Aproksymacja rozwiązań układów równań metodą Jacobiego i Gaussa-Seidla

Bartosz Zasieczny

26 stycznia 2014

Spis treści

1 Treść zadania

Za pomocą metod Jacobiego i Siedla wyznaczyć przybliżone rozwiązanie $\tilde{x}$ układu równań liniowych $Ax = b$ ($A = [a_{i,j}] \in \mathbb{R}^{n\times n}$), przyjmując że $\tilde{x} = x^{(k)}$, gdzie k jest najmniejszą liczbą naturalną dla której zachodzi nierówność:

$$\frac{\|x^{(k)} - x^{(k-1)}\|_\infty}{\|x^{(k)}\|_\infty} < \epsilon.$$

Wykonać obliczenia kontrolne m. in. dla macierzy Pei i Hillberta i omówić wyniki, podając wartość $\|b - A\tilde{x}\|_\infty$, gdzie $\tilde{x}$ jest obliczonym rozwiązaniem,

jak również przyjmując różne wartości parametrów n i d. Obliczenia wykonać dla $\epsilon = 5 \cdot 10^{-5}$ i $\epsilon = 5 \cdot 10^{-7}$.

2 Algorytmy

2.1 Metoda Jacobiego

Metoda Jacobiego jest metodą iteracyjną, gdzie kolejne przybliżenia rozwiązania układu równań $Ax = b$ znajdujemy poprzez rozwiązanie poniższego równania na macierzach:

$$x^{(k+1)} = Mx^{(k)} + Nb$$

gdzie:

$$N = D^{-1}$$

$$M = -N(L + U)$$

$$D_{ij} = \begin{cases} A_{ij} & i = j \\ 0 & \text{w p. p.} \end{cases}$$

$$L_{ij} = \begin{cases} A_{ij} & i < j \\ 0 & \text{w p. p.} \end{cases}$$

$$U_{ij} = \begin{cases} A_{ij} & i > j \\ 0 & \text{w p. p.} \end{cases}$$

Natomiast x^0 jest wektorem zerowym. Bezpośredni wzór na kolejne wartości wektora $x^{(k+1)}$ to:

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1;j\neq i}^{n} a_{ij} x_j^{(k)}}{a_{ii}}$$

2.2 Metoda Gaussa-Seidla

Metoda Gaussa-Seidla różni się od poprzedniej tylko wzorem, za pomocą którego wyznaczamy następne iteracje:

$$x^{(k+1)} = (D - L)^{-1} \cdot (Ux^{(k)} + b)$$

A bezpośredni wzór na kolejne wartosci wektora $x^{(k+1)}$ to:

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}}{a_{ii}}$$

3 Przykładowe rozwiązania

3.1 Macierz Pei

Macierz Pei jest zdefiniowana w następujący sposób:

$$P_{ij} = \begin{cases} d & i = j \\ 1 & \text{w p. p.} \end{cases}$$

gdzie d jest podanym parametrem rzeczywistym, a $(i, j = 1, 2, \ldots, n)$.

Uwarunkowanie tej macierzy jest tym lepsze, im większa jest wartość bezwzględna d. Metoda Gaussa-Seidla jest szybciej zbieżna i bardziej odporna na gorsze uwarunkowanie zadanej macierzy.

3.1.1 Metoda Jacobiego

- $n = 5$, $b = [3, 4, 5, 6, 7]^T$, $d = 1$, $\epsilon = 5 \cdot 10^{-7}$

  Brak wyniku po 510 iteracjach. Algorytm się zapetla, gdyż wartość błędu bezwzlędnego międyz kolejnymi iteracjami nie zmniejsza się i liczby szybko rosną do wartości wykraczających poza arytmetykę double i dalsze obliczenia nie są możliwe.

  Polecenie: `./program -peya 1 -p 0.0000005 -v 5 3 4 5 6 7 -j`

- $n = 3$, $b = [2, 3, 4]^T$, $d = 2$, $\epsilon = 5 \cdot 10^{-7}$

  Program się zapętla podobnie jak w pierwszym przykładzie - powód podobny. Algorytm zamiast zbiegać do rozwiązania - rozbiega się.

  Polecenie: `./program -peya 2 -p 0.0000005 -v 3 2 3 4 -j`

- $n = 10$, $b = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12]^T$, $d = -20$, $\epsilon = 5 \cdot 10^{-7}$

  Metoda Jacobiego potrzebowała tutaj 23 iteracji:

  $$x^{(23)} = [-0.467532249, -0.515151297, -0.562770344, -0.610389392, -0.658008439, -0.705627487, -0.753246535, -0.800865582, -0.84848463, -0.896103678]$$

  $$e = 2.9803581 \cdot 10^{-7}$$

  Polecenie: `./program -peya -20 -p 0.0000005 -v 10 3 4 5 6 7 8 9 10 11 12 -j`

- $n = 10$, $b = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12]^T$, $d = 10000$, $\epsilon = 5 \cdot 10^{-7}$

  Program zwrócił wynik juz po 5 iteracjach:

$$x^{(5)} = [0.000299280602, 0.000399290603, 0.000499300604,$$
$$0.000599310605, 0.000699320606, 0.000799330607, 0.000899340608,$$
$$0.000999350609, 0.00109936061, 0.00119937061]$$

$$e = 1.09501748 \cdot 10^{-7}$$

Polecenie: `./program -peya 10000 -p 0.0000005 -v 10 3 4 5 6 7 8 9 10 11 12 -j`

3.1.2 Metoda Gaussa-Seidla

Ten algorytm dał parę interesujących wyników:

- $n = 5$, $b = [3, 4, 5, 6, 7]^T$, $d = 1$, $\epsilon = 5 \cdot 10^{-7}$

  Jak widać, choć macierz ta nie powinna mieć jakiegokolwiek rozwiązania, algorytm po bardzo wielu iteracjach (2000002) zatrzymuje się i daje następującą odpowiedź:

  $$x^{(2000002)} = [-8000000, 2000000, 2000000, 2000000, 2000000]$$

  $$e = 5 \cdot 10^{-7}$$

  Widzimy więc, że dla $d = 1$ zadanie to jest **bardzo źle** uwarunkowane, ponieważ algorytm zwraca wynik, podczas gdy rozwiązanie nie istnieje.

  Polecenie: `./program -peya 1 -p 0.0000005 -v 5 3 4 5 6 7 -gs`

- $n = 3$, $b = [2, 3, 4]^T$, $d = 2$, $\epsilon = 5 \cdot 10^{-7}$

  Ten algorytm, w przeciwieństwie do metody Jacobiego daje dla tej niezbyt dobrze uwarunkowanej macierzy, dość szybko oczkiwane przybliżenie (16 iteracji):

  $$x^{(16)} = [-0.250000011, 0.750000171, 1.74999992]$$

  $$e = 1.82909527 \cdot 10^{-7}$$

  Polecenie: `./program -peya 2 -p 0.0000005 -v 3 2 3 4 -gs`

- $n = 10$, $b = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12]^T$, $d = -20$, $\epsilon = 5 \cdot 10^{-7}$

  Zadowalajacy wynik w tym wypadku osiągamy bardzo szybko - zaledwie 12 iteracji.

$$x^{(12)} = [-0.467532, -0.515151, -0.56277, -0.610389, -0.658008, \\ -0.705627, -0.753247, -0.800866, -0.848485, -0.896104]$$

$$e = 2.98036 \cdot 10^{-7}$$

Polecenie: `./program -peya -20 -p 0.0000005 -v 10 3 4 5 6 7 8 9 10 11 12 -gs`

- $n = 10$, $b = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12]^T$, $d = 10000$, $\epsilon = 5 \cdot 10^{-7}$

  Przy takiej wartości n algorytm daje wynik już po 3 iteracjach:

$$x^{(3)} = [0.000299281, 0.000399291, 0.000499301, 0.000599311, 0.000699321, \\ 0.000799331, 0.000899341, 0.000999351, 0.00109936, 0.00119937]$$

$$e = 2.89817 \cdot 10^{-7}$$

Polecenie: `./program -peya 10000 -p 0.0000005 -v 10 3 4 5 6 7 8 9 10 11 12 -gs`

3.2 Macierz Hillberta

Macierz Hillberta jest dana następujacym wzorem:

$$H_{ij} = \frac{1}{i + j + 1} \ (i, j = 1, 2, \ldots, n)$$

Macierz Hillberta jest macierzą źle uwarunkowaną numerycznie – wartości elementów macierzy maleją w kierunku zera wraz z rosnącymi indeksami, a gdy je (elementy macierzy) mnożymy to dostajemy jeszcze mniejsze wartości, przez co dochodzi do utraty cyfr znaczących i algorytm Jacobiego przestaje być zbieżny dla tej macierzy, a algorytm Gaussa-Seidla zbiega bardzo wolno. Potwierdza się też wcześniejsza obserwacja, że **metoda Gaussa-Siedla** jest szybciej zbieżna. Wyniki aproksymacji rozwiązania układów dla obu metod to kolejno:

3.2.1 Metoda Jacobiego

$$b = [3, 4]^T$$

- $\epsilon = 5 \cdot 10^{-5}$ – 257 iteracji, $x^{(257)} = [-95.9992, 139.999]^T$, $e = 4.80048 \cdot 10^{-5}$, polecenie: `./program -hillbert -p 0.00005 -v 2 3 4 -j`

- $\epsilon = 5 \cdot 10^{-7}$ – 399 iteracji, $x^{(399)} = [-95.9998, 140]^T$, $e = 4.91055 \cdot 10^{-7}$, polecenie: `./program -hillbert -p 0.0000005 -v 2 3 4 -j`

$$b = [3, 4, 5]^T$$

- $\epsilon = 5 \cdot 10^{-5}$ – 1089 iteracji, brak rozwiązania, polecenie: `./program -hillbert -p 0.00005 -v 3 3 4 5 -j`
- $\epsilon = 5 \cdot 10^{-7}$ – 1089 iteracji, brak rozwiązania, polecenie: `./program -hillbert -p 0.0000005 -v 3 3 4 5 -j`

3.2.2 Metoda Gaussa-Seidla

$$b = [3, 4]^T$$

- $\epsilon = 5 \cdot 10^{-5}$ – 112 iteracji, $x^{(112)} = [-95.9187, 139.898]^T$, $e = 4.8418 \cdot 10^{-5}$, polecenie: `./program -hillbert -p 0.00005 -v 2 3 4 -gs`
- $\epsilon = 5 \cdot 10^{-7}$ – 183 iteracje, $x^{(183)} = [-95.9992, 139.999]^T$, $e = 4.95057 \cdot 10^{-7}$, polecenie: `./program -hillbert -p 0.0000005 -v 2 3 4 -gs`

$$b = [3, 4, 5]^T$$

- $\epsilon = 5 \cdot 10^{-5}$ – 1723 iteracje, $x^{(1723)} = [438.496, -1643.71, 1338.76]^T$, $e = 4.9969 \cdot 10^{-5}$, polecenie: `./program -hillbert -p 0.00005 -v 3 3 4 5 -gs`
- $\epsilon = 5 \cdot 10^{-7}$ – 3751 iteracji, $x^{(3751)} = [449.883, -1679.63, 1364.73]^T$, $e = 4.99272 \cdot 10^{-7}$, polecenie: `./program -hillbert -p 0.0000005 -v 3 3 4 5 -gs`

$$b = [3, 4, 5, 6, 7]^T$$

- $\epsilon = 5 \cdot 10^{-7}$ – 655824 iteracji, $x^{(655824)} = [2887.91, -31059.7, 104369, -137452, 61880.9]^T$, $e = 4.99999 \cdot 10^{-7}$, polecenie: `./program -hillbert -p 0.0000005 -v 5 3 4 5 6 7 -gs`

4 Kompilacja i obsługa programu

4.1 Wymagania

Aby skompilować program należy spełnić następujące wymagania dotyczące oprogramowania:

- kompilator $G{+}{+}$ w wersji 4.7 lub późniejszej - kompilator musi obsługiwać standard $C^{++}11$,

- obecność narzędzia GNU Make

Powyższe wymagania powinny być automatycznie spełnione w każdej aktualnej dystrybucji GNU/Linux.

4.2 Kompilacja

Należy przejść do katalogu `prog` i wykonać polecenie `make` - kompilacja wykona się automatycznie. W pliku `Makefile` podane są polecenia, które należy wykonać aby skompilować program ręcznie.

4.3 Obsługa programu

Program uruchamiamy za pomocą pliku `program`, po jego nazwie podając ciąg będący kombinacją ponizszych parametrów:

- `-peya <d>` – użycie macierzy Pei z parametrem d
- `-hillbert` – użycie macierzy Hillberta
- `-p <e>` – definicja wielkości ϵ
- `-j` – użycie metody Jacobiego
- `-gs` – użycie metody Gaussa-Seidla
- `-v <n>` $b_0\ b_1 \ldots b_n$ – podanie rozmiaru macierzy kwadratowej/wektora b i podanie wartości wektora b

Przykład: szukamy przybliżonego rozwiązania dla macierzy hillberta, gdzie $n = 4$, $b = [465, 6, 7, -55]^T$, używając metody Gaussa-Seidla i precyzji $\epsilon = 5 \cdot 10^{-5}$.

```
./program -hillbert -p 0.00005 -v 4 465 6 7 -55 -gs
```