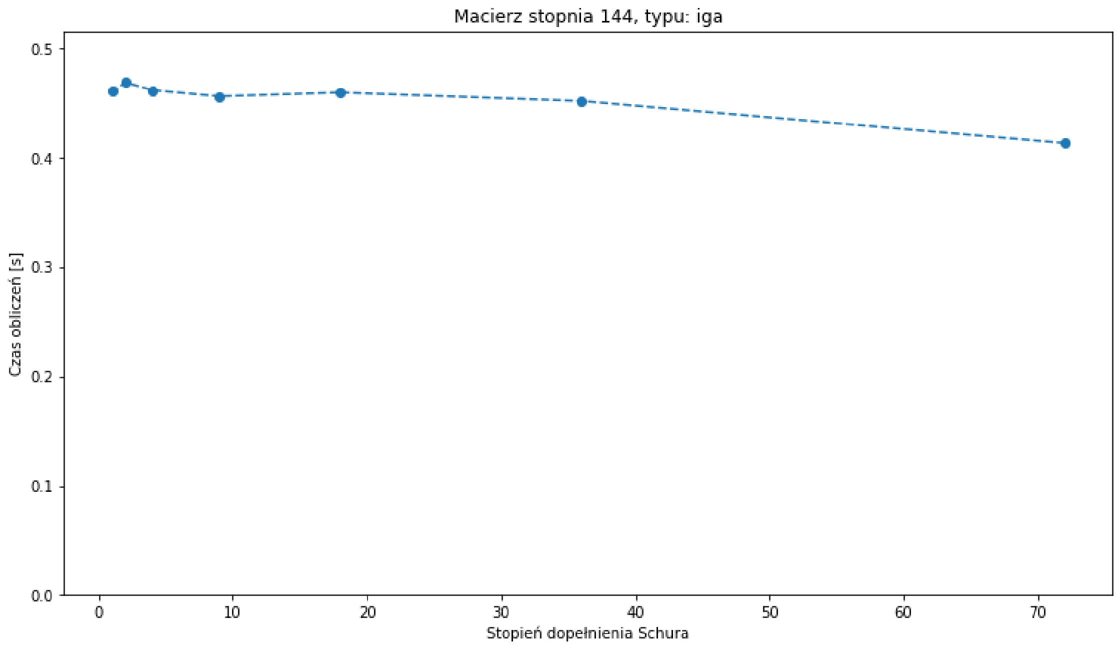

Macierz stopnia 49, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0
5
10
15
20
25

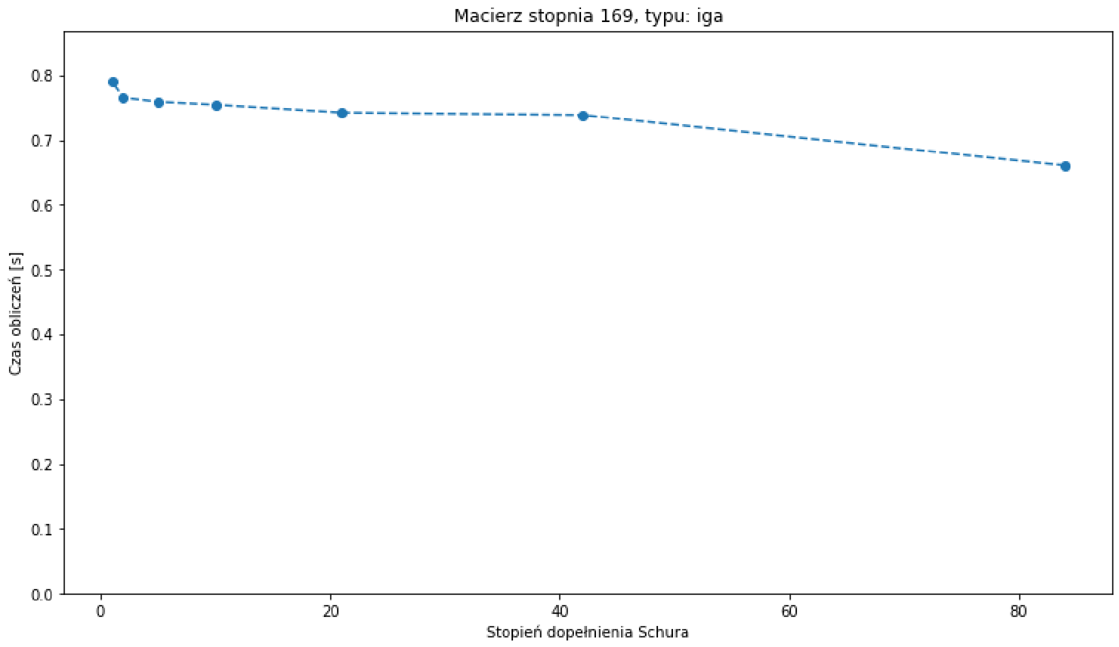

Macierz stopnia 64, typu: iga
Czas obliczeń [s]
0.00
0.01
0.02
0.03
0.04
0
5
10
15
20
25
30
Stopień dopełnienia Schura

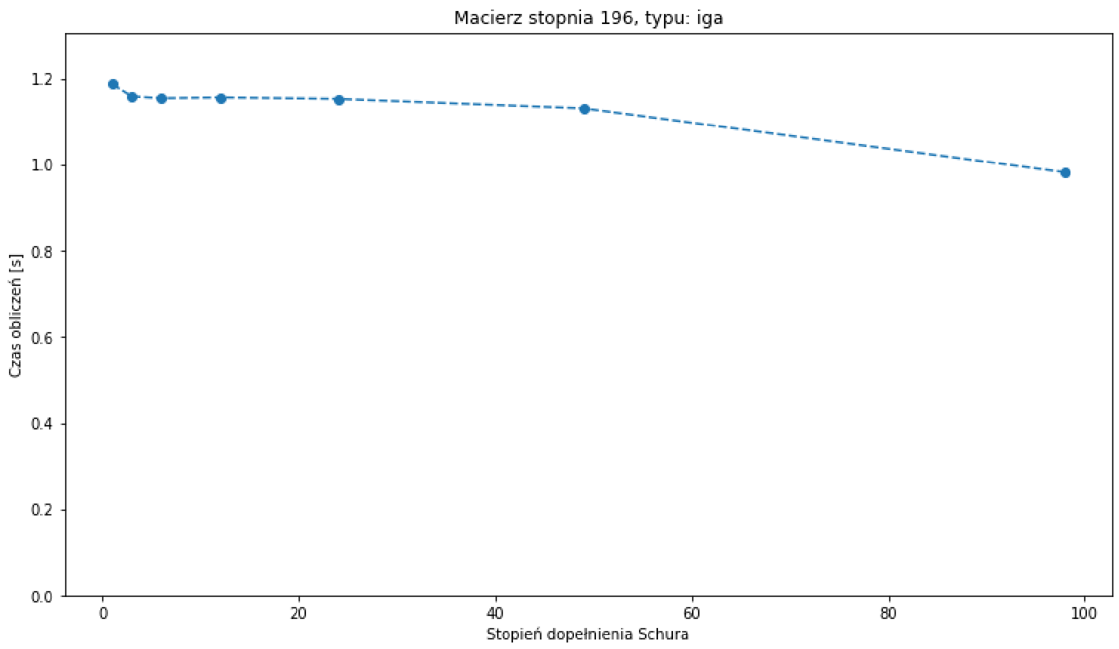

Macierz stopnia 81, typu: iga
Czas obliczeń [s]
0.00
0.02
0.04
0.06
0.08
0
5
10
15
20
25
30
35
40
Stopień dopełnienia Schura

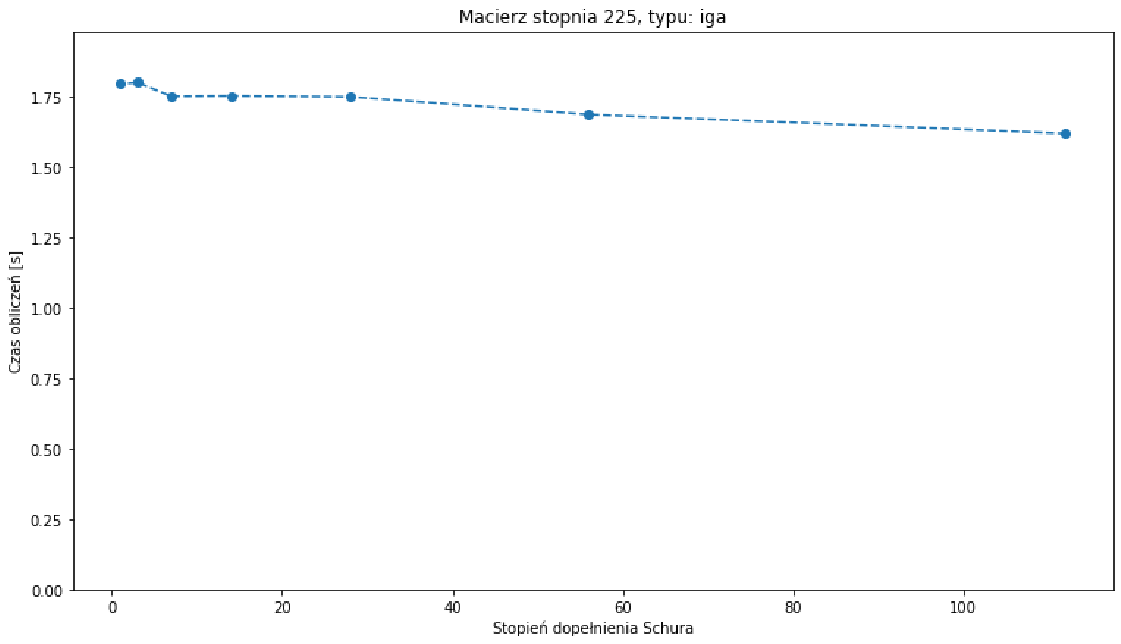

Macierz stopnia 100, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0
10
20
30
40
50

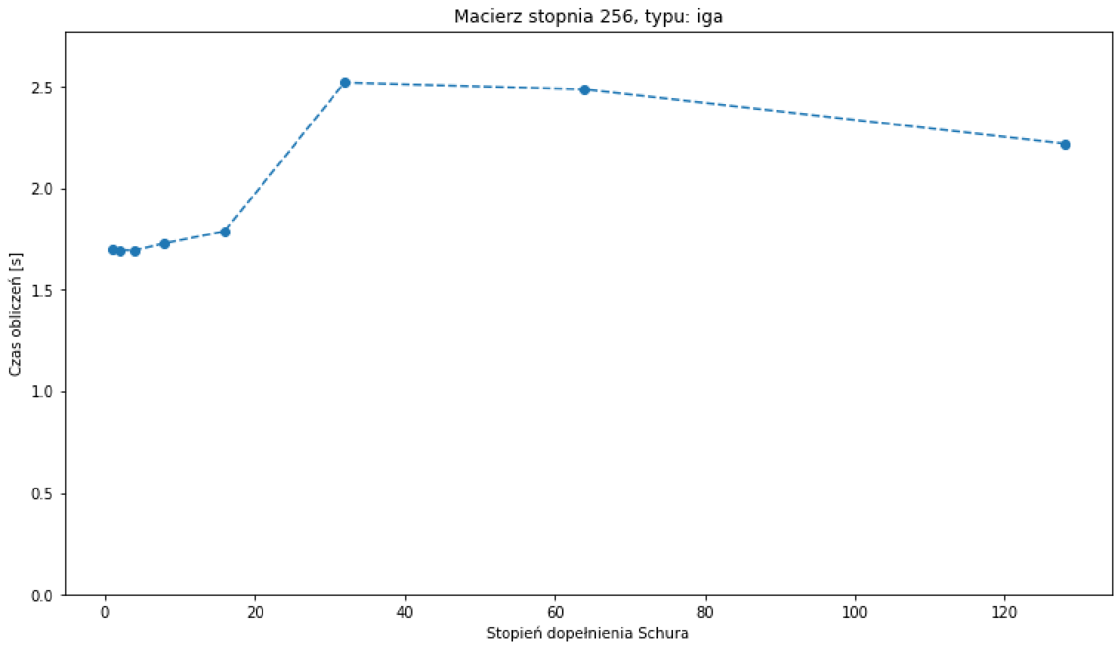

Macierz stopnia 121, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0
10
20
30
40
50
60

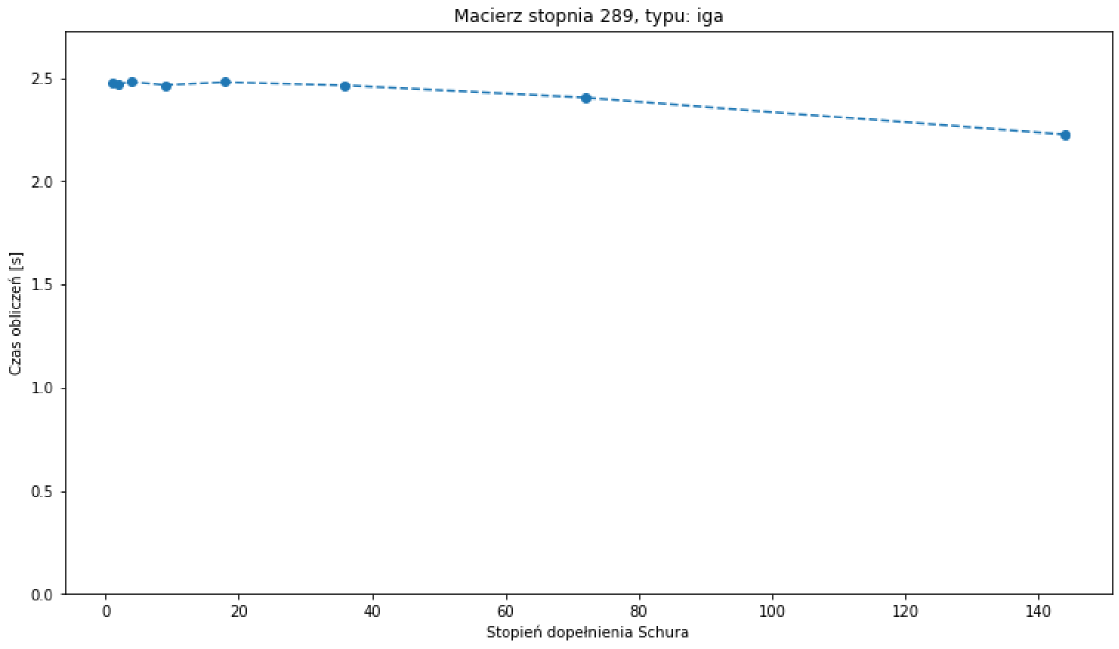

Macierz stopnia 144, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0.0
0.1
0.2
0.3
0.4
0.5
0
10
20
30
40
50
60
70

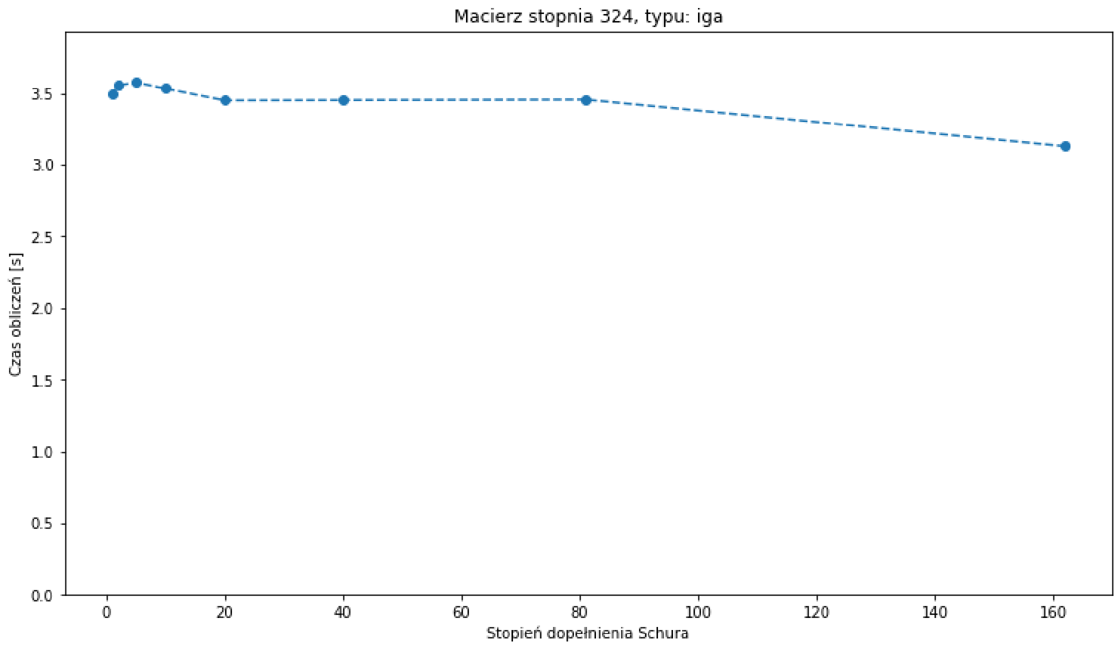

Macierz stopnia 169, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
20
40
60
80

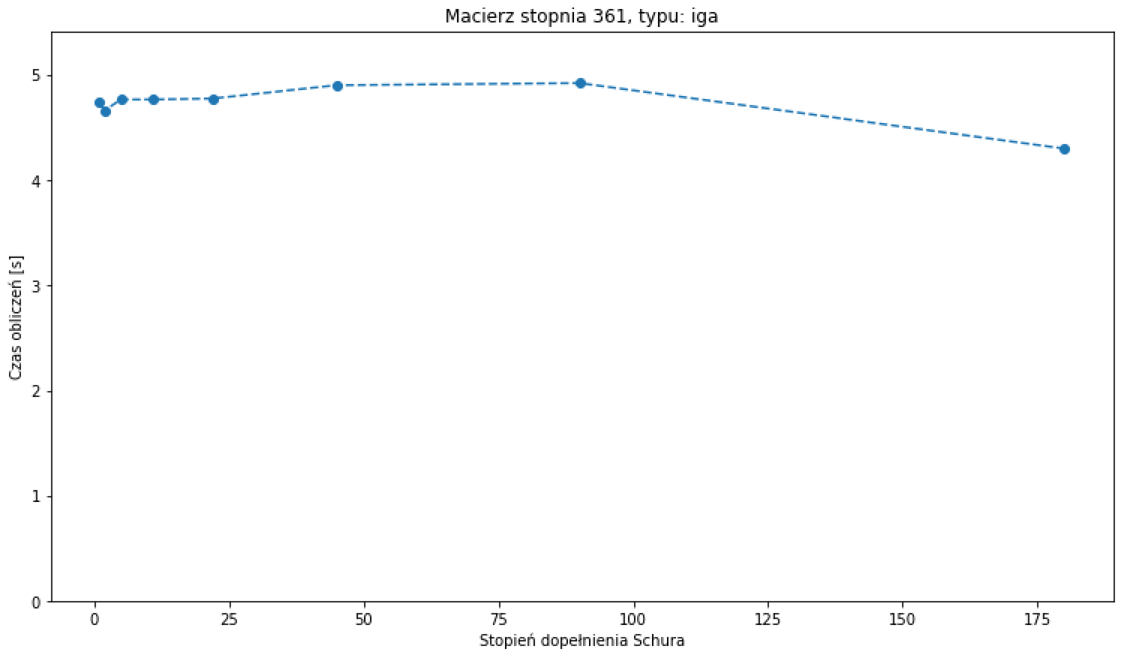

Macierz stopnia 196, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
20
40
60
80
100

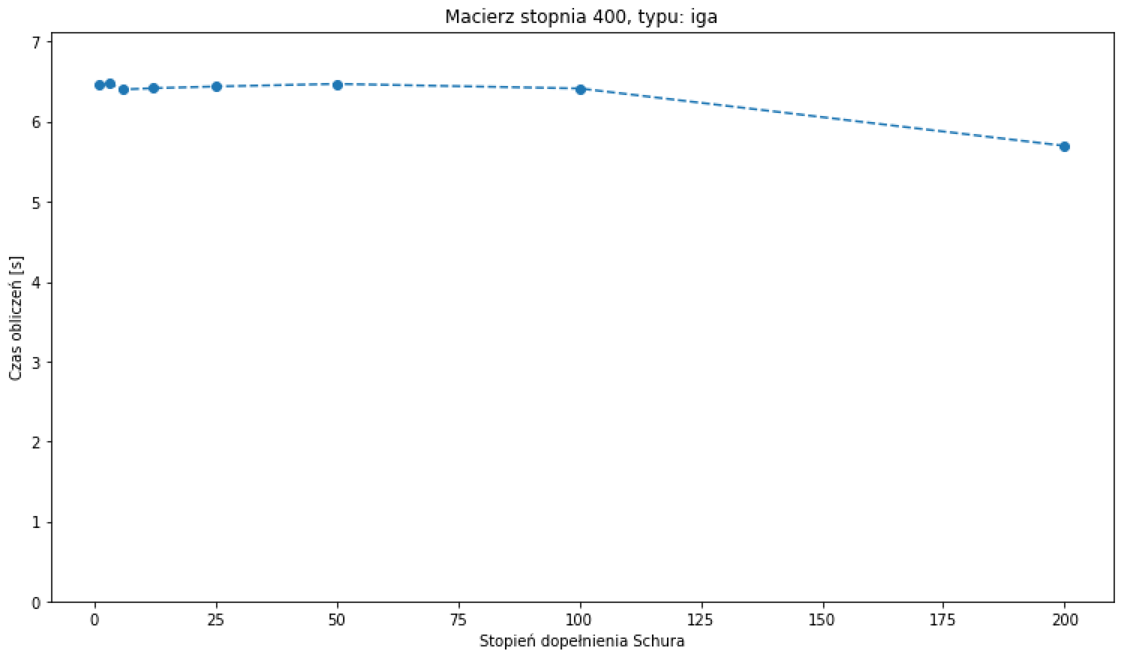

Macierz stopnia 225, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0
20
40
60
80
100

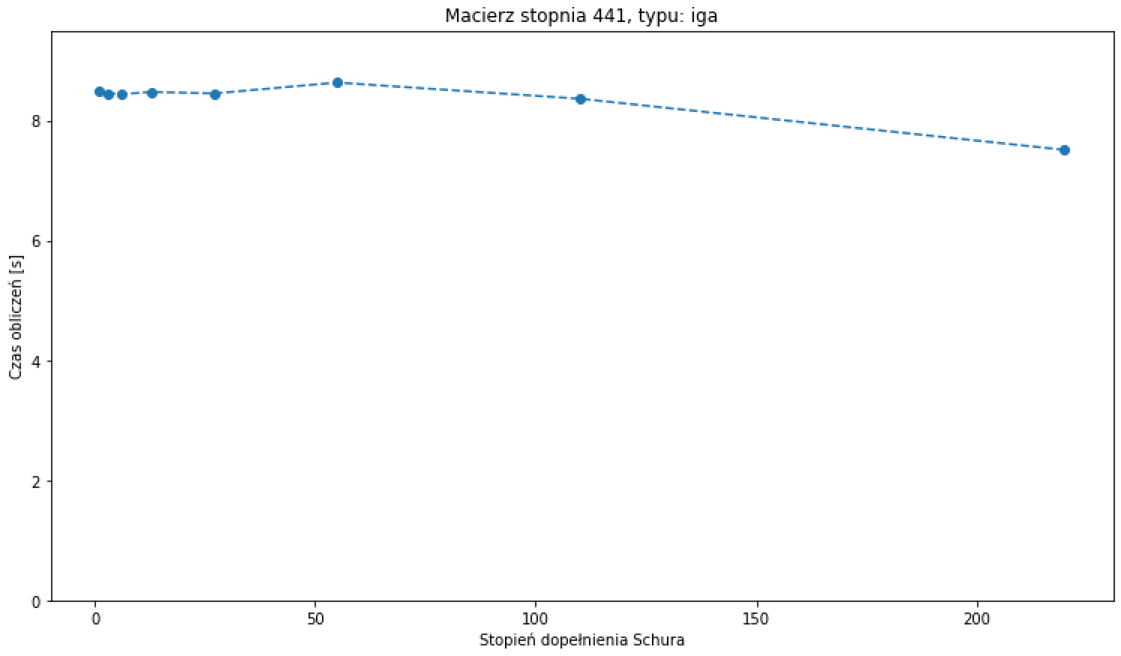

Macierz stopnia 256, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100
120

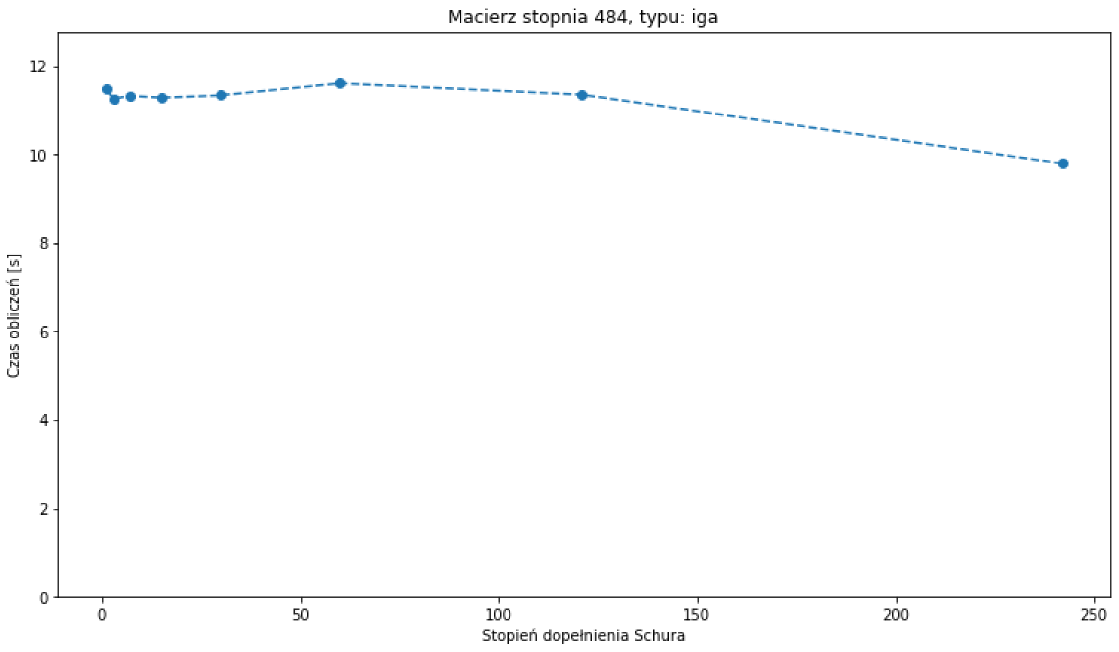

Macierz stopnia 289, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100
120
140

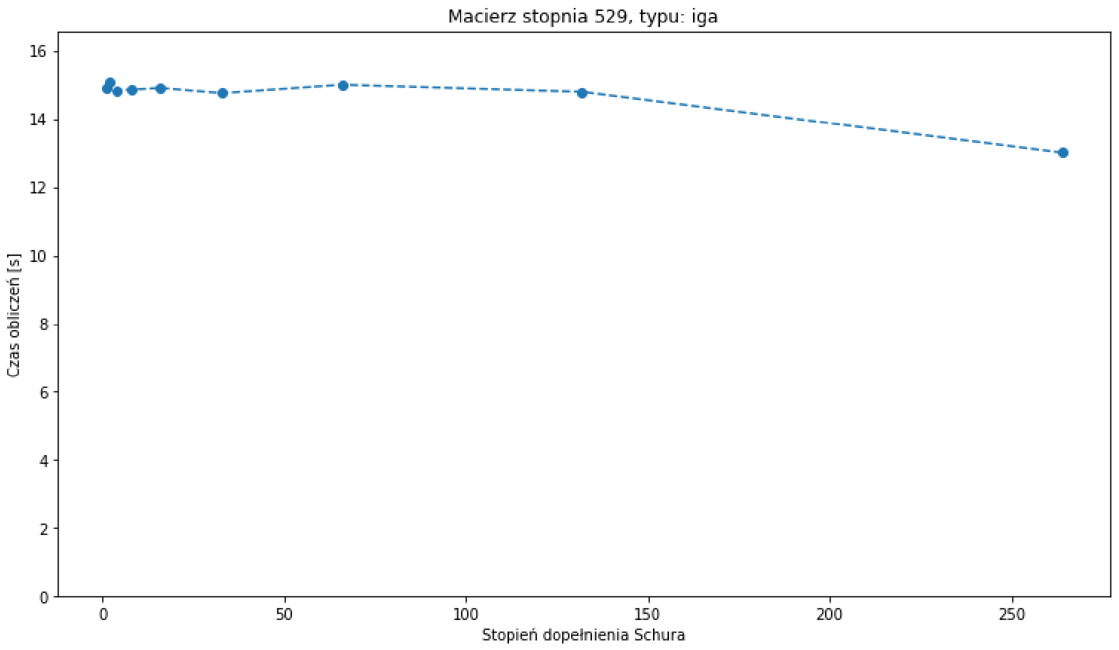

Macierz stopnia 324, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100
120
140
160

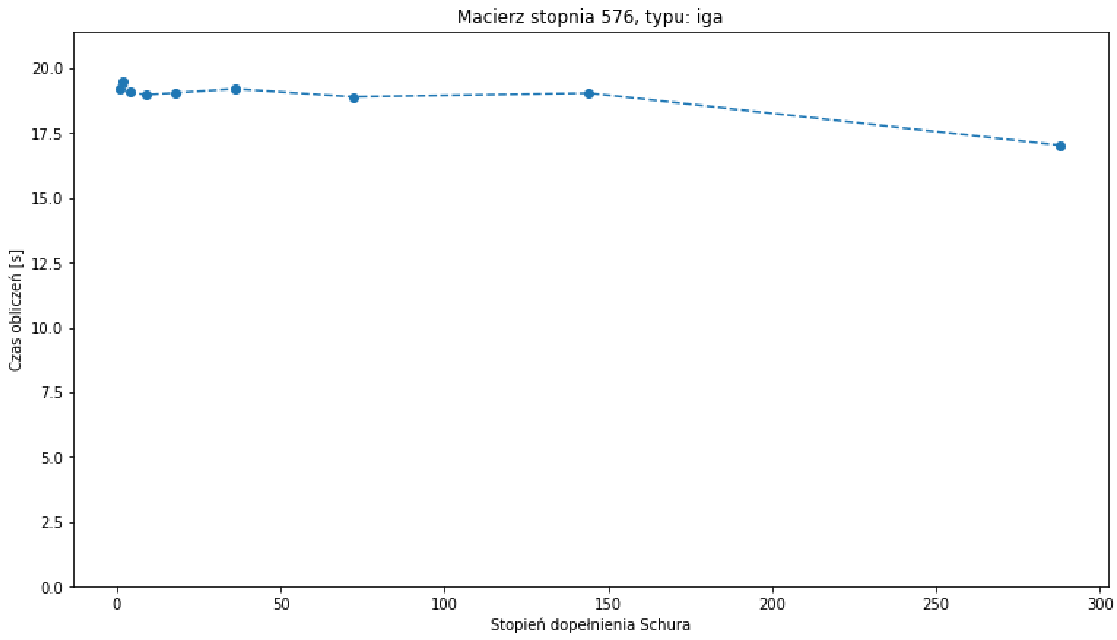

Macierz stopnia 361, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0
1
2
3
4
5
0
25
50
75
100
125
150
175

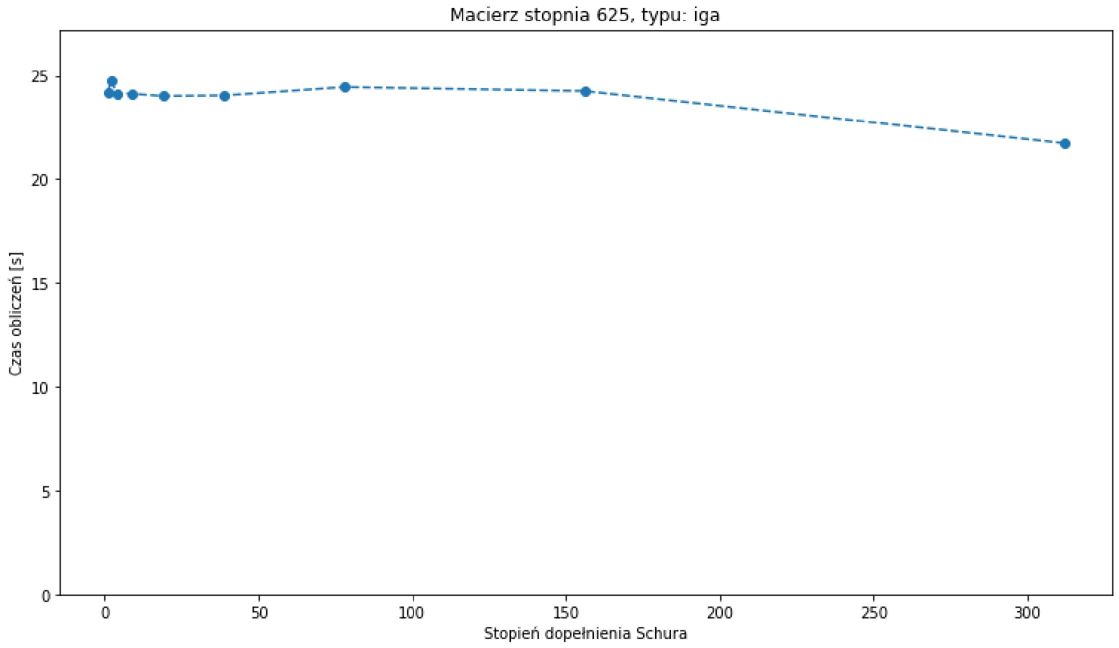

Macierz stopnia 400, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0
1
2
3
4
5
6
7
0
25
50
75
100
125
150
175
200

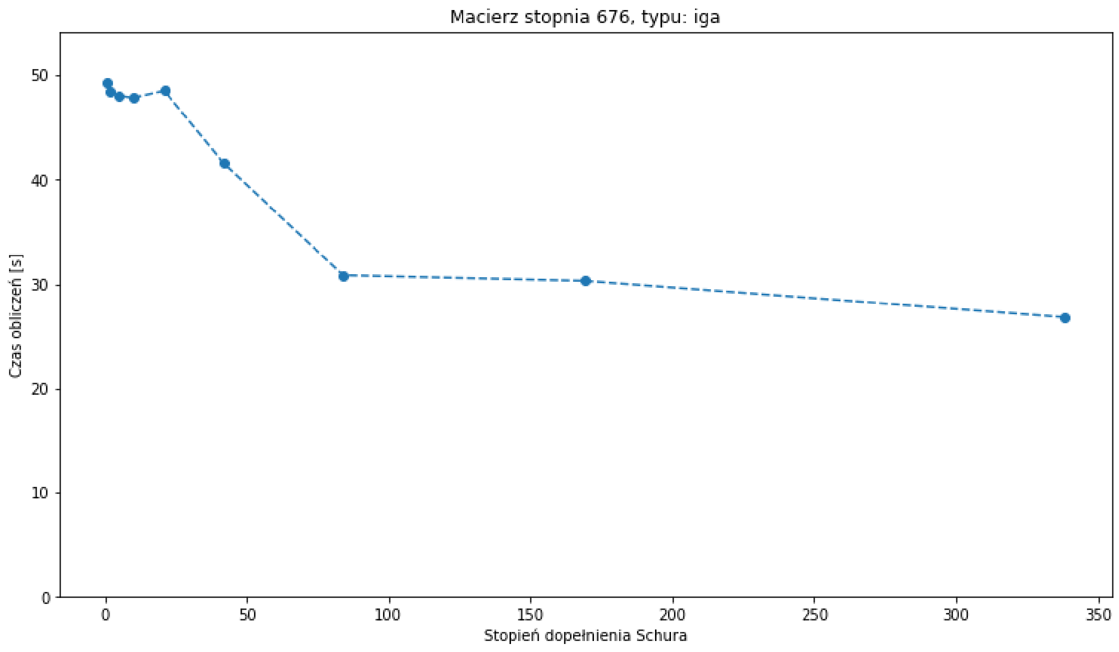

Macierz stopnia 441, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0
2
4
6
8
0
50
100
150
200

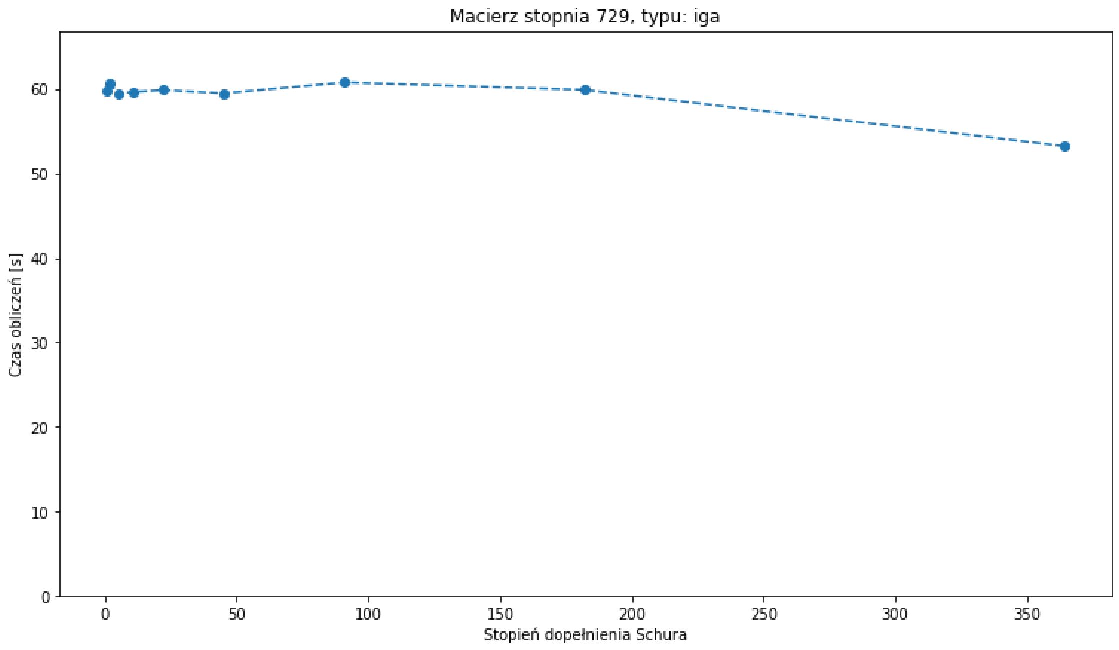

Macierz stopnia 484, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0
2
4
6
8
10
12
0
50
100
150
200
250

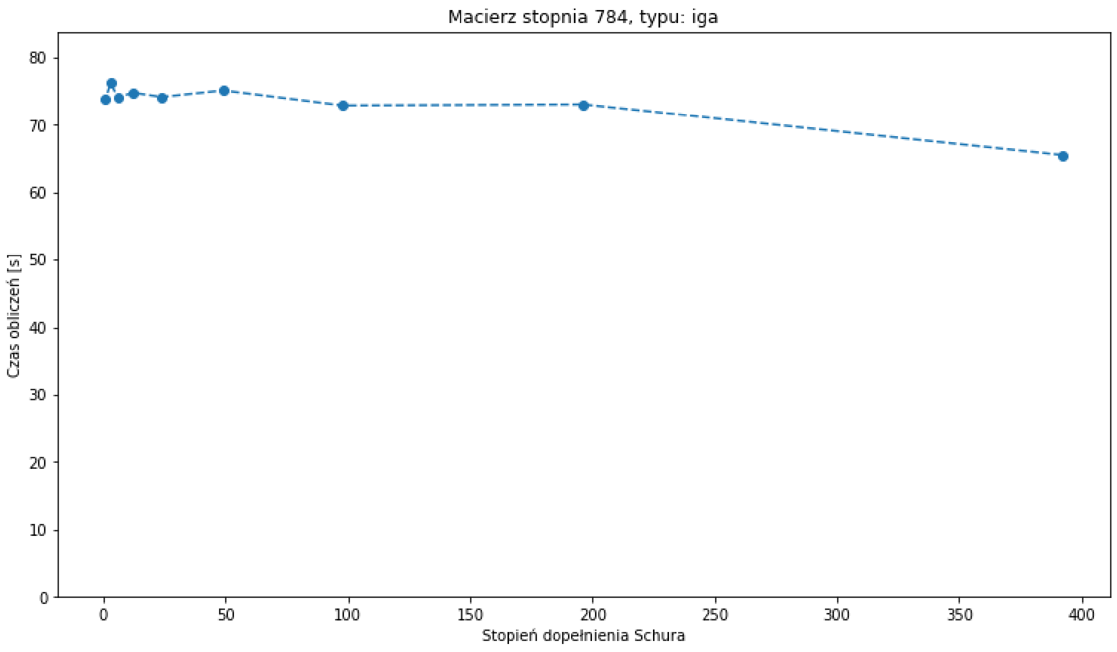

Macierz stopnia 529, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0
2
4
6
8
10
12
14
16
0
50
100
150
200
250

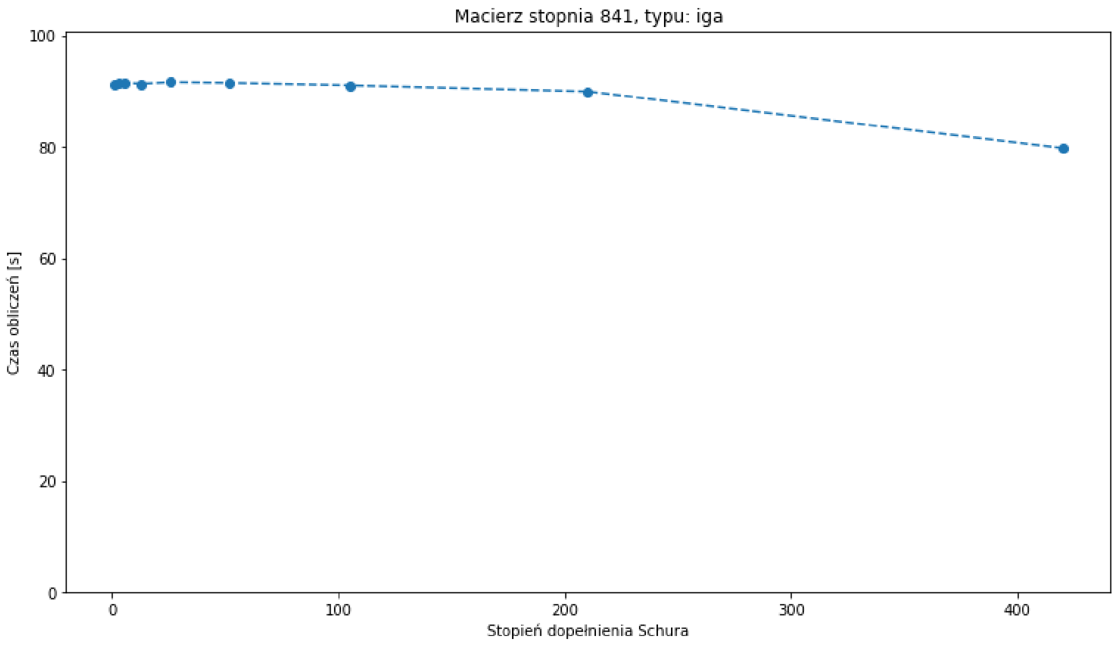

Macierz stopnia 576, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
50
100
150
200
250
300

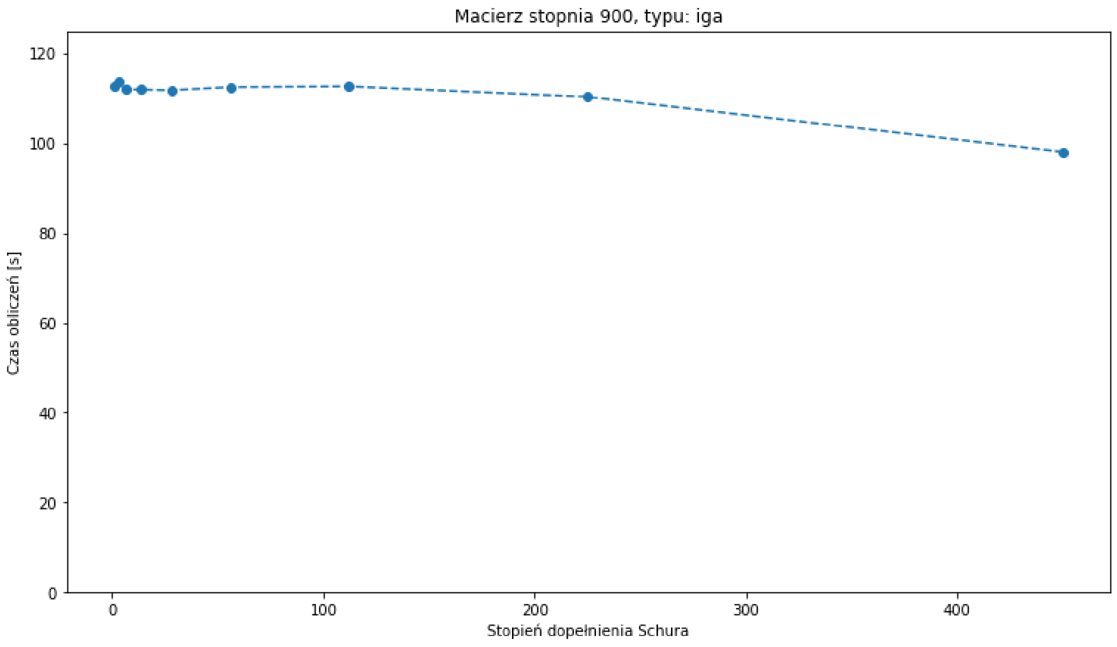

Macierz stopnia 625, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0
5
10
15
20
25
0
50
100
150
200
250
300

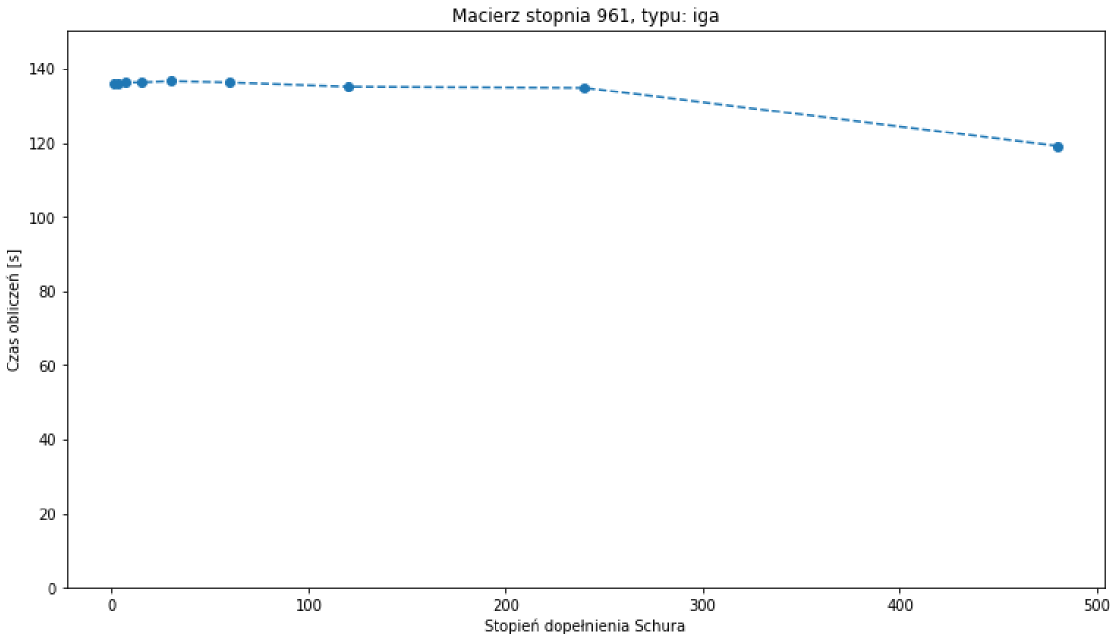

Macierz stopnia 676, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0
10
20
30
40
50
0
50
100
150
200
250
300
350

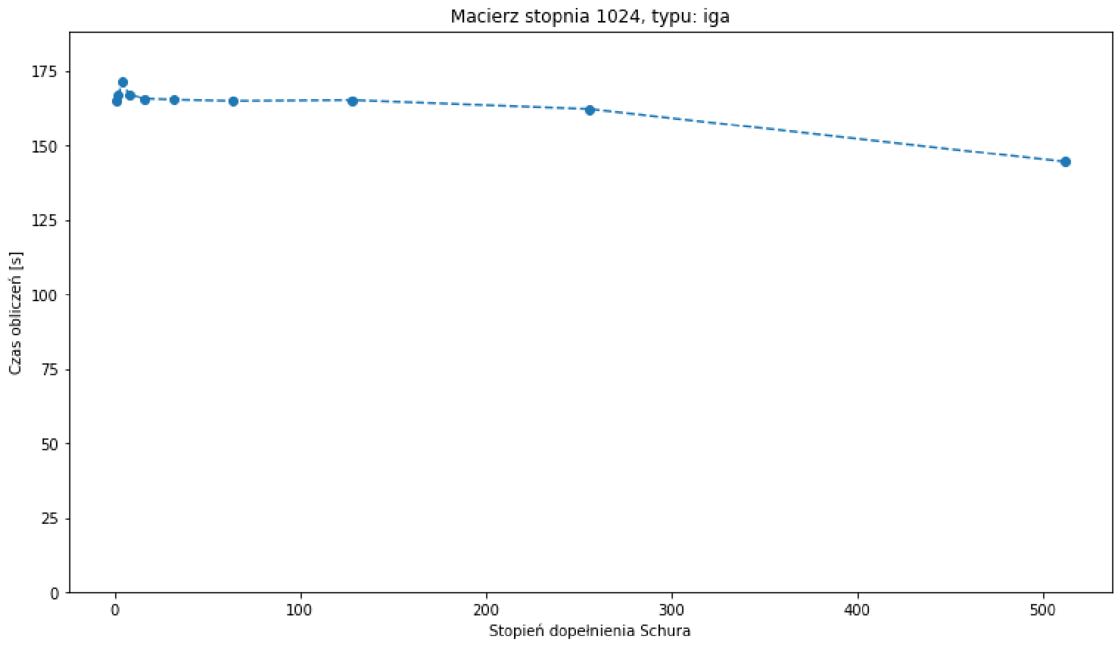

Macierz stopnia 729, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0
10
20
30
40
50
60
0
50
100
150
200
250
300
350

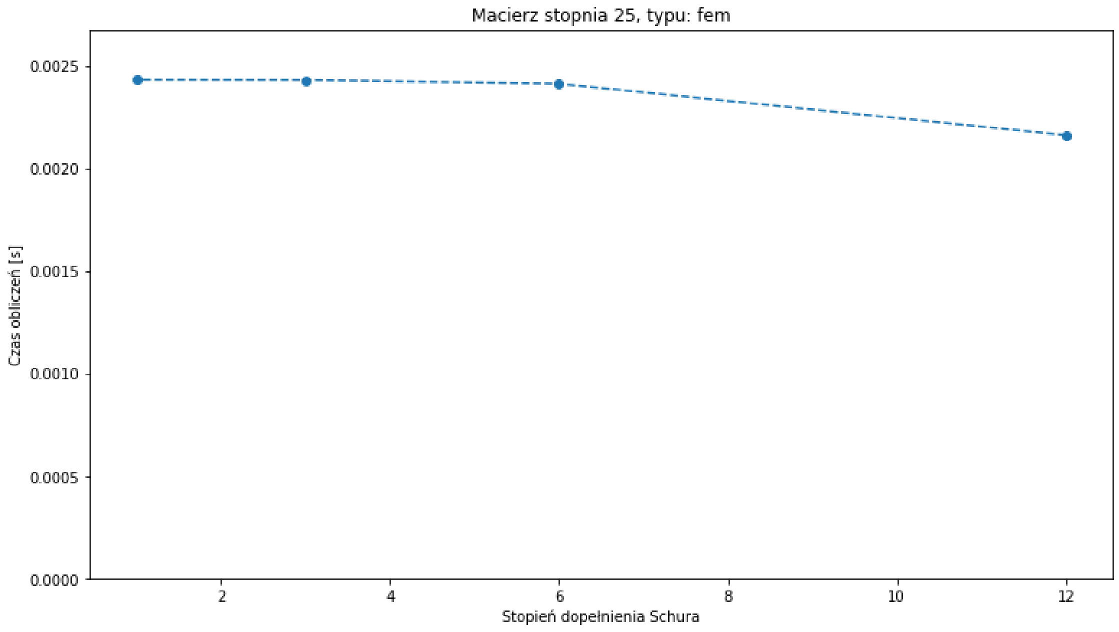

Macierz stopnia 784, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0
10
20
30
40
50
60
70
80
0
50
100
150
200
250
300
350
400

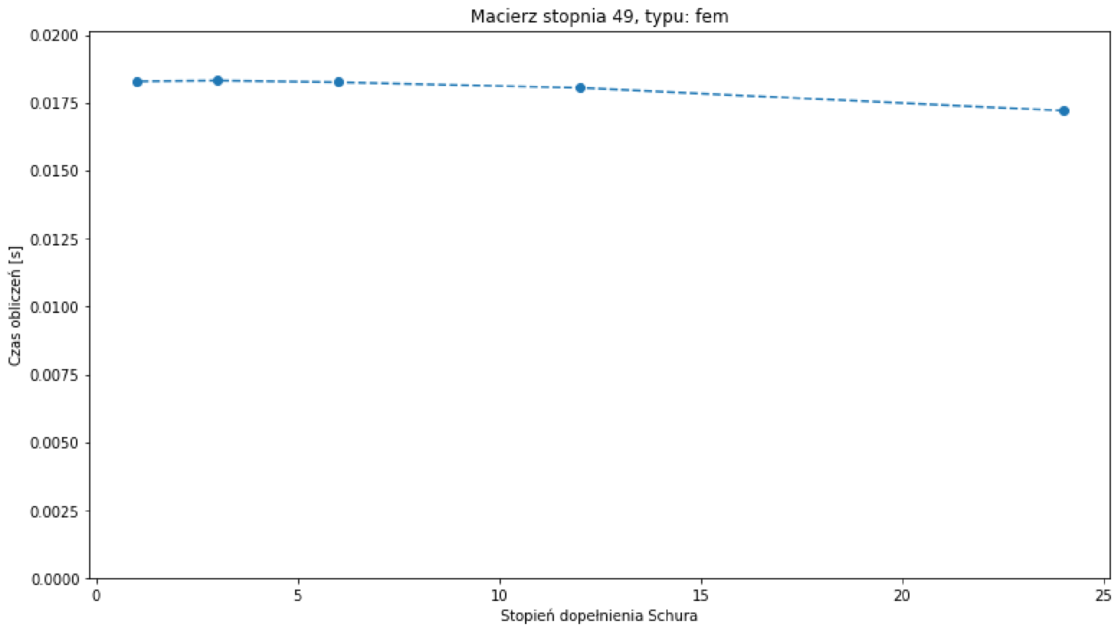

Macierz stopnia 841, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0
20
40
60
80
100
0
100
200
300
400

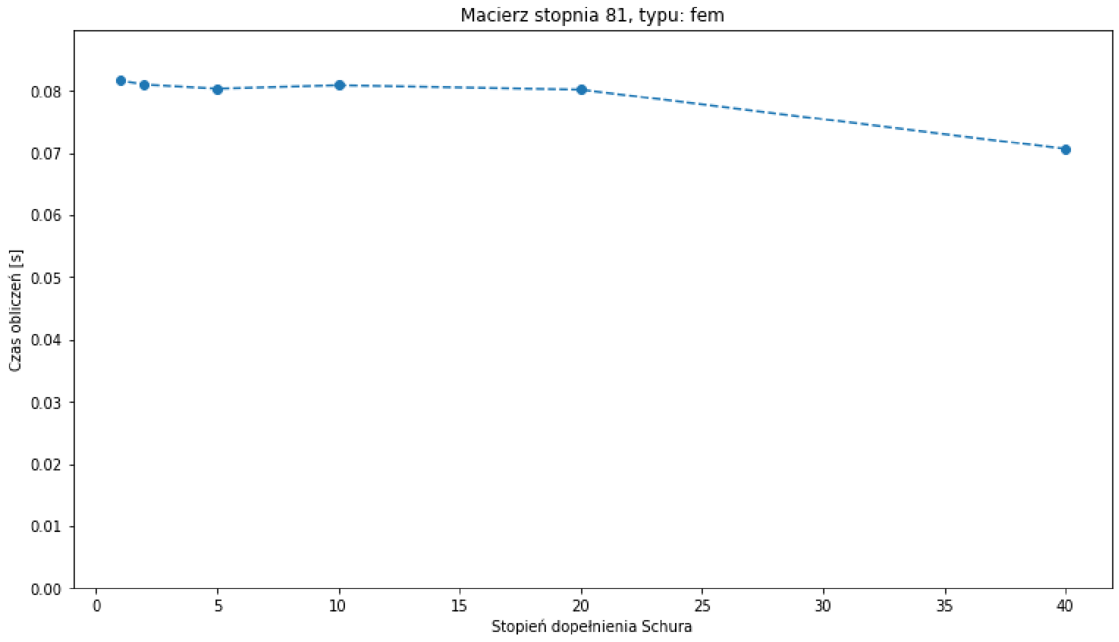

Macierz stopnia 900, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0
20
40
60
80
100
120
0
100
200
300
400

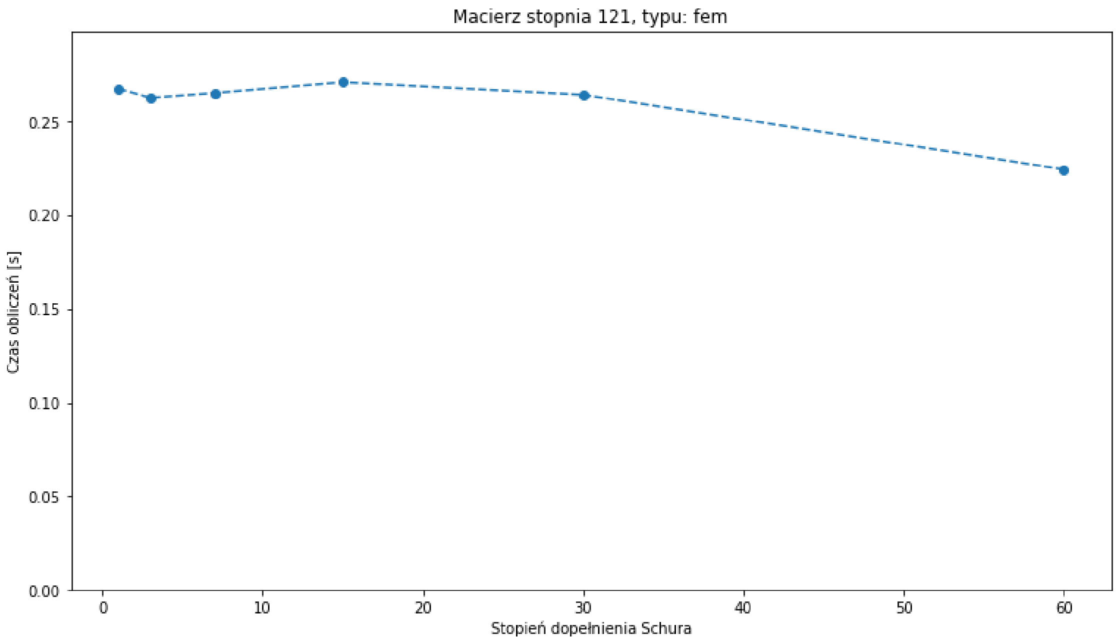

Macierz stopnia 961, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0
20
40
60
80
100
120
140
0
100
200
300
400
500

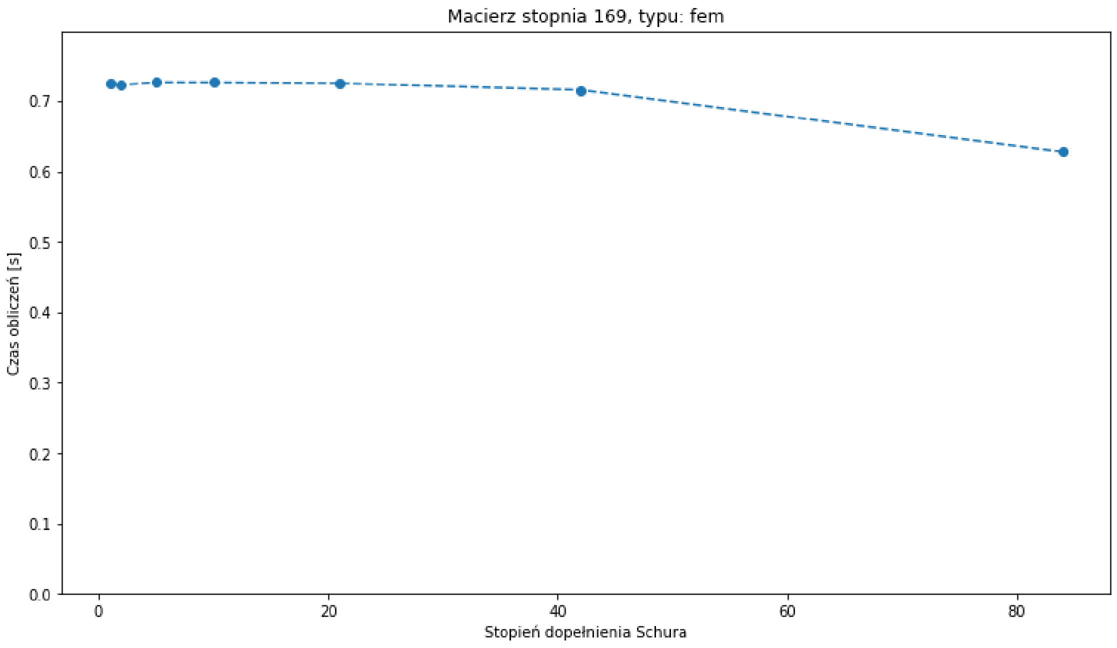

Macierz stopnia 1024, typu: iga
Czas obliczeń [s]
Stopień dopełnienia Schura
0
25
50
75
100
125
150
175
0
100
200
300
400
500

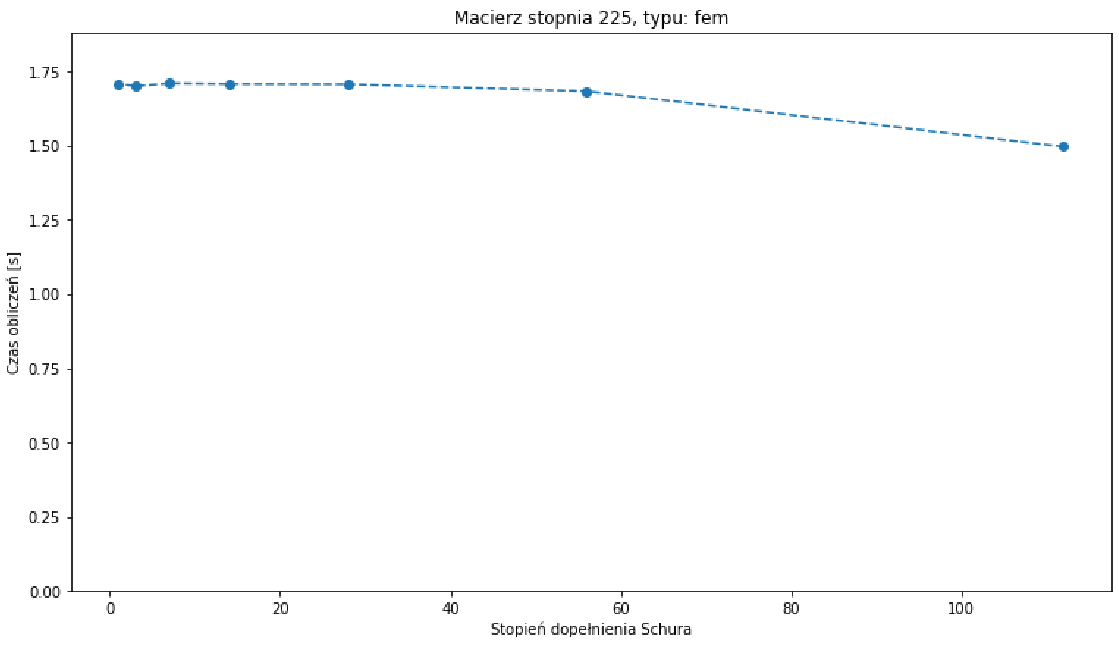

Macierz stopnia 25, typu: fem
Czas obliczeń [s]
Stopień dopełnienia Schura
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
2
4
6
8
10
12

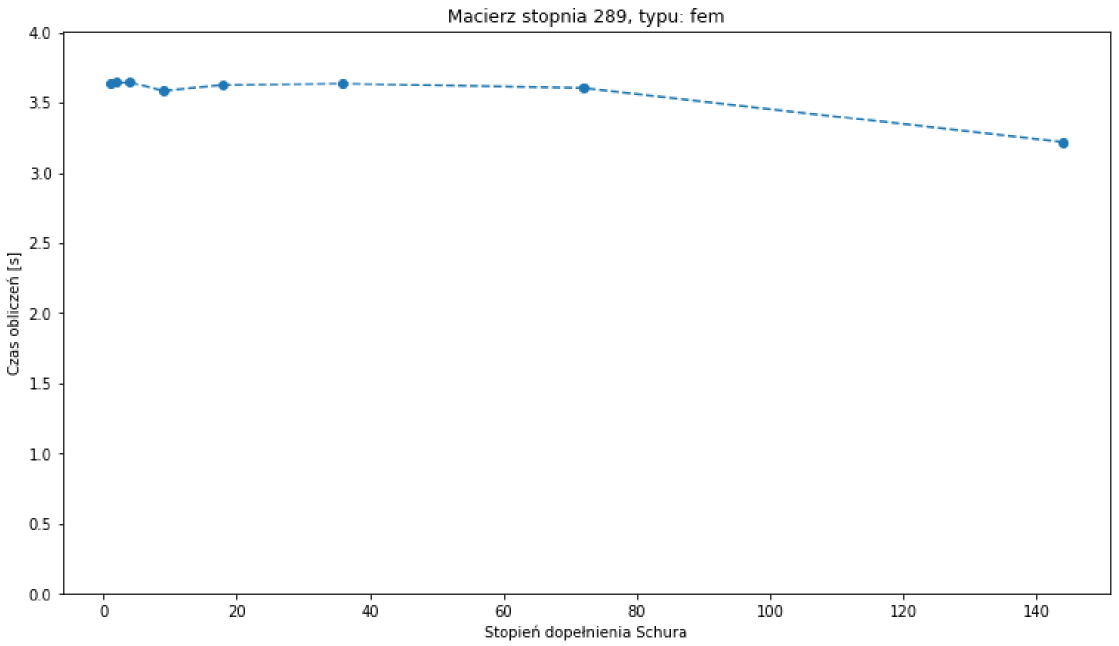

Macierz stopnia 49, typu: fem
Czas obliczeń [s]
Stopień dopełnienia Schura
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
0
5
10
15
20
25

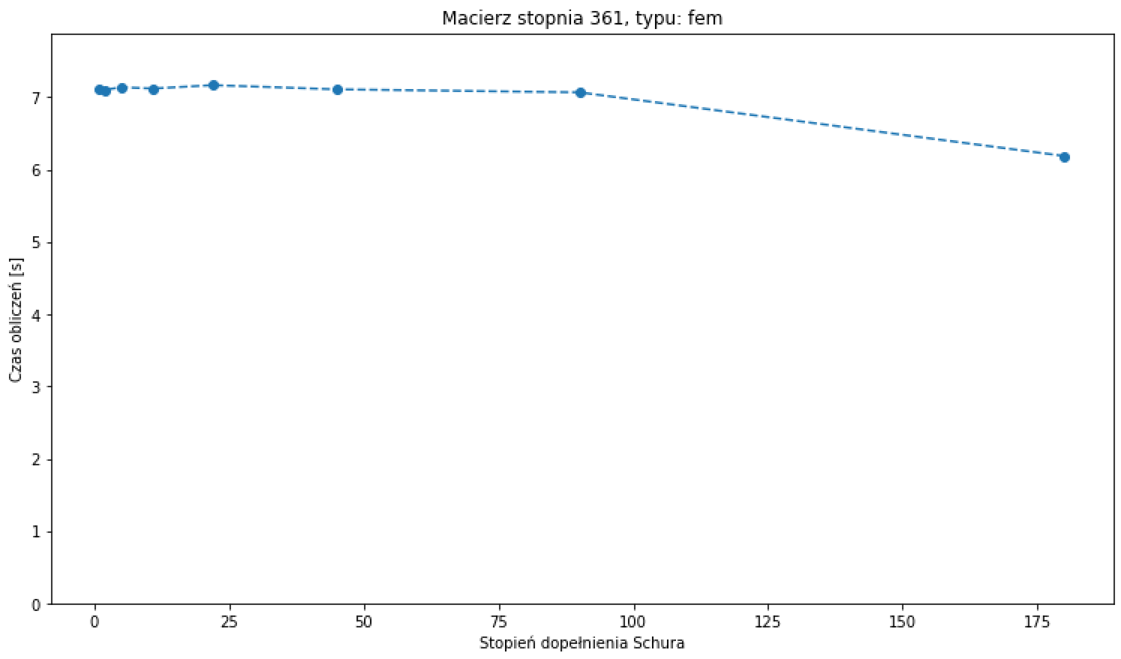

Macierz stopnia 81, typu: fem
Czas obliczeń [s]
Stopień dopełnienia Schura
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0
5
10
15
20
25
30
35
40


Macierz stopnia 121, typu: fem
Czas obliczeń [s]
Stopień dopełnienia Schura
0.00
0.05
0.10
0.15
0.20
0.25
0
10
20
30
40
50
60

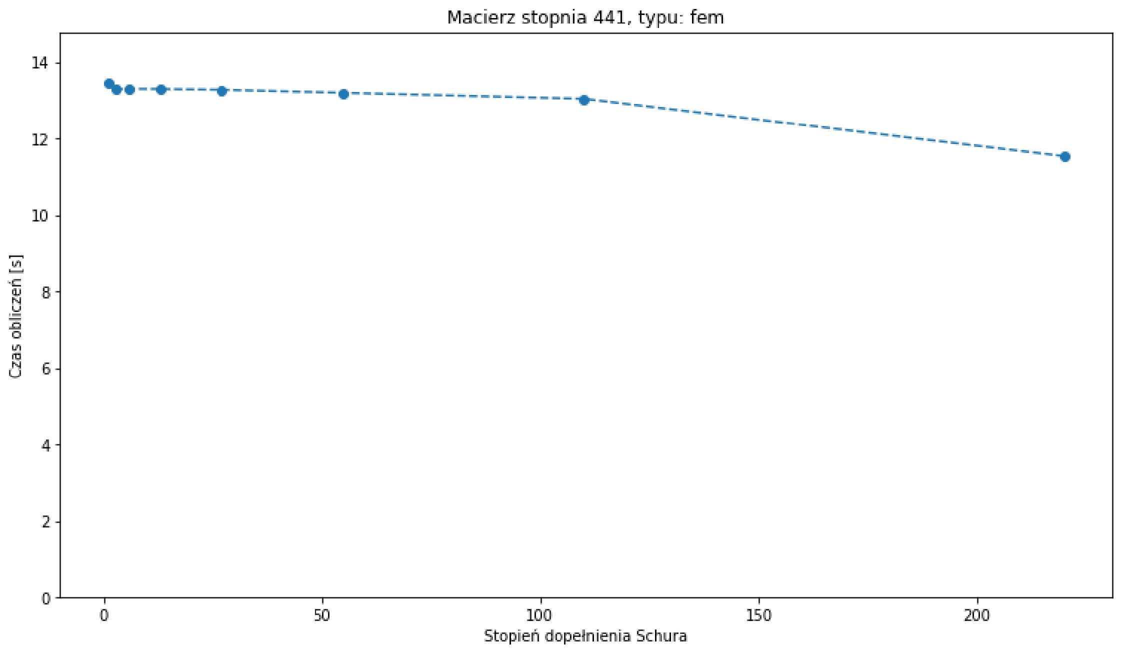

Macierz stopnia 169, typu: fem
Czas obliczeń [s]
Stopień dopełnienia Schura
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
20
40
60
80


Macierz stopnia 225, typu: fem
Czas obliczeń [s]
Stopień dopełnienia Schura
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0
20
40
60
80
100

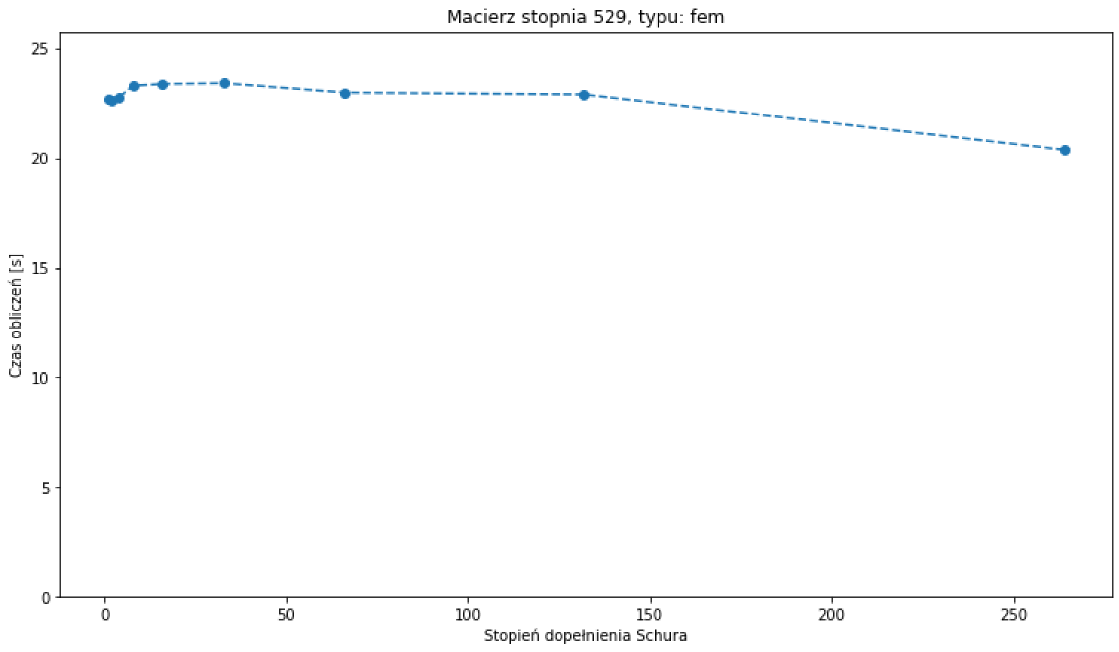

Macierz stopnia 289, typu: fem
Czas obliczeń [s]
Stopień dopełnienia Schura
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
20
40
60
80
100
120
140

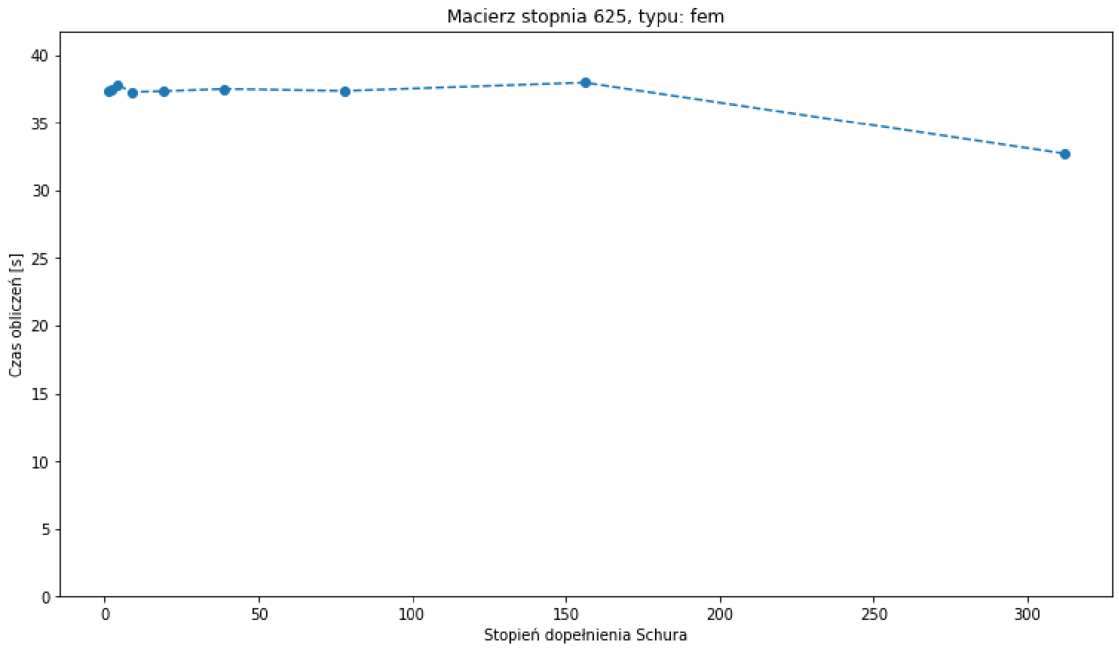


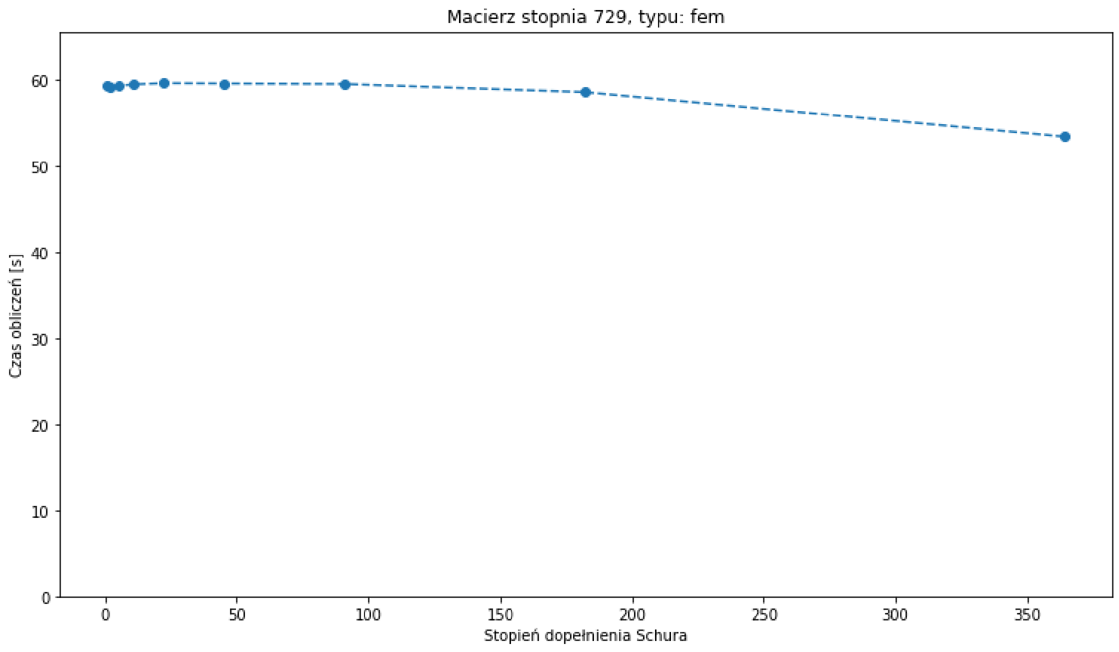
Macierz stopnia 361, typu: fem
Czas obliczeń [s]
Stopień dopełnienia Schura
0
1
2
3
4
5
6
7
0
25
50
75
100
125
150
175

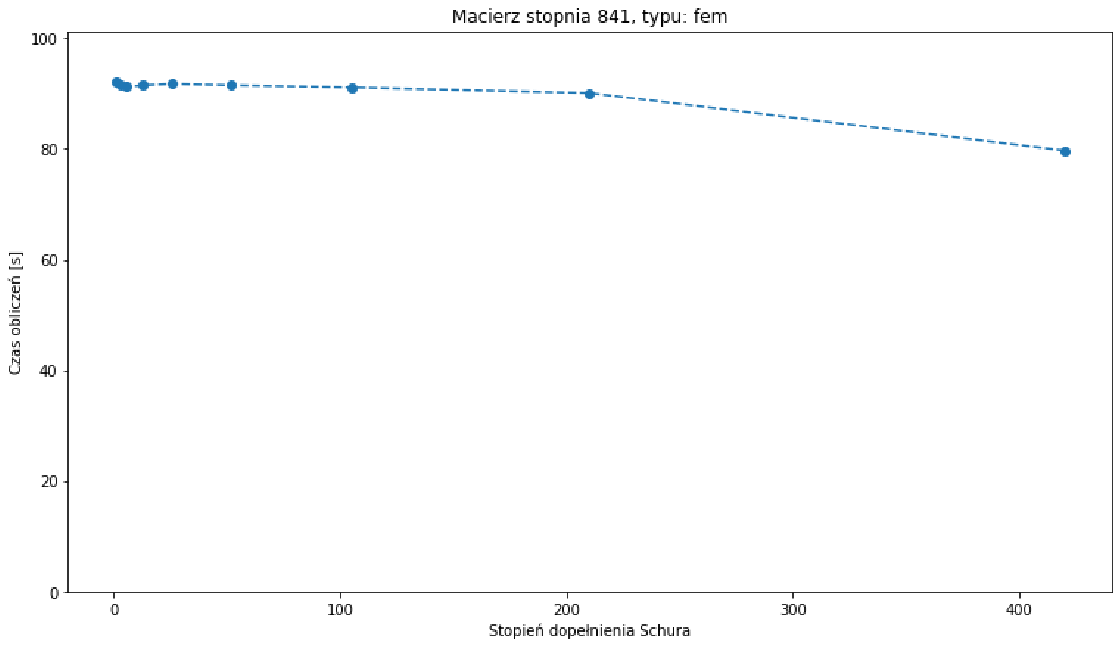


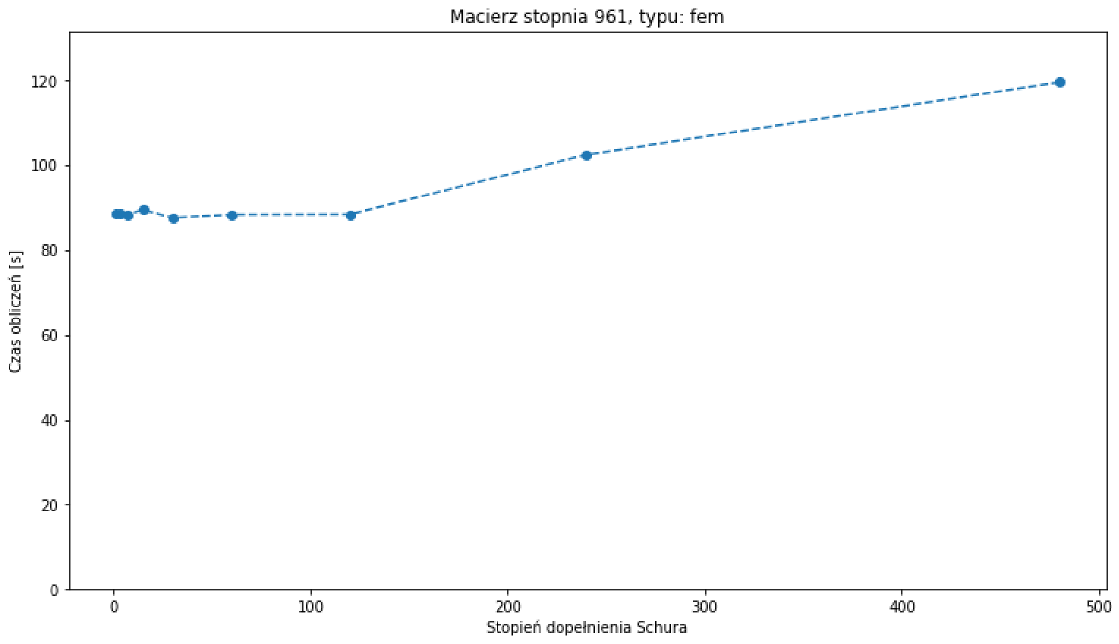
Macierz stopnia 441, typu: fem
Czas obliczeń [s]
Stopień dopełnienia Schura
0
2
4
6
8
10
12
14
0
50
100
150
200

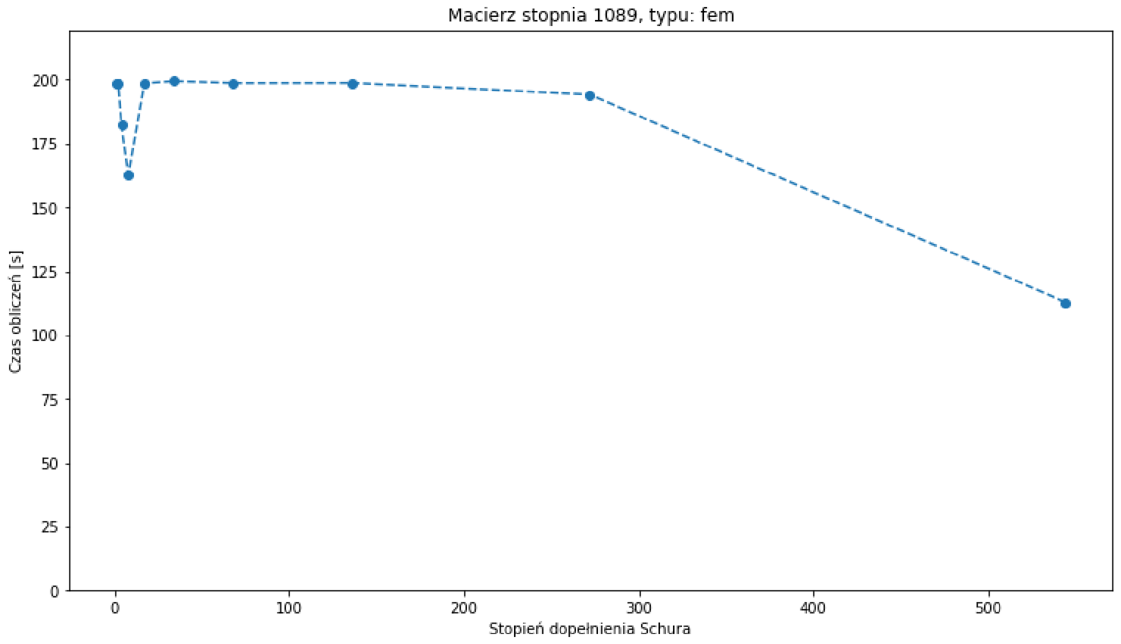

Macierz stopnia 529, typu: fem
Czas obliczeń [s]
Stopień dopełnienia Schura
0
5
10
15
20
25
0
50
100
150
200
250


Macierz stopnia 625, typu: fem
Czas obliczeń [s]
Stopień dopełnienia Schura
0
5
10
15
20
25
30
35
40
0
50
100
150
200
250
300


Macierz stopnia 729, typu: fem
Czas obliczeń [s]
Stopień dopełnienia Schura
0
10
20
30
40
50
60
0
50
100
150
200
250
300
350


Macierz stopnia 841, typu: fem
Czas obliczeń [s]
Stopień dopełnienia Schura
0
20
40
60
80
100
0
100
200
300
400





1.6 Koszt obliczeniowy zaimplementowanego algorytmu

Niech n będzie rozmiarem macierzy, a m rozmiarem obliczanego dopełnienia Schura.
Wtedy w algorytmie transform_matrix_gaussian_elem zewnętrzna pętla wykonuje się $n-m$ razy.
Kolejna $n-i$ razy i w niej wykonuje się jedną operację oraz następną pętlę wykonywaną znowu

$n - i$ razy. W tej ostatniej pętli wykonuje się 3 operacje zmiennoprzecinkowe. W sumie daje to:

$\sum_{i=0 \dots n-m-1} \sum_{j=i+1 \dots n-1} (1 + \sum_{k=i+1 \dots n-1} 3) = n^2(n-1) - m^3 + m^2$

Przykładowo dla macierzy stopnia 100 i dopełnienia Schura stopnia 25:

[15]:
```
n = 100
m = 25
FLOP = n**2 * (n - 1) - m**3 + m**2
print(FLOP)
```

```
975000
```

1.7 Koszt pamięciowy zaimplementowanego algorytmu

Obliczenia są analogiczne do tych powyższych. Jedynie w pierwszej pętli następuje jedno odwołanie do pamięci, w drugiej pętli dwa, a w trzeciej trzy. Daje to wynik:

$\sum_{i=0 \dots n-m-1} (1 + \sum_{j=i+1 \dots n-1} (2 + \sum_{k=i+1 \dots n-1} 3)) = n^3 - \frac{n^2}{2} + \frac{n}{2} - m^3 + \frac{m^2}{2} - \frac{m}{2} - 1$

[16]:
```
n = 100
m = 25
MEMOP = n**3 - n**2/2 + n/2 - m**3 + m**2/2 - m/2
print(MEMOP)
```

```
979725.0
```