Rozwiązywanie liniowego układu równań

Lab5c

Wojciech Klusek 305943 grupa C

12 grudnia 2023

Spis treści

1 Zadanie **2**

2 Opis algorytmu **2**
2.1 Algorytm gradientu sprzężonego z alphą analityczną 3
2.2 Algorytm gradientu sprzężonego z alphą Armijo 3
2.3 Algorytm najszybszego spadku 4

3 Rozwiązanie **4**

4 Liczba iteracji dla rozmiaru macierzy **5**
4.1 FR z analityczną alphą 6
4.2 FR z alphą Armijo 6
4.3 NS 7

5 Liczba iteracji dla liczby unikalnych wartości własnych **7**
5.1 FR z analityczną alphą 8
5.2 FR z alphą Armijo 8
5.3 NS 9

6 Normy gradientu dla kolejnych iteracji **9**
6.1 FR z analityczną alphą 10
6.2 FR z alphą Armijo 10
6.3 NS 11

7 Test dokładności rozwiązania **11**
7.1 fminunc 11
7.2 FR z analityczną alphą 12
7.3 FR z alphą Armijo 12
7.4 NS 12

8 Wnioski **12**

9 Oświadczenie o samodzielności **12**

1 Zadanie

Zadaniem było rozwiązanie następującego nieosobliwego kwadratowego układu równań:

$$Ax = b,$$

$$b \in R^n, A \in R^{nxn}, A = A^T$$

Macierz A w układzie powyżej jest dodatnio określona, ma narzuconą liczbę unikalnych wartości własnych i została wygenerowana w następujący sposób:

$$D = diag(eigenvalues),$$

$$V = orth(rand(n)),$$

$$A = V * D * V^T$$

2 Opis algorytmu

Do rozwiązania powyższego układu równań wykorzystamy fakt, że rozwiązaniem tego układu jest x, który minimalizuje wartość poniższej funkcji:

$$f(x) = 0.5x^T Ax - x^T b,$$

$$H(f(x)) = A,$$

$$\nabla f(x) = Ax - b$$

Natomiast to minimalizacji powyższej funkcji wykorzystane zostaną algorytmy: gradientu sprzężonego oraz najszybszego spadku opisane poniżej.

2.1 Algorytm gradientu sprzężonego z alphą analityczną

Dane wejściowe:

A - macierz symetryczna i dodatnio określona
b - wektor prawych stron
ϵ - tolerancja dla warunku stopu
x_0 - przybliżone rozwiązanie początkowe
$d = -\nabla f(x_0)$ - kierunek początkowy

for $k = 0, 1, 2, \ldots$ **do** {
$\alpha_k = \frac{-\nabla f(x_0)^T d}{d^T A d}$ // *Analityczne wyznaczenie* α
$x_{k+1} = x_k + \alpha_k d$ // *Aktualizacja przybliżonego rozwiązania*
$\beta_k = \frac{\nabla f(x_{k+1})^T \nabla f(x_{k+1})}{\nabla f(x_k)^T \nabla f(x_k)}$ // *Współczynnik* β
$d = -\nabla f(x_{k+1}) + \beta_k d$ // *Aktualizacja kierunku poszukiwań*
if $\|\nabla f(x_{k+1})\| < \epsilon$ **then break** // *Warunek stopu*
}

2.2 Algorytm gradientu sprzężonego z alphą Armijo

Dane wejściowe:

A - macierz symetryczna i dodatnio określona
b - wektor prawych stron
ϵ - tolerancja dla warunku stopu
x_0 - przybliżone rozwiązanie początkowe
$d = -\nabla f(x_0)$ - kierunek początkowy
$alpha\_tol = 1e - 3$ - tolerancja α
$beta = 0.5$ - współczynnik kontrakcji

for $k = 0, 1, 2, \ldots$ **do** {
$\alpha_k = 1$
while $f(x_k + \alpha d) > f(x_k) + \alpha * alpha\_tol * \nabla f(x_k) d$ **do** {
$\alpha_k = beta * \alpha_k$ // α *Armijo*
}
$\alpha_k = \frac{-\nabla f(x_0)^T d}{d^T A d}$
$x_{k+1} = x_k + \alpha_k d$ // *Aktualizacja przybliżonego rozwiązania*
$\beta_k = \frac{\nabla f(x_{k+1})^T \nabla f(x_{k+1})}{\nabla f(x_k)^T \nabla f(x_k)}$ // *Współczynnik* β
$d = -\nabla f(x_{k+1}) + \beta_k d$ // *Aktualizacja kierunku poszukiwań*
if $\|\nabla f(x_{k+1})\| < \epsilon$ **then break** // *Warunek stopu*
}

2.3 Algorytm najszybszego spadku

Dane wejściowe:

A - macierz symetryczna i dodatnio określona
b - wektor prawych stron
ϵ - tolerancja dla warunku stopu
x_0 - przybliżone rozwiązanie początkowe
$d = -\nabla f(x_0)$ - kierunek początkowy

Iteracje:
for $k = 0, 1, 2, \ldots$ **do** {
  for $i = 0, 1, 2, \ldots$ **do** {
    $\alpha_k = \frac{1}{\lambda_i}$
    $x_{k+1} = x_k + \alpha_i d$ *// Aktualizacja przybliżonego rozwiązania*
  }
    if $\|\nabla f(x_{k+1})\| < \epsilon$ **then break**   *// Warunek stopu*
  }

3 Rozwiązanie

Rozwiązanie składa się z 9 plików:

1. `FR.m` - Implementacja algorytmu gradientu sprzężonego. Przyjmuje jako wejście zmienne:

    - `fun` - funkcja do minimalizacji,
    - `x0` - rozwiązanie początkowe,
    - `e` - warunek stopu dla normy gradientu,
    - `use_analytical_alpha` - opcjonalna flaga określająca czy wyznaczać α analitycznie, czy za pomocą algorytmu Armijo,
    - `plot` - opcjonalna flaga określająca, czy wyświetlać zmianę gradientu na koniec algorytmu.

    Funkcja zwraca wektor xFR będący rozwiązaniem zadania, wektor $fval$ będący wartością funkcji oraz it określającą liczbę iteracji.

2. `NS_eigs.m` - Implementacja algorytmu najszybszego spadku. Przyjmuje jako wejście zmienne:

    - `fun` - funkcja do minimalizacji,
    - `x0` - rozwiązanie początkowe,
    - `e` - warunek stopu dla normy gradientu,
    - `plot` - opcjonalna flaga określająca, czy wyświetlać zmianę gradientu na koniec algorytmu.

Funkcja zwraca wektor xNS będący rozwiązaniem zadania, wektor $fval$ będący wartością funkcji oraz it określającą liczbę iteracji.

3. `test.m` - Porównanie algorytmu FR, NS_eigs, fminunc z rozwiązaniem dokładnym. Przyjmuje jako wejście zmienne:
   - `matrix_size` - rozmiar macierzy A,
   - `iter` - liczba iteracji do wykonania,
   - `unique_eigenvalues_count` - liczba unikalnych wartości własnych macierzy A.

   Funkcja 3 liczby zawierające średnią liczbę iteracji dla każdego algorytmu.
4. `generate.m` - Generuje macierz A, wektor b i zwraca wartości własne. Przyjmuje jako wejście zmienne:
   - `n` - rozmiar macierzy A,
   - `unique_eigenvalues_count` - liczba unikalnych wartości własnych macierzy A.

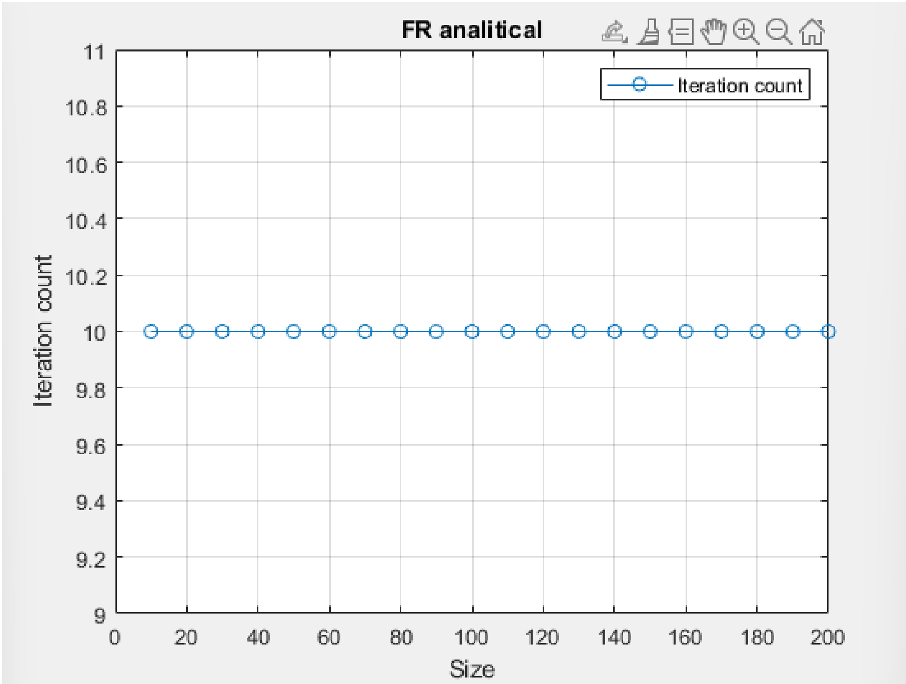
   Funkcja zwraca macierz A, wektor b oraz wartości własne - *eigenvalues* .
5. `fun.m` - Definiuje funkcję, której minimalizacja jest równoważna rozwiązaniu układu w zadaniu. Przyjmuje jako wejście zmienne:
   - `x` - argument funkcji,
   - `A` - macierz zgodna z zadaniem,
   - `b` - prawe strony układu zgodne z zadaniem.

   Funkcja zwraca wartość funkcji f, gradient *grad* oraz hesjan - *hess*.
6. `fminunc_solution.m` - Wywołuje wbudowaną funkcję fminunc z parametrami określonymi w zadaniu.
7. `plot_gradients.m` - Wyświetla kolejne wartości gradientów na wykresie.
8. `eigenvalues_iteration_test.m` - Wyświetla zależność liczby wartości własnych od liczby iteracji poszczególnych algorytmów.
9. `size_iteration_test.m` - Wyświetla zależność rozmiaru macierzy od liczby iteracji poszczególnych algorytmów.
10. `plot_iterations.m` - Funkcja pomocnicza do wyświetlania wykresów.

4 Liczba iteracji dla rozmiaru macierzy

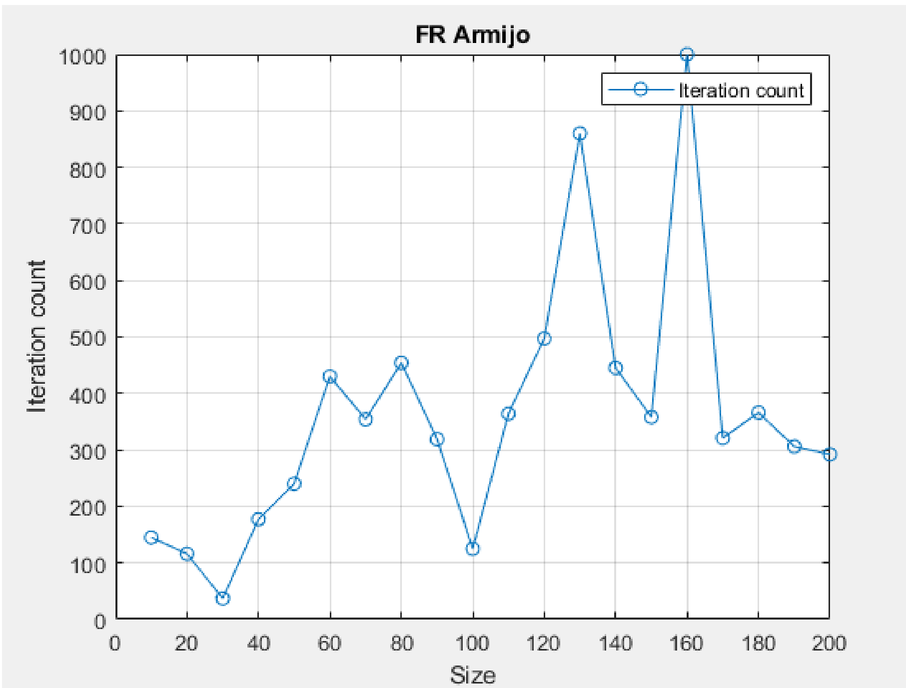
Podane wykresy zostały wygenerowane dla macierzy o 10 wartościach własnych.

4.1 FR z analityczną alphą



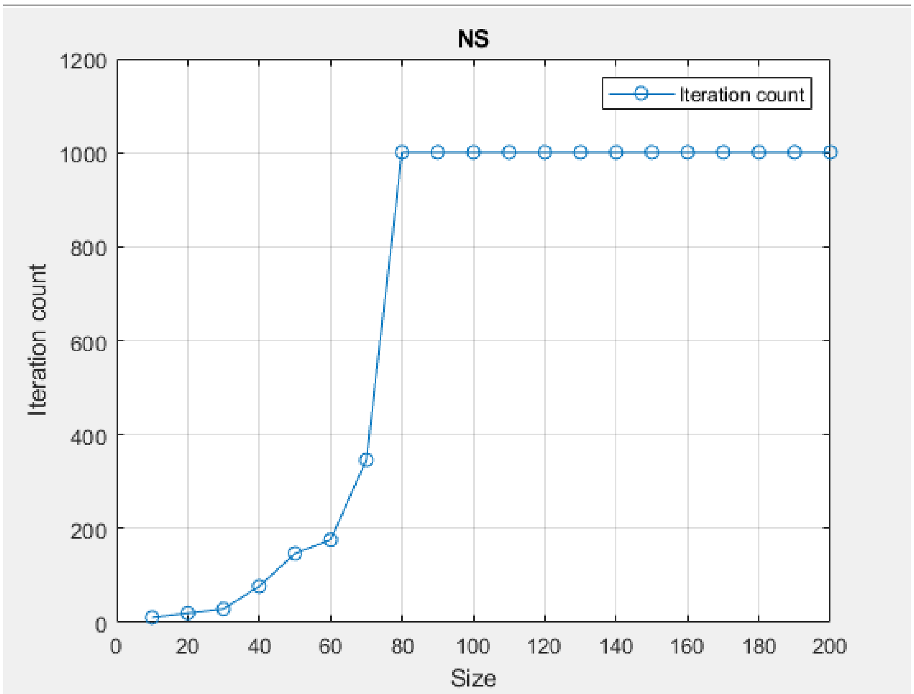
Rysunek 1: Liczba iteracji dla FR z analityczną alphą.

4.2 FR z alphą Armijo



Rysunek 2: Liczba iteracji dla FR z alphą Armijo.

4.3 NS

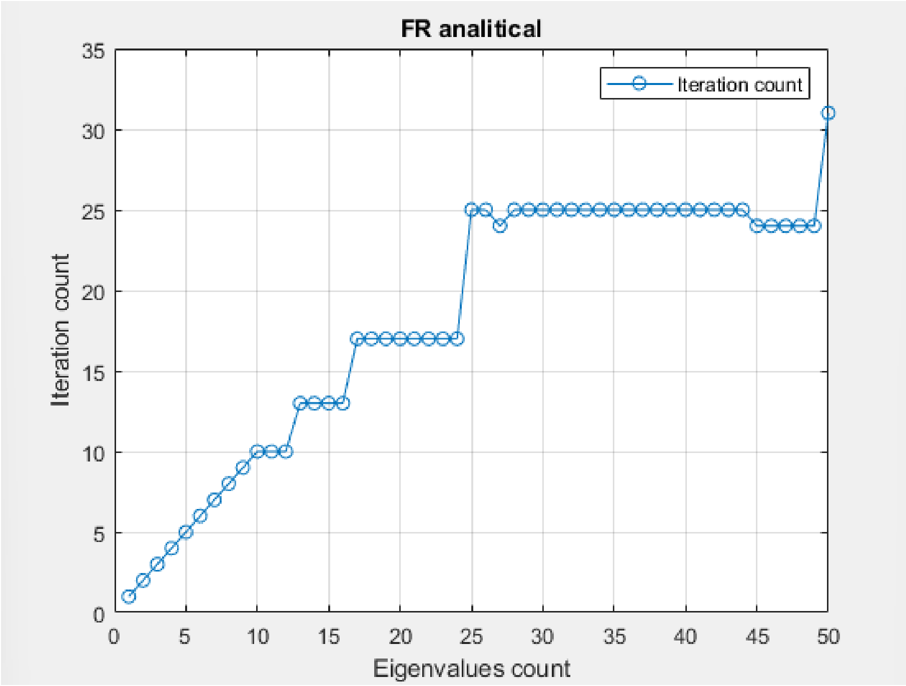


Rysunek 3: Liczba iteracji dla NS.

5 Liczba iteracji dla liczby unikalnych wartości własnych

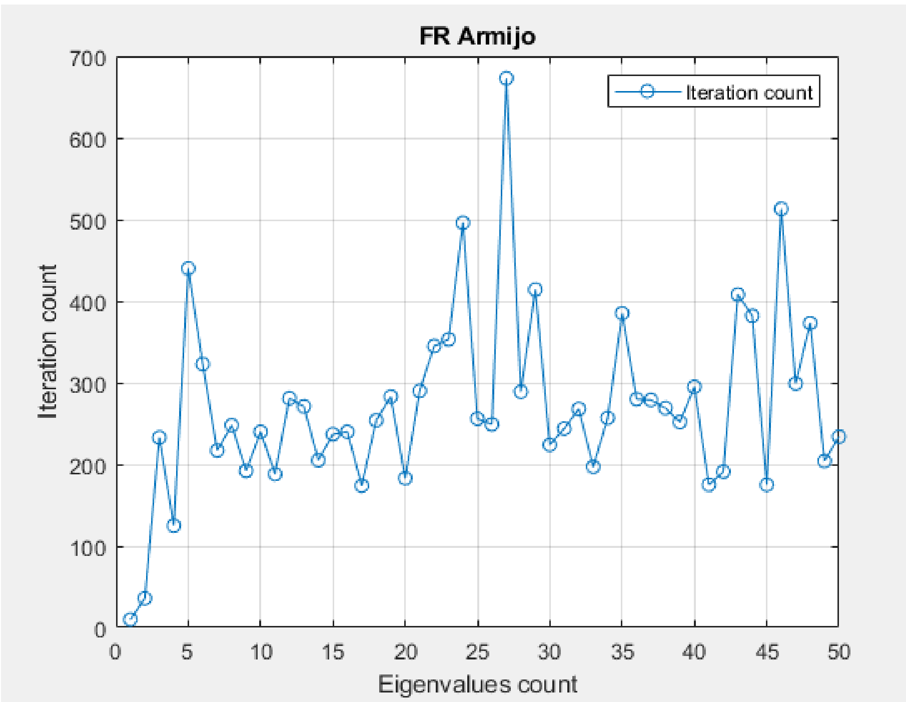
Podane wykresy zostały wygenerowane dla macierzy o rozmiarze 50.

5.1 FR z analityczną alphą



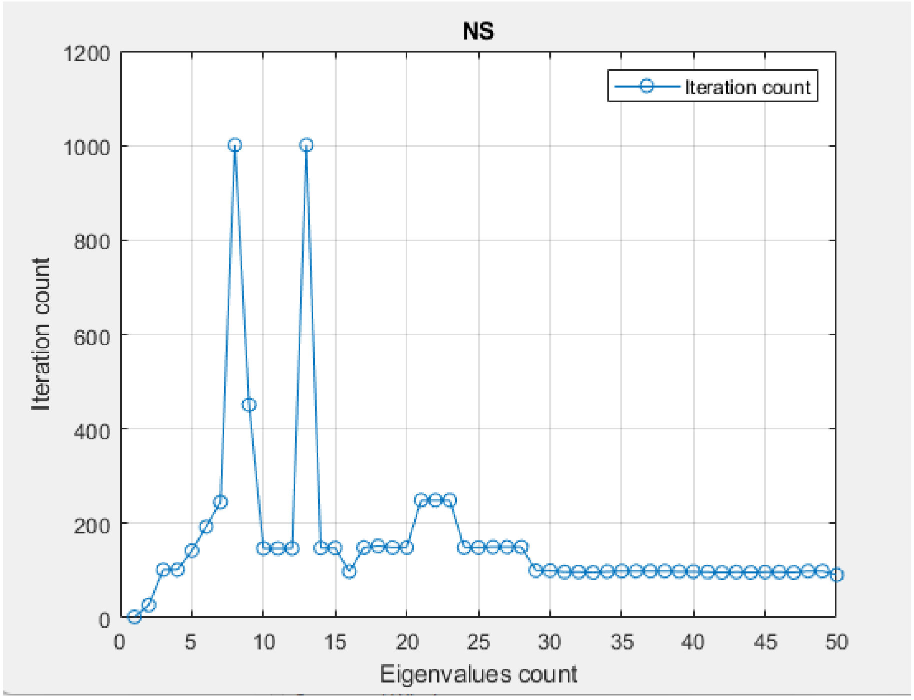
Rysunek 4: Liczba iteracji dla FR z analityczną alphą.

5.2 FR z alphą Armijo



Rysunek 5: Liczba iteracji dla FR z alphą Armijo.

5.3 NS

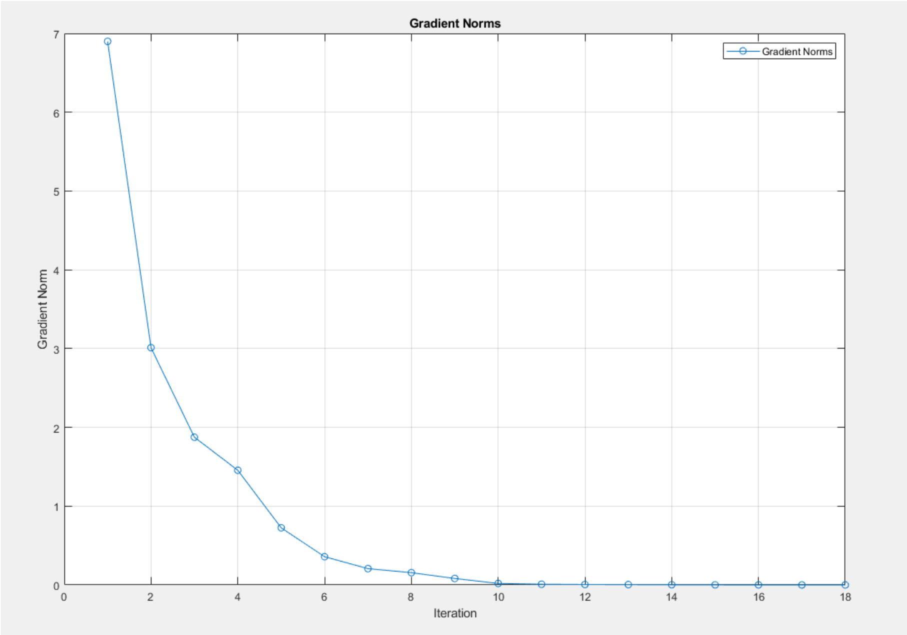


Rysunek 6: Liczba iteracji dla NS.

6 Normy gradientu dla kolejnych iteracji

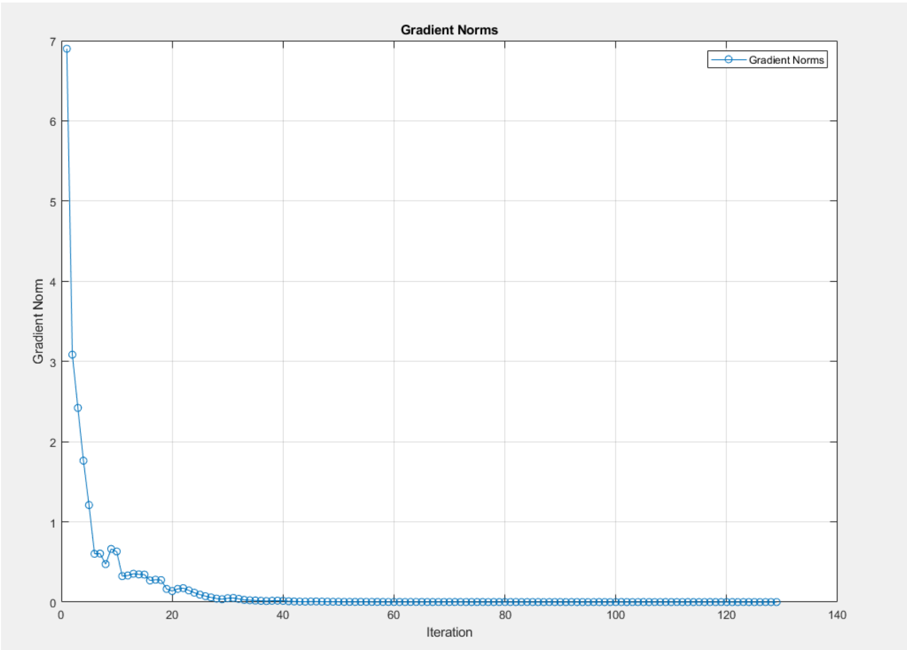
Podane wykresy zostały wygenerowane dla macierzy o rozmiarze 50 o 20 wartościach własnych.

6.1 FR z analityczną alphą



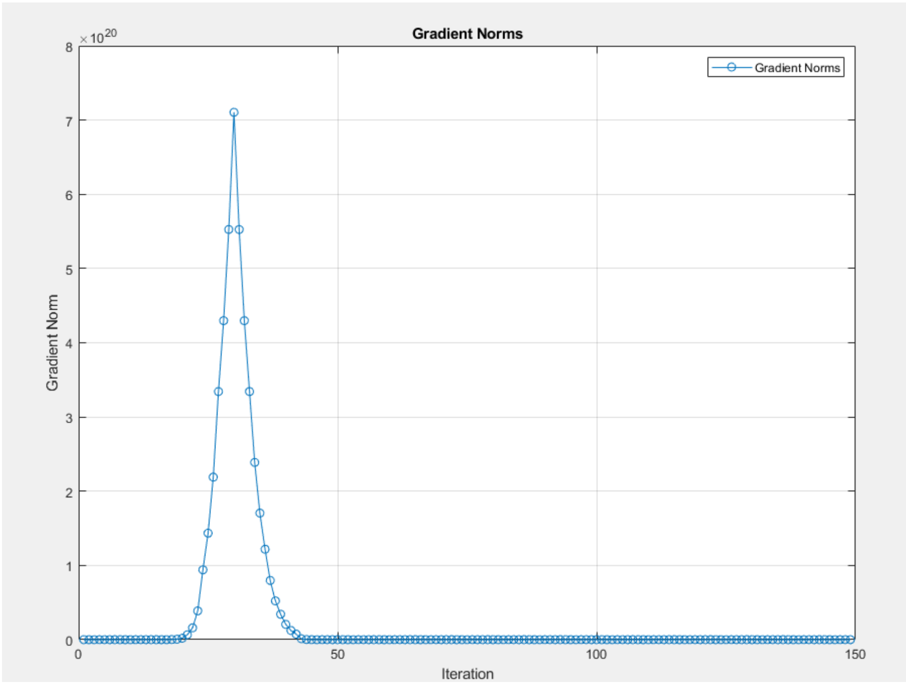
Rysunek 7: Norma gradientu dla FR z analityczną alphą.

6.2 FR z alphą Armijo



Rysunek 8: Norma gradientu dla FR z alphą Armijo.

6.3 NS



Rysunek 9: Norma gradientu dla NS.

7 Test dokładności rozwiązania

Do testu dokładności wykorzystane zostały następujące zmienne:

- Rozmiar macierzy A - 50,
- Liczba różnych wejść - 100,
- Liczba unikalnych wartości własnych - 20,
- Ziarno generatora liczb losowych - 1,
- Warunek normy gradientu - $1e-5$,
- Liczba miejsc po przecinku do porównania wartości funkcji - 10.

Poniżej przedstawione zostały wyniki dla powyższych parametrów:

7.1 fminunc

- Dokładność: 94% (określa jak często algorytm zwrócił tą samą wartość co rozwiązanie dokładne $x = A\backslash b$).
- Średnia norma różnicy rozwiązania z rozwiązaniem dokładnym: 6.4689e-07.

7.2 FR z analityczną alphą

- Dokładność: 100%.
- Średnia liczba iteracji: 15.82.
- Średnia norma różnicy rozwiązania z rozwiązaniem dokładnym: 6.8206e-08.

7.3 FR z alphą Armijo

- Dokładność: 96%.
- Średnia liczba iteracji: 178.93.
- Średnia norma różnicy rozwiązania z rozwiązaniem dokładnym: 1.2569e-06.

7.4 NS

- Dokładność: 98%.
- Średnia liczba iteracji: 72.71.
- Średnia norma różnicy rozwiązania z rozwiązaniem dokładnym: 3.2235e-07.

8 Wnioski

Podsumowując, najlepsze rezultaty osiągnął algorytm FR z analitycznie wyliczaną wartością parametru α. Liczba iteracji tego algorytmu była bliska ilości wartości własnych macierzy A, co stanowiło najmniejszą liczbę porównując z innymi analizowanymi algorytmami. Co istotne, ta liczba iteracji nie była uzależniona od rozmiaru macierzy. Algorytm FR z zastosowaniem metody Armijo także wykazywał się dobrą skutecznością, choć wykonuje większą liczbę iteracji. Niemniej jednak, zazwyczaj udawało mu się znaleźć poprawne rozwiązanie, niezależnie od rozmiaru problemu.

Najmniejszą efektywność prezentował algorytm NS, który przy rozmiarze macierzy rzędu około 60, wykazywał znaczną liczbę iteracji i nie był w stanie odnaleźć odpowiedniego rozwiązania w ustalonym limicie 1000 iteracji. Warto zaznaczyć, że pomimo tych różnic, wszystkie analizowane algorytmy dostarczały wyniki zbliżone do dokładnego rozwiązania.

9 Oświadczenie o samodzielności

Oświadczam, że niniejsza praca stanowiąca podstawę do uznani osiągnięcia efektów uczenia się z przedmiotu Programowanie Matematyczne została wykonana przeze mnie samodzielnie.

Wojciech Klusek 305934