OBLICZENIA NAUKOWE: Lista nr 5

Środa, 3 stycznia 2018

Tymoteusz Surynt

Numer indeksu: 229794

Podsumowanie

Zadanie 1

Opis problemu

Stworzenie modułu, dzięki któremu będzie można skorzystać ze struktury, która pozwoli efektywnie prowadzić obliczenia na blokowej macierzy A z dużą ilością elementów (e.g. 10000×10000, 50000×50000)

Opis rozwiązania

Na początku warto zauważyć, że macierz A jest wstęgową macierzą blokową. Z tego faktu wiemy, że niezerowe wartości znajdują się na przekątnej i zaraz obok przekątnej macierzy. Łatwo też da się zauważyć, że maksymalna liczba niezerowych bloków macierzy w jednym wierszu wynosi 3. Mając taką wiedzę można łatwo zbudować tablicę, która w każdej komórce będzie pamiętać 3 bloki z wartościami. Łatwo zauważyć, że w pierwszej i ostatniej komórce powinny być tylko 2 bloki, ale tę niedogodność łatwo obejść przez inicjonowanie komórek 1 i 3 (B i C) jako zera.
Każdy blok zapamiętywany w tablicy ma swoje właściwości pozwalające mu zapamiętać tylko najważniejsze dane:

1. komórka zapamiętująca blok B_k. Jest ona macierzą o wysokości l (wielkość pojedynczego bloku) oraz o szerokości 2. Taka konstrukcja wynika z faktu, że tylko dwie ostatnie wartości w tym bloku mogą mieć inne wartości niż 0, więc nie ma sensu pamiętania wszystkich

2. komórka zapamiętująca blok A_k. Jest ona macierzą $l \times l$. Niestety w tym bloku każda wartość może być niezerowa i na całości bloku będą prawdopodobnie prowadzone obliczenia dlatego nie ma dużego zysku w próbie optymalizacji miejsca zajmowanego przez ten blok

3. komórka zapamiętująca blok C_k. Jest ona macierzą $l \times l$ wypełnioną na początku samymi zerami. Tutaj celowo nie optymalizuje miejsca zajmowanego przez macierz, mimo że wartości niezerowe znajdują się tylko na przekątnej macierzy. Takie działanie wynika z faktu, że przy liczeniu eliminacji Gaussa z i bez częściowego wyboru elementu głównego korzystam z pozostałych komórek do obliczeń

4. przy obliczaniu wektora metodą eliminacji Gaussa z częściowym wyborem elementu głównego występuje tymczasowa komórka, w której zapisywane są odpowiednie liczby w razie, gdyby nastąpiła zamiana wiersza między blokami

Funkcje wchodzące w skład modułu blocksys wraz z krótkim opisem i przykładami użycia:

- $importMatrix(fileLocation)$
  funkcja odpowiedzialna za importowanie macierzy z pliku. Jej jedynym parametrem jest ścieżka do pliku. Funkcja zwraca rozmiar całej macierzy n, rozmiar pojedynczego bloku l oraz samą tablicę reprezentującą macierz blokową A w postaci opisanej powyżej.
  Przykład użycia:

$$result = importMatrix(\text{"}/home/user/Desktop/ON/Dane16\_1\_1/A.txt\text{"})$$

- $getError(x, value :: Float64, n :: Int64)$
  funkcja odpowiedzialna za wyliczanie średniego błędu bezwzględnego wektora, jeśli każda jego wartość powinna przyjmować jedną wartość(e.g. 1.0). Funkcja jako parametry przyjmuje wektor x, wartość jaką powinny przyjmować komórki wektora ($value$) oraz wielkość wektora n, natomiast zwraca wartość w typie Float64 reprezentującą średni błąd bezwzględny.
  Przykład użycia:

$$err = getError(x, 1.0, 16)$$

- $exportVectorNoError(fileLocation, x)$
  funkcja odpowiedzialna za eksportowanie wektora do zadanego pliku bez podawania błędu na początku pliku. Funkcja jako parametry przyjmuje ścieżkę do pliku oraz wektor, który chcemy zapisać
  Przykład użycia:

$$exportVectorNoError("/home/user/Desktop/ON/Tests/x.txt", x)$$

- $exportVectorError(fileLocation, x, err)$
  funkcja odpowiedzialna za eksportowanie wektora do zadanego pliku z podaniem na początku pliku błędu. Funkcja jako parametry przyjmuje ścieżkę do pliku, wektor oraz błąd.
  Przykład użycia:

$$exportVectorNoError("/home/user/Desktop/ON/Tests/x.txt", x, err)$$

- $importVector(fileLocation)$
  funkcja odpowiedzialna za importowanie wektora z pliku. Jej jedynym parametrem jest ścieżka do pliku. Funkcja zwraca rozmiar wektora n oraz wektor w postaci tablicy z elementami typu Float64.
  Przykład użycia:

$$b = importVector("/home/user/Desktop/ON/Dane16\_1\_1/b.txt")$$

- $printfMatrix(A, b, l :: Int64, n :: Int64)$ funkcja odpowiedzialna za wyświetlanie komórek tablicy, omówionej na samym początku tego zagadnienia. Wyświetla tylko komórki w których potencjalnie mogą być niezerowe wartości. Funkcja przyjmuje jako parametry macierz A, wektor b, wielkość pojedynczego bloku oraz wielkość całej macierzy.
  Przykład użycia:

$$printfMatrix(A, b, 4, 16)$$

- $printfMatrix2(A, b, l :: Int64, n :: Int64)$
  funkcja odpowiedzialna za wyświetlanie komórek tablicy, omówionej na samym początku tego zagadnienia z dodatkowym 4 blokiem występującym przy korzystaniu z eliminacji Gaussa z częściowym wyborem elementu głównego. Wyświetla tylko komórki w których potencjalnie mogą być niezerowe wartości. Funkcja przyjmuje jako parametry macierz A, wektor b, wielkość pojedynczego bloku oraz wielkość całej macierzy.
  Przykład użycia:

$$printfMatrix2(A, b, 4, 16)$$

- $gaussElimination(n :: Int64, l :: Int64, AB, v)$
  funkcja odpowiedzialna za wyliczenie wektora x metodą eliminacji Gaussa bez wyboru elementu głównego. Funkcja jako parametry przyjmuje wielkość macierzy A, długość pojedynczego bloku, macierz A w postacie opisanej na początku tego podpunktu oraz wektor b (w tym przypadku nazywa się v, później w miarę obliczeń zmieni nazwę na właściwą, czyli b). Funkcja zwraca jako wynik czwórkę, której pierwszą składową jest nowa macierz A będąca wynikiem pierwszej części eliminacji Gaussa, druga składowa to nowy wektor b, który również powstał przez eliminację Gaussa, trzecia składowa to wektor x, natomiast ostatnia składowa przyjmuje wartość 0, jeśli obliczanie przeszło bez większego problemu oraz wartość 1, jeśli element na przekątnej był zbliżony do 0.
  Przykład użycia:

$$w = gaussElimination(16, 4, A, b)$$

- $gaussElimination2(n :: Int64, l :: Int64, AB, v)$
  funkcja odpowiedzialna za wyliczenie wektora x metodą eliminacji Gaussa z wyborem elementu głównego. Funkcja jako parametry przyjmuje wielkość macierzy A, długość pojedynczego bloku, macierz

A w postacie opisanej na początku tego podpunktu z dodatkowym blokiem na obliczenia w każdym wierszu oraz wektor b (w tym przypadku nazywa się v, później w miarę obliczeń zmieni nazwę na właściwą, czyli b). Funkcja zwraca jako wynik czwórkę, której pierwszą składową jest nowa macierz A będąca wynikiem pierwszej części eliminacji Gaussa, druga składowa to nowy wektor b, który również powstał przez eliminację Gaussa, trzecia składowa to wektor x, natomiast ostatnia składowa przyjmuje wartość 0, jeśli obliczanie przeszło bez większego problemu oraz wartość 1, jeśli element na przekątnej był zbliżony do 0.
Przykład użycia:

$$w = gaussElimination(16, 4, A, b)$$

Testy

Testowanie obu eliminacji Gaussa będzie omawiane przy następnym zadaniu, w tym podpunkcie są przedstawione testy pozostałych funkcji. Aby przetestować powyższe funkcje posłużyłem się danymi przykładowymi, które można znaleźć na stronie pana prof. Zielińskiego ($http://cs.pwr.edu.pl/zielinski/$) w zakładce obliczenia naukowe, lista nr 5. Do uzyskania czasu posłużyłem się programem linuxowym time.

- Importowanie macierzy:
  Czas dla 16 elementów:

real	0m 0.396s
user	0m 0.354s
sys	0m 0.040s

  Czas dla 10000 elementów:

real	0m 0.457s
user	0m 0.417s
sys	0m 0.038s

  Czas dla 50000 elementów:

real	0m 0.848s
user	0m 0.818s
sys	0m 0.054s

  Poprawność importowanych danych będzie testowana przy użyciu funkcji $printfMatrix$.

- Importowanie wektora:
  Czas dla 16 elementów:

real	0m 0.357s
user	0m 0.316s
sys	0m 0.039s

  Czas dla 10000 elementów:

real	0m 0.360s
user	0m 0.323s
sys	0m 0.036s

  Czas dla 50000 elementów:

real	0m 0.369s
user	0m 0.340s
sys	0m 0.026s

Poprawność importowanych danych będzie testowana przy użyciu funkcji $printfMatrix$.

- Wyświetlanie macierzy:
  Przykład dla 16 elementów:

```
-------------------------------------------------------------------------------------------------------------
|      0.00     0.00 |    -2.09    -2.21     4.77    -1.43 |     0.13     0.00     0.00     0.00 |    -0.82 |
|      0.00     0.00 |     3.65    -3.69     2.70    -3.06 |     0.00     0.18     0.00     0.00 |    -0.22 |
|      0.00     0.00 |    -0.57     5.73     0.32    -4.68 |     0.00     0.00     0.28     0.00 |     1.08 |
|      0.00     0.00 |     0.94    -1.22    -4.35     8.11 |     0.00     0.00     0.00     0.20 |     3.68 |
-------------------------------------------------------------------------------------------------------------
|      0.20     0.29 |    -3.85     6.56    -0.21     0.77 |     0.18     0.00     0.00     0.00 |     3.93 |
|      0.25     0.28 |     0.84    -4.36    -0.90     3.96 |     0.00     0.25     0.00     0.00 |     0.32 |
|      0.05     0.19 |    -3.23    -0.16     5.17     1.18 |     0.00     0.00     0.01     0.00 |     3.20 |
|      0.13     0.07 |     7.25    -0.98    -3.01    -4.91 |     0.00     0.00     0.00     0.25 |    -1.21 |
-------------------------------------------------------------------------------------------------------------
|      0.03     0.22 |     2.60    -1.05    -2.90     0.86 |     0.02     0.00     0.00     0.00 |    -0.23 |
|      0.23     0.03 |    -4.77     7.85    -4.97     0.52 |     0.00     0.12     0.00     0.00 |    -0.98 |
|      0.30     0.11 |     3.69    -1.01     3.63    -4.32 |     0.00     0.00     0.24     0.00 |     2.62 |
|      0.14     0.05 |    -2.68    -2.10     1.73     5.70 |     0.00     0.00     0.00     0.14 |     2.98 |
-------------------------------------------------------------------------------------------------------------
|      0.17     0.30 |     1.52    -0.21    -8.20     2.43 |     0.00     0.00     0.00     0.00 |    -4.00 |
|      0.13     0.04 |    -3.34     1.20    -0.10     2.62 |     0.00     0.00     0.00     0.00 |     0.55 |
|      0.04     0.22 |     1.43     4.63     3.01    -6.52 |     0.00     0.00     0.00     0.00 |     2.82 |
|      0.01     0.21 |     1.14    -2.74     6.35     0.83 |     0.00     0.00     0.00     0.00 |     5.80 |
-------------------------------------------------------------------------------------------------------------
```

  Czas dla 16 elementów:

real	0m 0.610s
user	0m 0.565s
sys	0m 0.040s

  Czas dla 10000 elementów:

real	0m 2.420s
user	0m 1.346s
sys	0m 1.058s

  Czas dla 50000 elementów:

real	0m 9.229s
user	0m 4.048s
sys	0m 5.127s

  Dla przykładów z 10000 i 50000 nie ma obrazka, gdyż zajął by on dużo miejsca i byłby nieczytelny. Nie zmienia to jednak faktu, że wartości zgadzają się z tymi znajdującymi się w plikach. Warto również zauważyć, że czas jest podany razem z czasem potrzebnym na wczytanie wektora oraz macierzy, aby uzyskać czas samego wyświetlania można odjąć podane czas i te znajdujące się wyżej.

Zadanie 2

Opis problemu

Zadanie polegało na stworzeniu algorytmu, który efektywnie liczy metodą eliminacji Gaussa wektor x z równania $Ax = b$:

a) bez wyboru elementu głównego

b) z wyborem elementu głównego

Eliminacja Gaussa bez wyboru elementu głównego:

Dane:
n - wielkość macierzy A
l - wielkość pojedynczego bloku
AB - macierz A
v - wektor b
Wynik: (A,b,x,0|1) – co zwracają składowe wyniku zostało wyjaśnione na stronie 4.
Funkcja: gaussElimination(n::Int64,l::Int64,AB,v)

```
A ← deepcopy(AB);
b ← deepcopy(v);
X ← ArrayFloat64(n);
for i in 1:(l-1) do
    for j in (i+1):l do
        if abs(A[1,2][i,i]) < ϵ then
            return (A,b,X,1)
        end
        m ← A[1,2][j,i]/A[1,2][i,i];
        for k in i:l do
            A[1,2][j,k] ← A[1,2][j,k] − m ∗ A[1,2][i,k];
        end
        for k in 1:l do
            A[1,3][j,k] ← A[1,3][j,k] − m ∗ A[1,3][i,k];
        end
        b[j] ← b[j] − m ∗ b[i];
    end
end
```

```
for x in 2 : (n/l − 1) do
    for j in 1: l do
        if abs(A[x − 1, 2][l − 1, l − 1]) < ϵ then
            return (A,b,X,1)
        end
        m ← A[x, 1][j, 1]/A[x − 1, 2][l − 1, l − 1];
        A[x, 1][j, 2] ← A[x, 1][j, 2] − m ∗ A[x − 1, 2][l − 1, l];
        A[x, 1][j, 1] ← A[x, 1][j, 1] − m ∗ A[x − 1, 2][l − 1, l − 1];
        for k in 1:l do
            A[x, 2][j, k] ← A[x, 2][j, k] − m ∗ A[x − 1, 3][l − 1, k];
        end
        b[(x − 1) ∗ l + j] ← b[(x − 1) ∗ l + j] − m ∗ b[(x − 2) ∗ l + l − 1];
    end
    for j in 1: l do
        if abs(A[x − 1, 2][l, l]) < ϵ then
            return (A,b,X,1)
        end
        m ← A[x, 1][j, 2]/A[x − 1, 2][l, l];
        A[x, 1][j, 2] ← A[x, 1][j, 2] − m ∗ A[x − 1, 2][l, l];
        for k in 1:l do
            A[x, 2][j, k] ← A[x, 2][j, k] − m ∗ A[x − 1, 3][l, k];
        end
        b[(x − 1) ∗ l + j] ← b[(x − 1) ∗ l + j] − m ∗ b[(x − 2) ∗ l + l];
    end
    for i in 1:(l-1) do
        for j in (i+1):l do
            if abs(A[x, 2][i, i]) < ϵ then
                return (A,b,X,1)
            end
            m ← A[x, 2][j, i]/A[x, 2][i, i];
            for k in i:l do
                A[x, 2][j, k] ← A[x, 2][j, k] − m ∗ A[x, 2][i, k];
            end
            for k in 1:l do
                A[x, 3][j, k] ← A[x, 3][j, k] − m ∗ A[x, 3][i, k];
            end
            b[(x − 1) ∗ l + j] ← b[(x − 1) ∗ l + j] − m ∗ b[(x − 1) ∗ l + i];
        end
    end
end
```

for *j in 1: l* **do**
  if $abs(A[n/l-1,2][l-1,l-1]) < \epsilon$ **then**
    return *(A,b,X,1)*
  end
  $m \leftarrow A[n/l,1][j,1]/A[n/l-1,2][l-1,l-1]$;
  $A[n/l,1][j,1] \leftarrow A[n/l,1][j,1] - m * A[n/l-1,2][l-1,l-1]$;
  $A[n/l,1][j,2] \leftarrow A[n/l,1][j,2] - m * A[n/l-1,2][l-1,l]$;
  for *k in 1:l* **do**
    $A[n/l,2][j,k] \leftarrow A[n/l,2][j,k] - m * A[n/l-1,3][l-1,k]$;
  end
  $b[n-l+j] \leftarrow b[n-l+j] - m * b[n-l-1]$;
end
for *j in 1: l* **do**
  if $abs(A[n/l-1,2][l,l]) < \epsilon$ **then**
    return *(A,b,X,1)*
  end
  $m \leftarrow A[n/l,1][j,2]/A[n/l-1,2][l,l]$;
  $A[n/l,1][j,2] = A[n/l,1][j,2] - m * A[n/l-1,2][l,l]$;
  for *k in 1:l* **do**
    $A[n/l,2][j,k] = A[n/l,2][j,k] - m * A[n/l-1,3][l,k]$;
  end
  $b[n-l+j] \leftarrow b[n-l+j] - m * b[n-l]$;
end
for *i in 1:(l-1)* **do**
  for *j in (i+1):l* **do**
    if $abs(A[n/l,2][i,i]) < \epsilon$ **then**
      return *(A,b,X,1)*
    end
    $m \leftarrow A[n/l,2][j,i]/A[n/l,2][i,i]$;
    for *k in i:l* **do**
      $A[n/l,2][j,k] \leftarrow A[n/l,2][j,k] - m * A[n/l,2][i,k]$;
    end
    $b[n-l+j] \leftarrow b[n-l+j] - m * b[n-l+i]$;
  end
end
$it \leftarrow n$;
for *i in l:-1:1* **do**
  $s \leftarrow b[it]$;
  if $abs(A[n/l,2][i,i]) < \epsilon$ **then**
    return *(A,b,X,1)*
  end
  for *j in l:-1:(i+1)* **do**
    $s \leftarrow s - A[trunc(Int64, n/l),2][i,j] * X[n-l+j]$;
  end
  $X[it] \leftarrow s/A[trunc(Int64, n/l),2][i,i]$;
  $it \leftarrow it - 1$;
end

```
offsetA ← n − 2 ∗ l;
offsetC ← n − l;
for i in n/l − 1:-1:1 do
    for j in l:-1:1 do
        s ← b[it];
        if abs(A[i, 2][j, j]) < ϵ then
            return (A,b,X,1)
        end
        for k in l:-1:(j+1) do
            s ← s − A[i, 2][j, k] ∗ X[offsetA + k];
        end
        for k in j:-1:1 do
            s ← s − A[i, 3][j, k] ∗ X[offsetC + k];
        end
        X[it] ← s/A[i, 2][j, j];
        it ← it − 1;
    end
    offsetA ← offsetA − l;
    offsetC ← offsetC − l;
end
return (A,b,X,0)
```

Algorithm 1: Algorytm eliminacji Gaussa bez wyboru elementu głównego

Wyżej przedstawiony algorytm działa w myśl algorytmu eliminacji Gaussa tj. w pierwszej kolejności znajduje element na przekątnej macierzy A (te elementy zawsze znajdują się w bloku A_k). A następnie znajduje stosunek elementu głównego do kolejnych niezerowych wierszy A i odejmuje od każdego z elementów, wiersza za równo w bloku A_k jaki i C_k oraz wektora b, jego odpowiednik, pomnożony razy wyliczony stosunek, z wiersza z elementem głównym. Na początku kolejnego przebiegu, w sposób opisany wcześniej zeruje obie kolumny bloku B_k. Po skończeniu dostanie macierz górno trójkątną z której korzystając ze wzoru: $x_i = \frac{b''_i - a''_{i,n}x_n - \ldots - a''_{i,i+1}x_{i+1}}{a''_{i,i}}$, dla i=n, n-1, ...
można łatwo policzyć wektor x. Złożoność obliczeniowa:

$$O(l^2) + O(\frac{n}{l} * (l + l + l^2) + O(l + l + l^2) + O(l) + O(\frac{n}{l} * (l + l))$$

jako, że l jest stałą, to:

$$O(n) + O(n) = O(n)$$

Eliminacja Gaussa z wyborem elementu głównego:

Algorytm użyty do metody eliminacji Gaussa z wyborem elementu głównego jest bardzo podobny do algorytmu opisanego powyżej, więc ominę pseudokod. Główna różnica polega na tym, że zanim wybierzemy element na przekątnej, sprawdzamy inne elementy "pod" nim (czyli niezerowe wartości bloku A_k oraz czasami elementy bloku B_{k+1}) w celu znalezienia elementu którego wartość bezwzględna będzie największa. Celem tego zabiegu jest zminimalizowanie błędu powstającego podczas dzielenia. Kolejną różnicą jest moment wyliczenia bloku $B_k + 1$. Jest on redukowany zaraz przy wykonywaniu obliczeń na dwóch ostatnich kolumnach bloku A_k. Ta zmiana jest wymuszona możliwością złej zamiany wierszy przez co w miejscu gdzie powinny już być same 0 pojawi się nie zerowa wartość. Samo tworzenie macierzy górno trójkątnej czy liczenie wektora x przebiega dokładnie na tej samej zasadzie co w wcześniej wymienionym algorytmie eliminacji Gaussa bez

wyboru elementu głównego.
Złożoność obliczeniowa:

$$O(n)$$

ponieważ n występuje tylko w sumie z innym n, a l jest stałą

Testy dołączone do zadania

Czasy oraz błędy względne (jako, że wartość jest równa 1.0 to i bezwzględne) dla wybranych testów z danych testowych pochodzących ze strony internetowej pana prof. Zielińskiego (http://cs.pwr.edu.pl/zielinski/), zakładka Obliczenia Naukowe, lista nr 5.

1. Dla macierzy mającej 16 elementów:

   (a) Eliminacja Gaussa bez wyboru elementu głównego
   Czas:

real	0m 0.832s
user	0m 0.781s
sys	0m 0.044s

   Błąd:

   $$8.534839501805891e^{-16}$$

   (b) Eliminacja Gaussa z wyborem elementu głównego
   Czas:

real	0m 1.084s
user	0m 1.020s
sys	0m 0.049s

   Błąd:

   $$1.734723475976807e^{-16}$$

2. Dla macierzy mającej 10000 elementów:

   (a) Eliminacja Gaussa bez wyboru elementu głównego
   Czas:

real	0m 1.035s
user	0m 0.971s
sys	0m 0.056s

   Błąd:

   $$2.565070378324208e^{-15}$$

   (b) Eliminacja Gaussa z wyborem elementu głównego
   Czas:

real	0m 1.297s
user	0m 1.215s
sys	0m 0.070s

Błąd:

$$3.9874770152437124e^{-16}$$

3. Dla macierzy mającej 50000 elementów:

   (a) Eliminacja Gaussa bez wyboru elementu głównego
   Czas:

real	0m 1.523s
user	0m 1.439s
sys	0m 0.073s

   Błąd:

$$7.269969071188598^{-15}$$

   (b) Eliminacja Gaussa z wyborem elementu głównego

real	0m 1.817s
user	0m 1.744s
sys	0m 0.079s

   Błąd:

$$3.9874770152437124e^{-16}$$

Warto dodać, że powyższe wyniki są sumą importowania macierzy i wektora, wykonywania samego algorytmu jak i eksportowania wyniku do pliku. Aby uzyskać wyniki samego algorytmu można by odjąć odpowiednie wartości ze statystyk powyżej. Jako, że każda z macierzy jest obarczona podobnym błędem wyniki mimo to da się łatwo i w miarę obiektywnie porównywać.

Testy własne

Aby dalej zbadać zachowanie funkcji przeprowadzono dodatkowe testy z użyciem macierzy, z różnym uwarunkowaniem bloków, generowanych przy użyciu funkcji blockmat. Wszystkie macierze w tych testach miały rozmiar 10000.

1. Dla c_k=10:

   (a) Eliminacja Gaussa bez wyboru elementu głównego
   Czas:

real	0m 1.065s
user	0m 0.993s
sys	0m 0.066s

   Błąd:

$$2.817024391532641e^{-15}$$

(b) Eliminacja Gaussa z wyborem elementu głównego
Czas:

real	0m 1.355s
user	0m 1.279s
sys	0m 0.065s

Błąd:

$$3.6831648841939567e^{-16}$$

2. Dla $c_k = 10^2$:

(a) Eliminacja Gaussa bez wyboru elementu głównego
Czas:

real	0m 1.046s
user	0m 0.968s
sys	0m 0.070s

Błąd:

$$1.322600029496357e^{-13}$$

(b) Eliminacja Gaussa z wyborem elementu głównego
Czas:

real	0m 1.379s
user	0m 1.291s
sys	0m 0.076s

Błąd:

$$2.676736610141006e^{-15}$$

3. Dla $c_k = 10^5$:

(a) Eliminacja Gaussa bez wyboru elementu głównego
Czas:

real	0m 1.046s
user	0m 0.977s
sys	0m 0.061s

Błąd:

$$3.157195055392137e^{-11}$$

(b) Eliminacja Gaussa z wyborem elementu głównego

real	0m 1.411s
user	0m 1.329s
sys	0m 0.071s

Błąd:

$$2.593390435734477e^{-12}$$

4. Dla $c_k = 10^10$:

   (a) Eliminacja Gaussa bez wyboru elementu głównego
   Czas:

real	0m 1.092s
user	0m 1.006
sys	0m 0.073s

   Błąd:

$$2.9030699014245488e^{-6}$$

   (b) Eliminacja Gaussa z wyborem elementu głównego

real	0m 1.401s
user	0m 1.336s
sys	0m 0.052s

   Błąd:

$$2.5449254313170135e^{-7}$$

5. Dla $c_k = 10^13$:

   (a) Eliminacja Gaussa bez wyboru elementu głównego
   Czas:

real	0m 1.087s
user	0m 1.014s
sys	0m 0.060s

   Błąd:

$$0.0045178028780950225$$

   (b) Eliminacja Gaussa z wyborem elementu głównego

real	0m 1.349s
user	0m 1.260s
sys	0m 0.071s

   Błąd:

$$0.0002520045948780372$$

Warto dodać, że powyższe wyniki są sumą importowania macierzy i wektora, wykonywania samego algorytmu jak i eksportowania wyniku do pliku. Aby uzyskać wyniki samego algorytmu można by odjąć odpowiednie wartości ze statystyk powyżej. Jako, że każda z macierzy jest obarczona podobnym błędem wyniki mimo to da się łatwo i w miarę obiektywnie porównywać.

Wnioski

Łatwo da się zauważyć, że wybór elementu głównego zapewnia lepsze wyniki, natomiast łatwo też zauważyć, że obliczenia trwają nieco dłużej. Taka wymiana jest opłacalna gdy uwarunkowanie bloków jest duże, przez co błędy popełniane przez brak wyboru elementu głównego stają się coraz większe. Dla małych macierzy lub dobrego uwarunkowania bardziej opłacalna jest opcja bez wyboru elementu głównego, chyba że zależy nam na dokładności do 16 liczb po przecinku. Jak chodzi o wzrost czasu w stosunku do wielkości macierzy, to jest on zauważalny, ale nie porażająco wielki.
Wszystkie testy były przeprowadzane na komputerze wyposażonym w procesor Intel Core i5-6600K CPU @ 3.50GHz x 4 oraz 16GB ramu.