Zadanie 1 - mnożenie macierzy za pomocą algorytmu klasycznego i Strassena

Mateusz Wejman, Andrzej Starzyk

Marzec 2024

1 Wykonanie

Celem zadania było napisanie programu, który przeprowadza mnożenie macierzy za pomocą klasycznego algorytmu, algorytmu Strassena i hybrydy tych dwóch podejść. Kod został napisany w języku Elixir i zamieszczony na listingu poniżej. W module Matrix zostały zdefiniowane funkcje przeprowadzające mnożenie macierzy i zliczające wykonywane operacje. Moduł Computations służy do przeprowadzenia obliczeń.

Za pomocą przedstawionego poniżej kodu przeprowadzone zostało mnożenie macierzy o rozmiarach $2^k \times 2^k$, dla k od 2 do 10. Zastosowany został algorytm Strassena oraz klasyczny. Dla każdego rozmiaru macierzy rozważane były progi: 2, 5, 8 oraz 4, 32, 256. Algorytm rekurencyjny mnożył macierze metodą klasyczną, gdy ich rozmiary były mniejsze lub równe od danego progu.

```
defmodule Matrix do

  def reset_counters() do
    {:ok, agent}= Agent.start_link(fn -> %{ additions: 0,
    multiplications: 0, subtractions: 0 } end)
    agent
  end




  defp update_counter(operation, agent) do
    operation = case operation do
      :multiplications -> :multiplications
      :additions -> :additions
      :subtractions -> :subtractions
      _ -> IO.puts("Error: Invalid operation.")
    end
    if operation != nil do
     Agent.update(agent, fn counters ->
                                          Map.update!(counters,
    operation, &(&1 + 1))
     end)
   end
  end
```

```
23
24  def multiply(matrix1, matrix2) do
25    if !is_matrix(matrix1) or !is_matrix(matrix2) do
26      IO.puts("Error: Matrices must be 2D lists of numbers.")
27      []
28    end
29    agent = reset_counters()
30    rows1 = length(matrix1)
31    cols1 = length(Enum.at(matrix1, 0))
32    rows2 = length(matrix2)
33    cols2 = length(Enum.at(matrix2, 0))
34
35    if cols1 != rows2 do
36      IO.puts("Error: Matrices cannot be multiplied. Number of
      columns in the first matrix must equal the number of rows in
      the second matrix.")
37      []
38    else
39      for i <- 0..rows1-1, do: for j <- 0..cols2-1, do: do_multiply
      (matrix1, matrix2, i, j, cols1, agent)
40    end
41  end
42
43  defp do_multiply(matrix1, matrix2, row, col, size, agent) do
44    result = Enum.reduce(0..size-1, 0, fn k, acc ->
45      update_counter(:multiplications, agent)
46      val1 = Enum.at(matrix1, row) |> Enum.at(k)
47      val2 = Enum.at(matrix2, k) |> Enum.at(col)
48      update_counter(:additions, agent)
49      acc + val1 * val2
50    end)
51    update_counter(:subtractions, agent)
52    result
53  end
54
55  def multiply_strassen(matrix1, matrix2, l) do
56    if !is_matrix(matrix1) or !is_matrix(matrix2) do
57      IO.puts("Error: Matrices must be 2D lists of numbers.")
58      []
59    end
60    agent = reset_counters()
61    size1 = length(matrix1)
62    size2 = length(matrix2)
63
64    # Perform Strassen multiplication
65    result = strassen_multiply(matrix1, matrix2, agent, l)
66
67    # Trim result to original size
68    {trim_matrix(result, size1), agent}
69  end
70
71  defp strassen_multiply(matrix1, matrix2, agent, l) when length(
      matrix1) == 1 do
72    update_counter(:multiplications, agent)
73    [[Enum.at(Enum.at(matrix1, 0), 0) * Enum.at(Enum.at(matrix2, 0)
      , 0)]]
74  end
```

```
75
76  defp strassen_multiply(matrix1, matrix2, agent, l) when length(
      matrix1) <= l do
77    if !is_matrix(matrix1) or !is_matrix(matrix2) do
78      IO.puts("Error: Matrices must be 2D lists of numbers.")
79      []
80    end
81    rows1 = length(matrix1)
82    cols1 = length(Enum.at(matrix1, 0))
83    rows2 = length(matrix2)
84    cols2 = length(Enum.at(matrix2, 0))
85
86    if cols1 != rows2 do
87      IO.puts("Error: Matrices cannot be multiplied. Number of
      columns in the first matrix must equal the number of rows in
      the second matrix.")
88      []
89    else
90      for i <- 0..rows1-1, do: for j <- 0..cols2-1, do: do_multiply
      (matrix1, matrix2, i, j, cols1, agent)
91    end
92  end
93
94  defp strassen_multiply(matrix1, matrix2, agent, l) do
95    size = length(matrix1)
96    mid = div(size, 2)
97
98    a11 = submatrix(matrix1, 0, 0, mid)
99    a12 = submatrix(matrix1, 0, mid, mid)
100   a21 = submatrix(matrix1, mid, 0, mid)
101   a22 = submatrix(matrix1, mid, mid, mid)
102
103   b11 = submatrix(matrix2, 0, 0, mid)
104   b12 = submatrix(matrix2, 0, mid, mid)
105   b21 = submatrix(matrix2, mid, 0, mid)
106   b22 = submatrix(matrix2, mid, mid, mid)
107
108   m1 = strassen_multiply(add_matrices(a11, a22, agent),
      add_matrices(b11, b22, agent), agent, l)
109   m2 = strassen_multiply(add_matrices(a21, a22, agent), b11,
      agent, l)
110   m3 = strassen_multiply(a11, subtract_matrices(b12, b22, agent),
       agent, l)
111   m4 = strassen_multiply(a22, subtract_matrices(b21, b11, agent),
       agent, l)
112   m5 = strassen_multiply(add_matrices(a11, a12, agent), b22,
      agent, l)
113   m6 = strassen_multiply(subtract_matrices(a21, a11, agent),
      add_matrices(b11, b12, agent), agent, l)
114   m7 = strassen_multiply(subtract_matrices(a12, a22, agent),
      add_matrices(b21, b22, agent), agent, l)
115
116   c11 = add_matrices(subtract_matrices(add_matrices(m1, m4, agent
      ), m5, agent), m7, agent)
117   c12 = add_matrices(m3, m5, agent)
118   c21 = add_matrices(m2, m4, agent)
119   c22 = add_matrices(subtract_matrices(add_matrices(m1, m3, agent
```

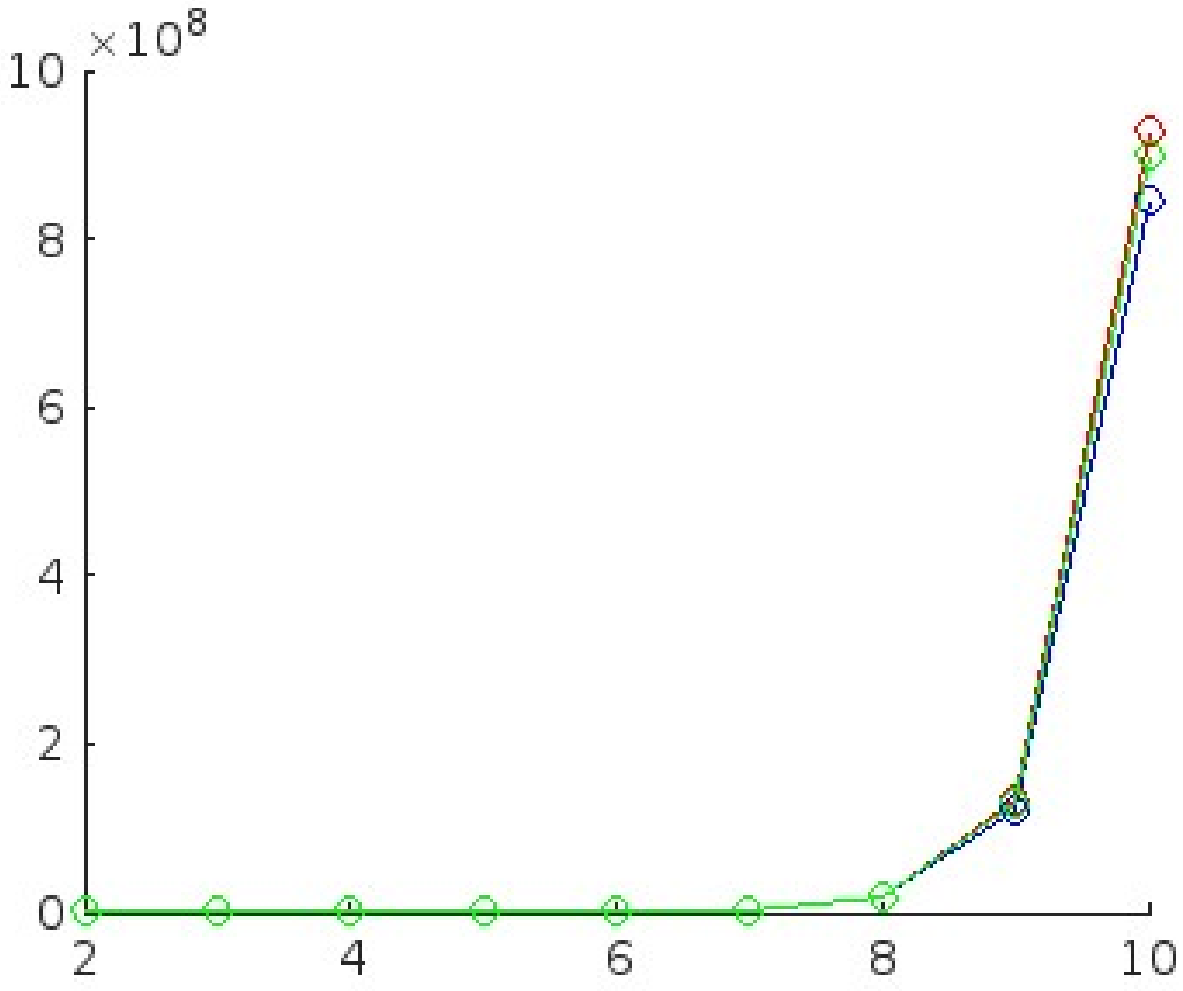
```
    ), m2, agent), m6, agent)

    combine_submatrices(c11, c12, c21, c22)
  end

  defp add_matrices(matrix1, matrix2, agent) do
    update_counter(:additions, agent)
    Enum.zip(matrix1, matrix2) |> Enum.map(fn {row1, row2} -> Enum.zip(row1, row2) |> Enum.map(fn {elem1, elem2} -> elem1 + elem2 end) end)
  end

  defp subtract_matrices(matrix1, matrix2, agent) do
    update_counter(:subtractions, agent)
    Enum.zip(matrix1, matrix2) |> Enum.map(fn {row1, row2} -> Enum.zip(row1, row2) |> Enum.map(fn {elem1, elem2} -> elem1 - elem2 end) end)
  end

  defp submatrix(matrix, row_start, col_start, size) do
    Enum.slice(matrix, row_start, size)
    |> Enum.map(&Enum.slice(&1, col_start, size))
  end

  defp combine_submatrices(c11, c12, c21, c22) do
    c_top = Enum.zip(c11, c12) |> Enum.map(fn {row1, row2} -> row1 ++ row2 end)
    c_bottom = Enum.zip(c21, c22) |> Enum.map(fn {row1, row2} -> row1 ++ row2 end)
    c_top ++ c_bottom
  end

  defp trim_matrix(matrix, size) do
    Enum.slice(matrix, 0, size) |> Enum.map(&Enum.slice(&1, 0, size))
  end

  defp is_matrix(matrix) do
    is_list(matrix) and
    length(matrix) > 0 and
    is_list(Enum.at(matrix, 0))
  end
end

# Example usage
matrix1 = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
]
matrix2 = [
  [9, 8, 7],
  [6, 5, 4],
  [3, 2, 1]
]

{result, agent} = Matrix.multiply_strassen(matrix1, matrix2, 1)
```

```
169 IO.inspect(Agent.get(agent, &(&1)))
```

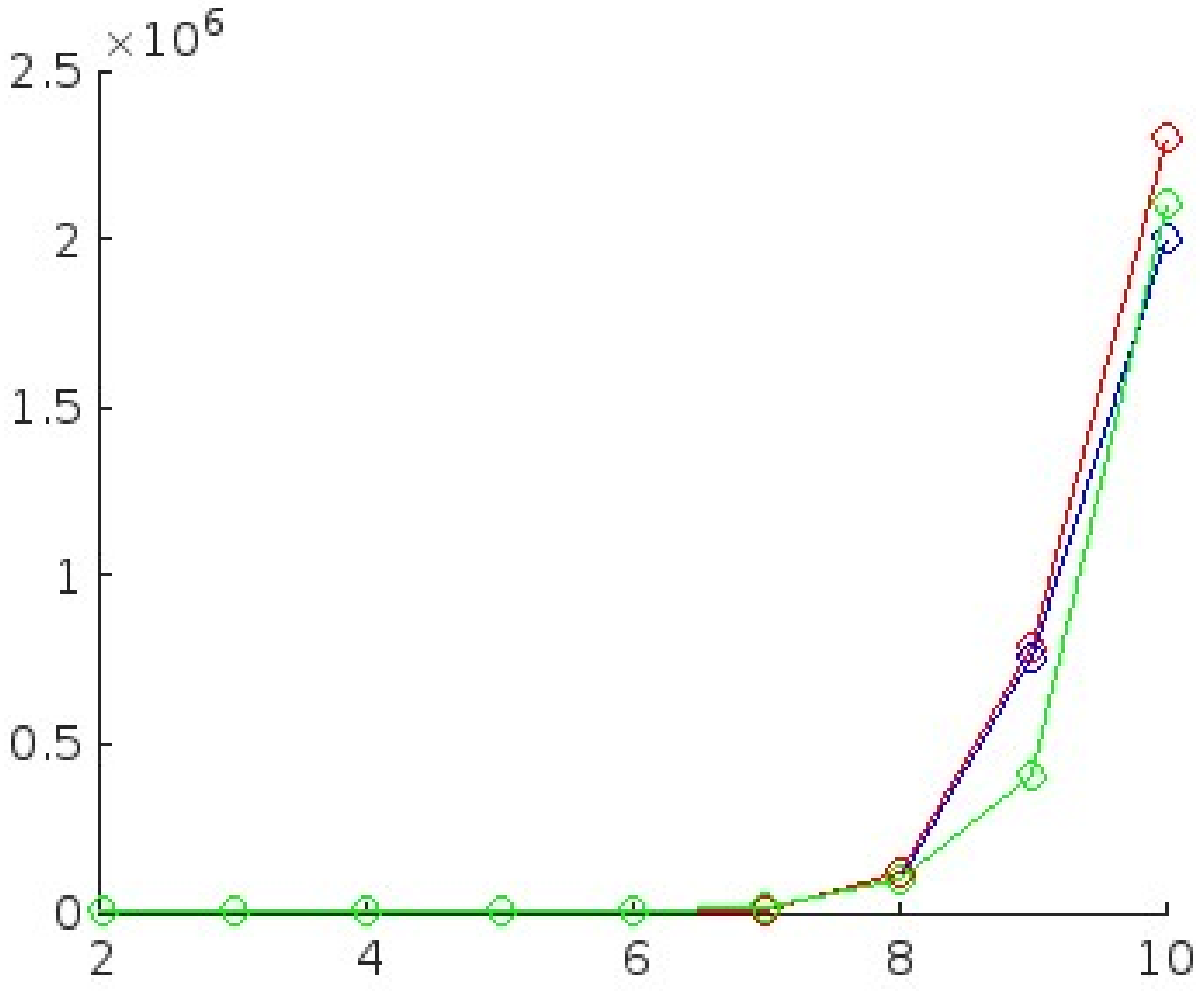
```
defmodule Computations do
  def main do
    ks = 2..7
    ls = [2, 5, 8]

    for k <- ks, do: (for l <- ls, do: compute(k, l))
  end

  def compute(k, l) do
    size = Integer.pow(2, k)
    matrix1 = for _ <- 1..size, do: (for _ <- 1..size, do: Enum.
    random(1..1_000))
    matrix2 = for _ <- 1..size, do: (for _ <- 1..size, do: Enum.
    random(1..1_000))
    start = System.os_time(:millisecond)
    {_, agent} = Matrix.multiply_strassen(matrix1, matrix2, l)
    fin = System.os_time(:millisecond)
    IO.inspect(Agent.get(agent, &(&1)))
    res = Agent.get(agent, &(&1).additions) + Agent.get(agent,
    &(&1).multiplications) + Agent.get(agent, &(&1).subtractions)
    IO.puts("#{res} ")
    File.write("results.csv", "#{res},#{fin-start}\n", [:append])
  end
end
```

2 Wyniki

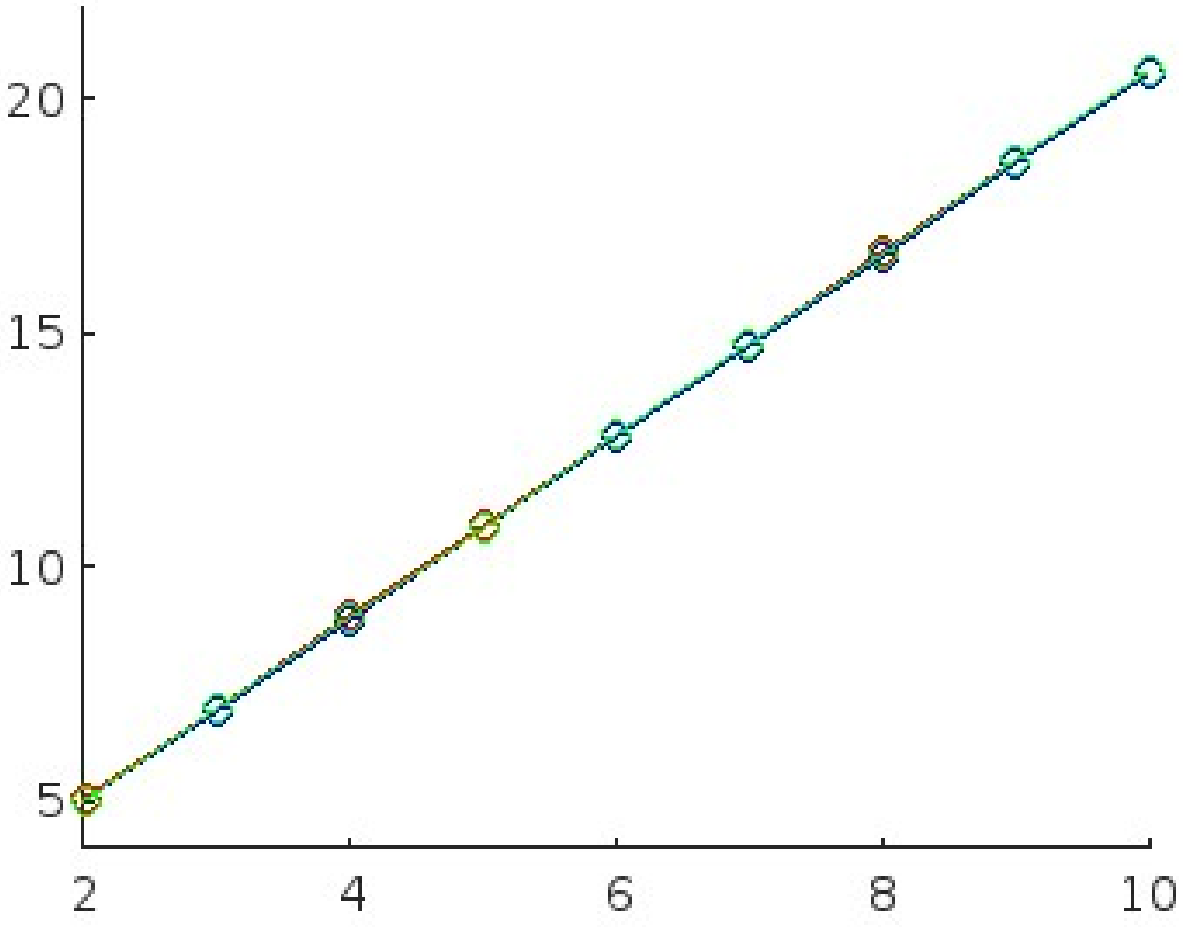
Poniższe wykresy obrazują wydajność algorytmu, mierzoną jako liczę operacji elementarnych potrzebnych do wykonania mnożenia oraz czas obliczeń, w zależności od rozmiaru macierzy. Każda z trzech linii obrazuje obliczenia dla innego progu zmiany algorytmu. Dla ułatwienia odczytywania danych zostały zamieszczone wykresy z wartościami na osi Y w skali logarytmicznej oraz wykresy pokazujące różnice w FLOPach i czasie pomiędzy obliczeniami z różnymi progami. Skróty w opisach wykresów oznaczają kolory linii: r - czerwony, g - zielony, b - niebieski, m - magenta, y - żółty



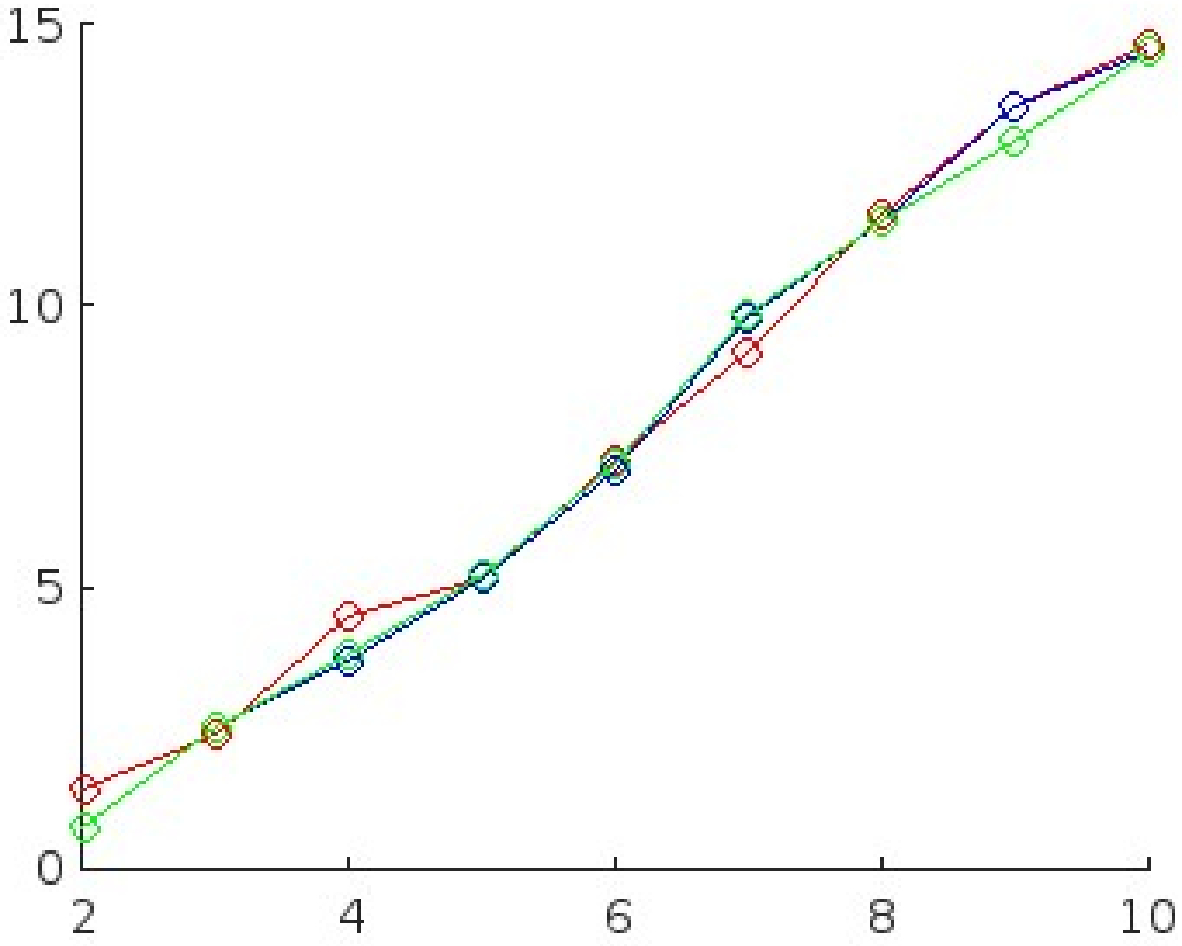
Rysunek 1: Ilość operacji zmiennoprzecinkowych w zależności od rozmiaru macierzy, dla progów 3, 5, 8 (rbg)



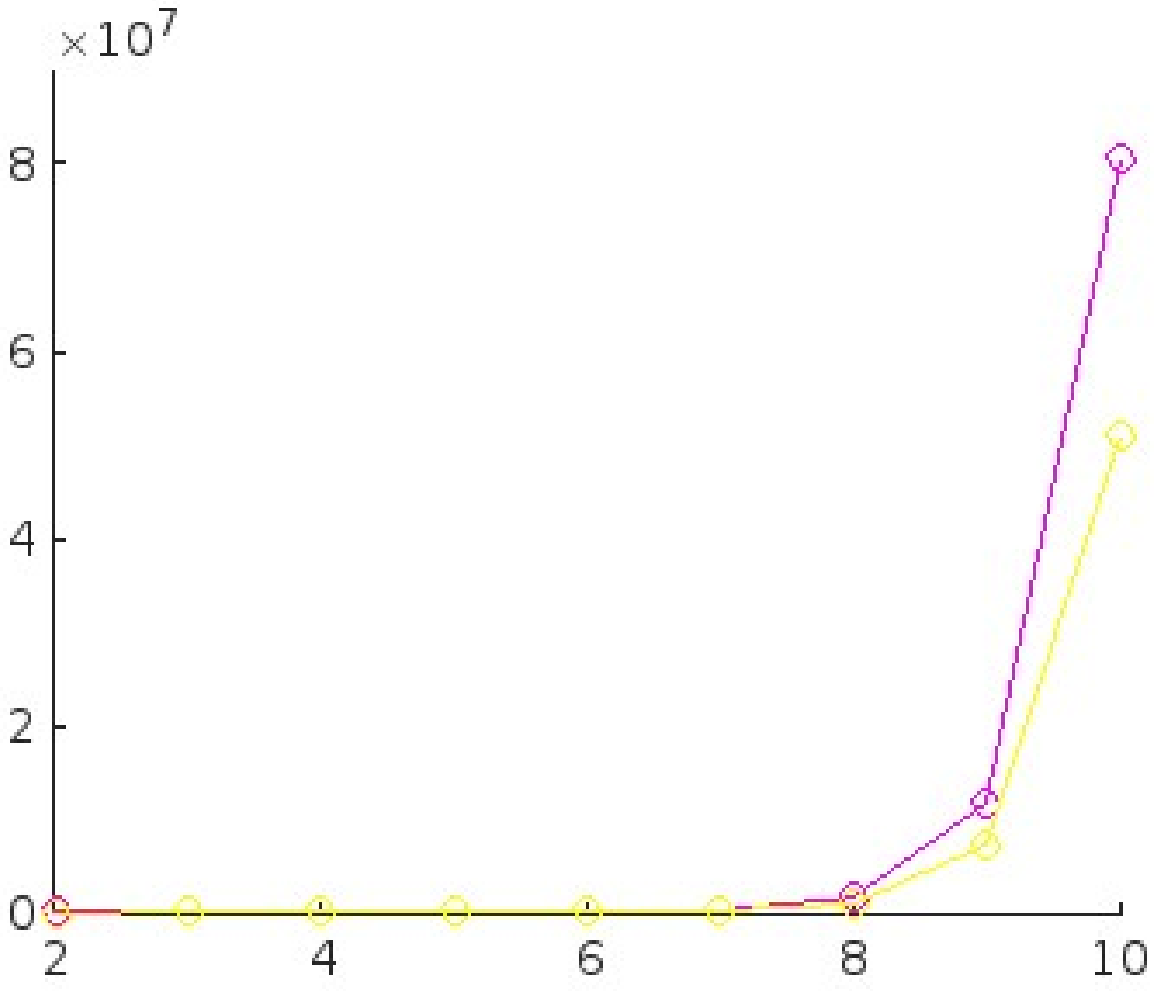
Rysunek 2: Czas wykonywania obliczeń (ms) w zależności od rozmiaru macierzy , dla progów 3, 5, 8 (rbg)



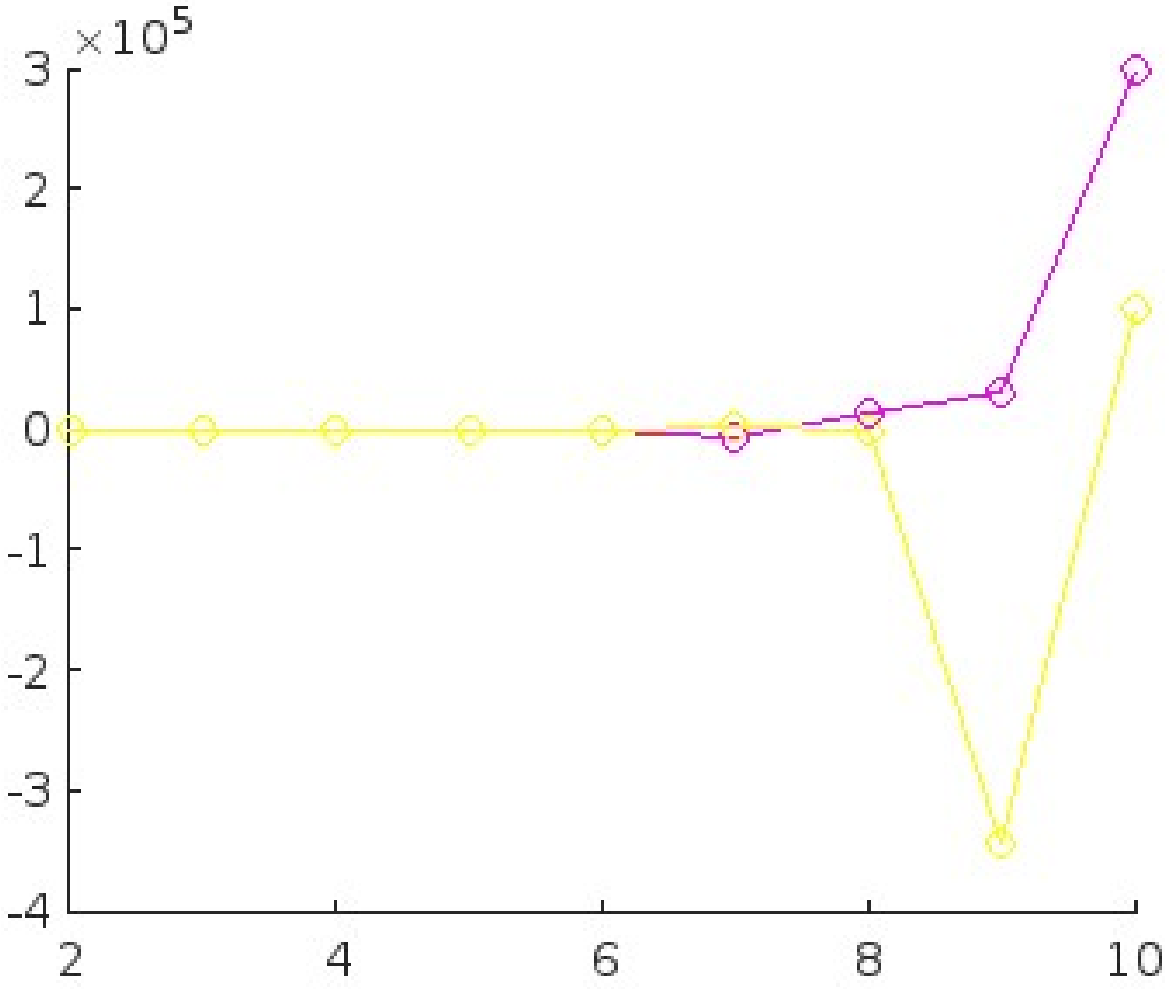
Rysunek 3: Ilość operacji zmiennoprzecinkowych w zależności od rozmiaru macierzy, dla progów 3, 5, 8 (rbg) (skala logarytmiczna)



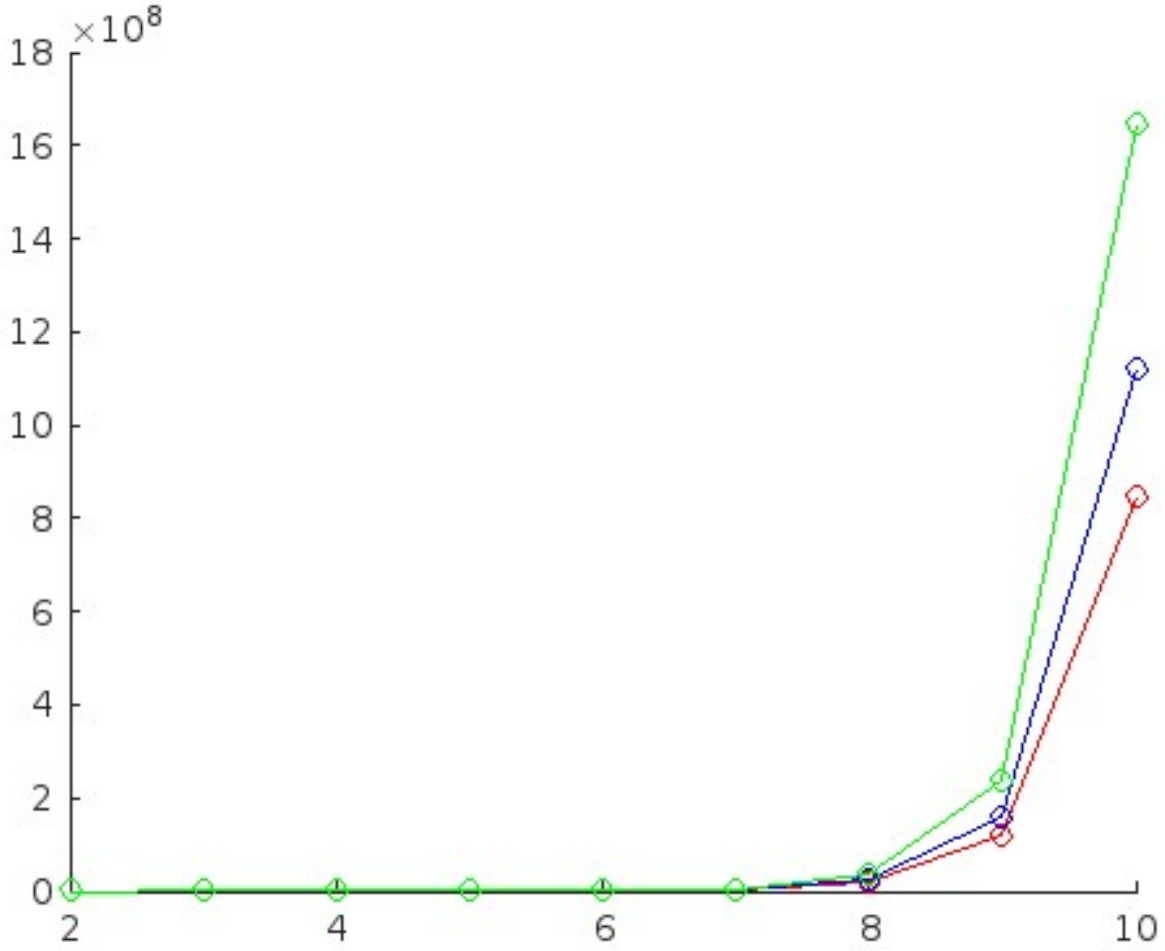
Rysunek 4: Czas wykonywania obliczeń (ms) w zależności od rozmiaru macierzy, dla progów 3, 5, 8 (rbg) (skala logarytmiczna)



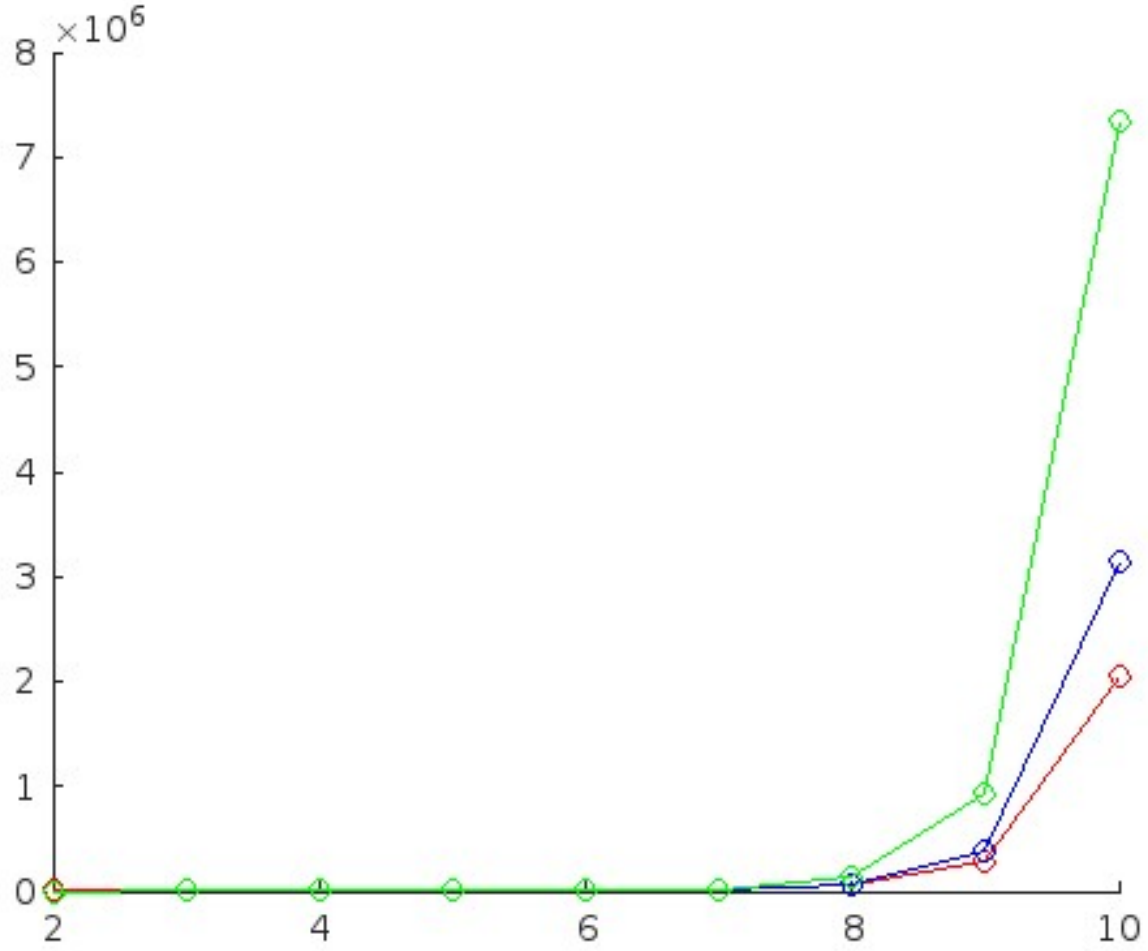
Rysunek 5: Różnica pomiędzy ilością operacji zmiennoprzecinkowych w zależności od rozmiaru macierzy, pomiędzy progami 5 i 3 oraz 8 i 3 (my)



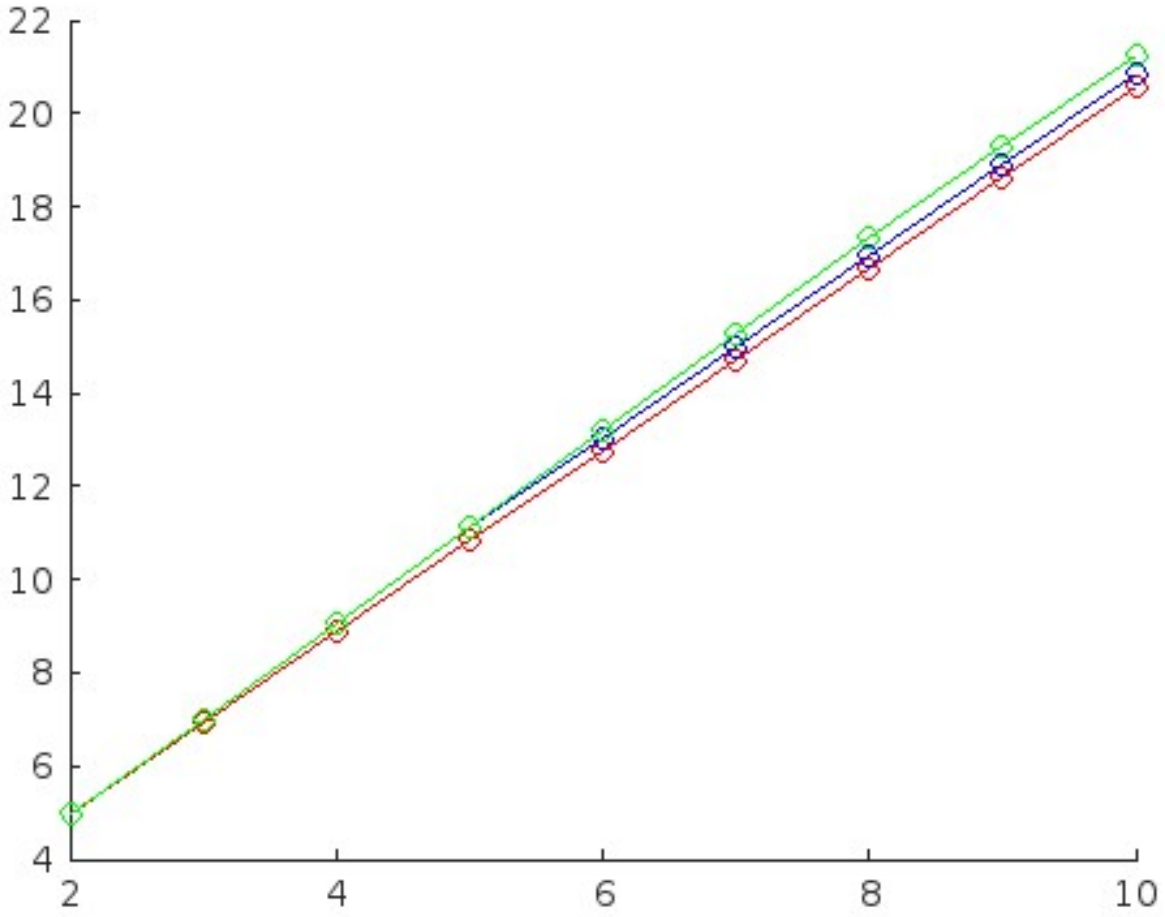
Rysunek 6: Różnica pomiędzy czasem wykonywania obliczeń (ms) w zależności od rozmiaru macierzy, pomiędzy progami 5 i 3 oraz 8 i 3 (my)



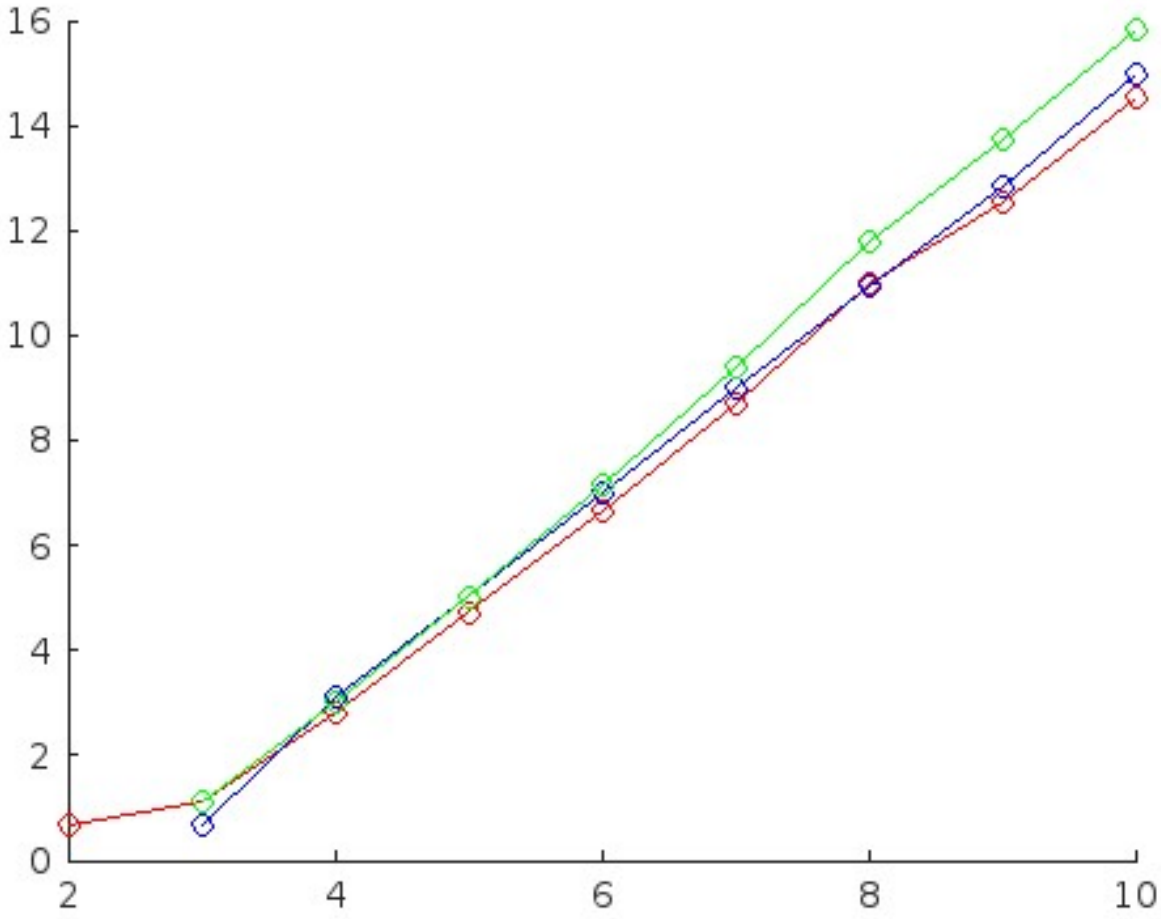
Rysunek 7: Ilość operacji zmiennoprzecinkowych w zależności od rozmiaru macierzy, dla progów 4, 32, 256 (rbg)



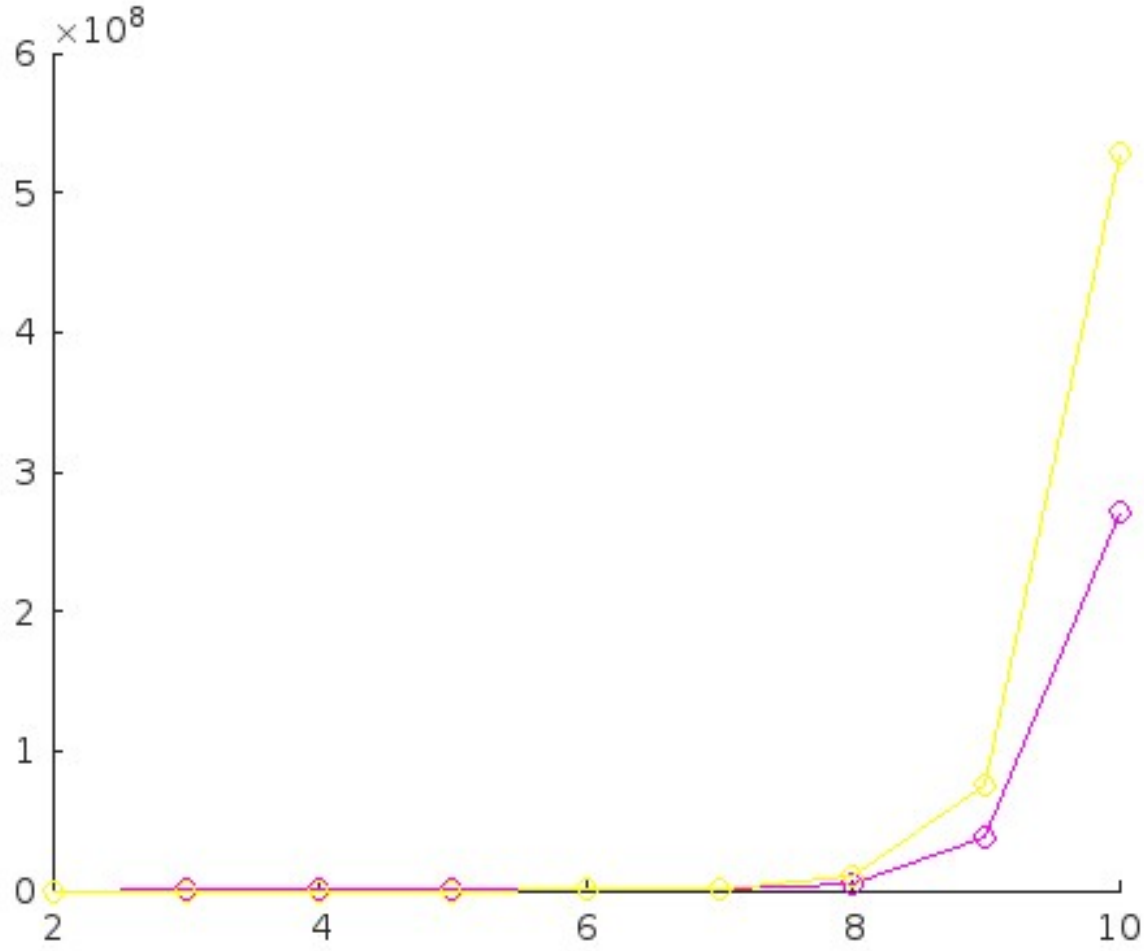
Rysunek 8: Czas wykonywania obliczeń (ms) w zależności od rozmiaru macierzy, dla progów 4, 32, 256 (rbg)



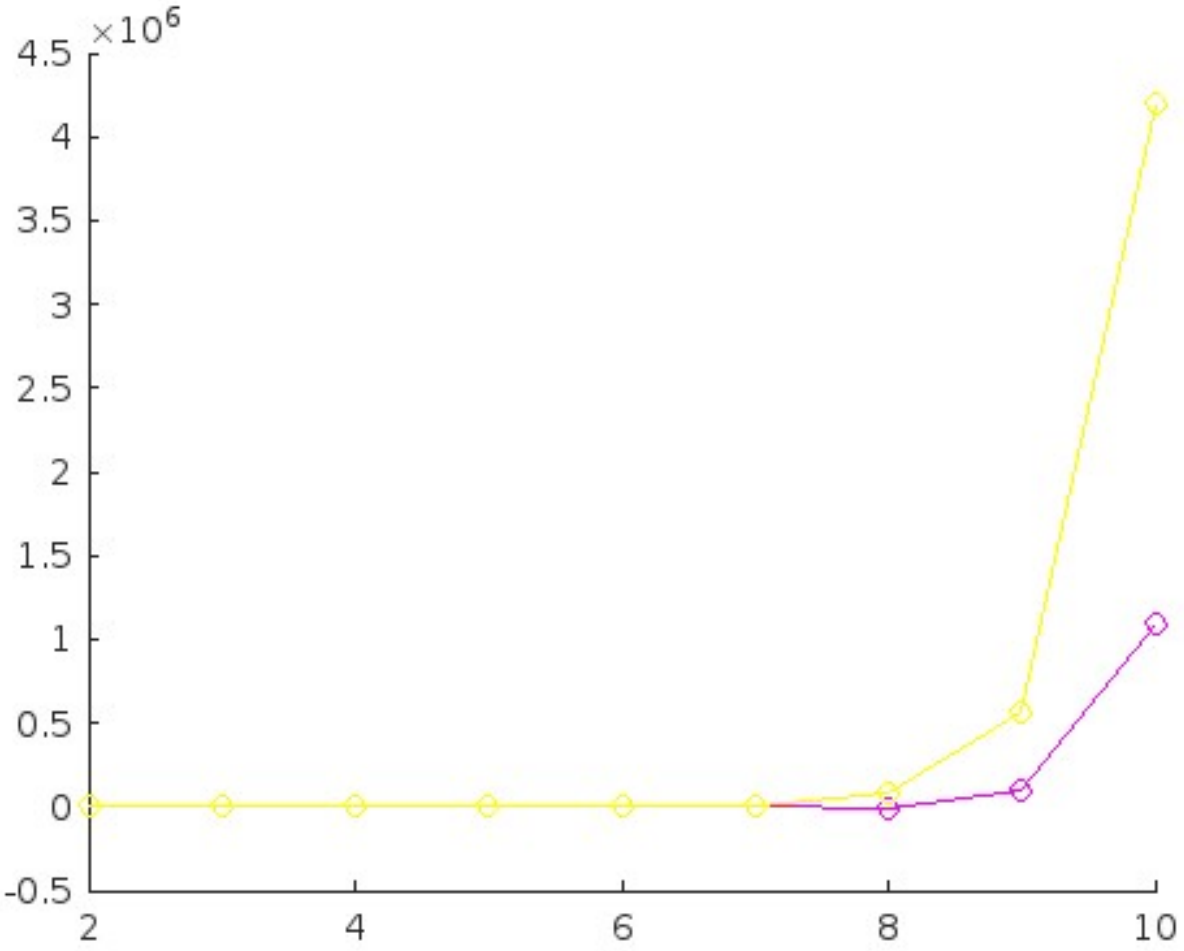
Rysunek 9: Ilość operacji zmiennoprzecinkowych w zależności od rozmiaru macierzy, dla progów 4, 32, 256 (rbg) (skala logarytmiczna)



Rysunek 10: Czas wykonywania obliczeń (ms) w zależności od rozmiaru macierzy, dla progów 4, 32, 256 (rbg) (skala logarytmiczna)



Rysunek 11: Różnica pomiędzy ilością operacji zmiennoprzecinkowych w zależności od rozmiaru macierzy, pomiędzy progami 32 i 4 oraz 256 i 4 (my)



Rysunek 12: Różnica pomiędzy czasem wykonywania obliczeń (ms) w zależności od rozmiaru macierzy, pomiędzy progami 32 i 4 oraz 256 i 4 (my)

3 Wnioski

- Różnica pomiędzy poszczególnymi sposobami przeprowadzenia obliczeń jest nieznaczna dla macierzy o rozmiarach mniejszych niż 2^8.
- Zastosowanie progu zmiany algorytmu Strassena na klasyczny wydaje się być bezzasadne, gdyż pogorsza złożoność obliczeniową i czasową.
- Na wykresach czasu obliczeń (szczególnie wykresach różnic) pojawiają się podejrzane anomalie, które najpewniej wynikają z faktu istnienia innych procesów, działających w czasie obliczeń.