Układy równań liniowych

Joanna Klimek

Informatyka, WIET

11.05.2021r.

Zadanie 1.

Dana jest macierz $A = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \end{bmatrix}$.

a) Udowodnij, ze macierz A jest osobliwa.

Macierz jest osobliwa gdy jej wyznacznik jest równy zero.

Obliczam wyznacznik:

$$detA = \begin{vmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \end{vmatrix} = 0.1 * 0.5 * 0.9 + 0.4 * 0.8 * 0.3 + 0.7 * 0.2 * 0.6$$

$$-0.3 * 0.5 * 0.7 - 0.6 * 0.8 * 0.1 - 0.9 * 0.2 * 0.4$$

$$= 0.045 + 0.096 + 0.084 - 0.105 - 0.048 - 0.072 = 0$$

A więc nasza macierz jest osobliwa.

b) Jaki jest zbiór rozwiązań układu równań $Ax = b$ jesli $b = \begin{bmatrix}0.1\\0.3\\0.5\end{bmatrix}$.

Powstały układ równań:

$$\begin{bmatrix}0.1 & 0.2 & 0.3\\0.4 & 0.5 & 0.6\\0.7 & 0.8 & 0.9\end{bmatrix} * \begin{bmatrix}x\\y\\z\end{bmatrix} = \begin{bmatrix}0.1\\0.3\\0.5\end{bmatrix}$$

Układ równań w postaci macierzy współczynników przekształcam za pomocą operacji elementarnych metodą Gaussa:

$$\left[\begin{array}{ccc|c}0.1 & 0.2 & 0.3 & 0.1\\0.4 & 0.5 & 0.6 & 0.3\\0.7 & 0.8 & 0.9 & 0.5\end{array}\right] \quad w2 - 4 * w1$$

$$\left[\begin{array}{ccc|c}0.1 & 0.2 & 0.3 & 0.1\\0.0 & -0.3 & -0.6 & -0.1\\0.7 & 0.8 & 0.9 & 0.5\end{array}\right] \quad w3 - 7 * w1$$

$$\left[\begin{array}{ccc|c}0.1 & 0.2 & 0.3 & 0.1\\0.0 & -0.3 & -0.6 & -0.1\\0.0 & -0.6 & -1.2 & -0.2\end{array}\right] \quad w3 - 2 * w2$$

$$\left[\begin{array}{ccc|c}0.1 & 0.2 & 0.3 & 0.1\\0.0 & -0.3 & -0.6 & -0.1\\0.0 & 0.0 & 0.0 & 0.0\end{array}\right]$$

Otrzymałam macierz w postaci schodkowej, na podstawie której układam równania zaczynając od ostatniego wiersza (w tym przypadku ostatnie równanie jest zbędne, gdyż cały wiersz jest zapełniony zerami).

$$\begin{cases} -0.3y - 0.6z = -0.1\\ 0.1x + 0.2y + 0.3z = 0.1\end{cases}$$

$$\begin{cases} 3y + 6z = 1\\ x + 2y + 3z = 1\end{cases}$$

$$\begin{cases} y = \frac{1}{3} - 2z\\ x + 2\left(\frac{1}{3} - 2z\right) + 3z = 1\end{cases}$$

$$\begin{cases} y = \frac{1}{3} - 2z \\ x - z = \frac{1}{3} \end{cases}$$

$$\begin{cases} y = 1 - 2x \\ z = x - \frac{1}{3} \end{cases}$$

Ostateczny zbiór rozwiązań:

$$\begin{cases} x \in \mathbb{R} \\ y = 1 - 2x \\ z = x - \frac{1}{3} \end{cases}$$

Jest ich nieskończenie wiele – jest to zgodne z faktem, że jeśli macierz główna układu jest osobliwa, układ nie może mieć dokładnie jednego rozwiązania.

c) W którym momencie eliminacja Gaussa z częściowym przesuwaniem elementu wiodącego (ang. partial pivoting) nie powiedzie się, jeśli rozwiązujemy ten układ równań w dokładnej arytmetyce?

Algorytm wyboru częściowego:

W k-tym kroku:

1) Szukamy indeksu p (numer wiersza) elementu największego co do modułu w k-tej kolumnie od przekątnej w dół.
2) Jeśli odnaleziony element jest równy zero układ jest osobliwy i przerywamy metodę.
3) Jeśli odnaleziony element jest różny od zera:
   a. zamieniamy miejscami wiersz p-ty z k-tym
   b. wykonujemy eliminację Gaussa w danej kolumnie (zerujemy k-tą kolumnę poniżej przekątnej, odejmując od kolejnych wierszy k-ty wiersz przemnożony przez odpowiedni współczynnik)

Spróbuję wykorzystać tę metodę.

Początkowa macierz uzupełniona:

$$\left[\begin{array}{ccc|c} 0.1 & 0.2 & 0.3 & 0.1 \\ 0.4 & 0.5 & 0.6 & 0.3 \\ 0.7 & 0.8 & 0.9 & 0.5 \end{array}\right]$$

Krok 1:

W pierwszej kolumnie odnajduję element największy co do modułu – jest to 0.7.

$0.7 \neq 0$ więc układ wejściowy nie jest osobliwy i mogę kontynuować.

Zamieniam miejscami pierwszy wiersz z wierszem, w którym odnalazłam element największy, czyli w tym przypadku trzecim, otrzymując macierz jak poniżej:

$$\left[\begin{array}{ccc|c} 0.7 & 0.8 & 0.9 & 0.5 \\ 0.4 & 0.5 & 0.6 & 0.3 \\ 0.1 & 0.2 & 0.3 & 0.1 \end{array}\right]$$

Kontynuuję metodę eliminacji Gaussa, czyli dążę do wyzerowania pierwszej kolumny poniżej przekątnej.

$$\left[\begin{array}{ccc|c} 0.7 & 0.8 & 0.9 & 0.5 \\ 0.4 & 0.5 & 0.6 & 0.3 \\ 0.1 & 0.2 & 0.3 & 0.1 \end{array}\right] w2 - \frac{0.4}{0.7} * w1$$

$$\left[\begin{array}{ccc|c} 0.7 & 0.8 & 0.9 & 0.5 \\ 0.0 & X & X & X \\ 0.1 & 0.2 & 0.3 & 0.1 \end{array}\right]$$

W punktach wyznaczonych powyżej znakiem *X* pojawia się problem, z powodu niedokładnej wartości wyników:

$$0.5 - \frac{0.4}{0.7} * 0.8 = 0{,}042857142857142857142857142857 \ldots$$

$$0.6 - \frac{0.4}{0.7} * 0.9 = 0{,}085714285714285714285714285714 \ldots$$

$$0.7 - \frac{0.4}{0.7} * 0.5 = 0{,}414285714285714285714285714285 \ldots$$

W dokładnej arytmetyce nie jest możliwe rozwiązanie tego równania w ten sposób, z powodu utraty dokładności przy zerowaniu kolumn.

d) Ponieważ niektóre elementy macierzy A nie są dokładnie reprezentowalne w pamięci komputera, macierz ta zapisana w pamięci komputera przestanie być dokładnie osobliwa a eliminacja Gaussa może się powieść. Rozwiąż układ równań metodą *numpy.linalg.solve*.
Porównaj otrzymany wynik z dokładnym rozwiązaniem z punktu (b).

Wykorzystany kod programu:

```
import numpy as np

A = np.array([[0.1, 0.2, 0.3],
              [0.4, 0.5, 0.6],
              [0.7, 0.8, 0.9]])

B = np.array([0.1, 0.3, 0.5])

X = np.linalg.solve(A, B)
X2 = np.linalg.inv(A).dot(B)  # dodatkowa próba

# SPRAWDZENIE 1
print(X)
print(np.allclose(np.dot(A, X), B))

print(X2)
print(np.allclose(np.dot(A, X2), B))

# SPRAWDZENIE 2

x = X[0]
y = 1 - (2 * x)
z = x - (1 / 3)

X3 = B = np.array([x, y, z])

print(X3)
print(np.allclose(X, X3))
```

Otrzymane wyniki:

```
[ 0.16145833  0.67708333 -0.171875  ]
True
[0.5 1.  0. ]
False
[ 0.16145833  0.67708333 -0.171875  ]
True
```

Spostrzeżenia:

- Drugi sposób nie dał poprawnego rozwiązania. Jest to zgodne z przewidywaniami – macierz osobliwa jest nieodwracalna, więc ten sposób nie mógł dać dobrego wyniku.
- Wykorzystana metoda *numpy.linalg.solve* zwróciła tylko jeden wynik, a jak pokazano w punkcie b powinno być ich nieskończenie wiele.
- Warto jednak zauważyć, że wynik jest w przybliżeniu poprawny co dowodzi wykorzystana funkcja *np.allclose(np.dot(A, X2), B)*, sprawdzająca wynik bezpośrednio w układzie równań, a także druga metoda, porównująca wynik ze zbiorem rozwiązań otrzymanym w punkcie b.

Co ciekawe po przestawieniu pierwszego i drugiego wiersza wyniki są zgoła inne:

```
[-2.08333333 -2.83333333  4.25      ]
True
[-1.5 -3.5  4.5]
False
[-2.08333333  5.16666667 -2.41666667]
False
```

Otrzymany wynik, różni się od poprzedniego i tylko pierwsza metoda sprawdzająca potwierdza jego poprawność.

e) Oblicz metodą *numpy.linalg.cond* współczynnik uwarunkowania macierzy $cond(A)$. Do ilu cyfr powinno być dokładne rozwiązanie otrzymane metodą *numpy.linalg.solve*?

Wykorzystany kod:

```
import numpy as np

A = np.array([[0.1, 0.2, 0.3],
              [0.4, 0.5, 0.6],
              [0.7, 0.8, 0.9]])

condA = np.linalg.cond(A)

print("cond(A) = " + "{:.2f}".format(condA))
```

Wynik:

```
cond(A) = 21118968335779856.00
```

Od razu widać, że jest on bardzo duży, czyli nasz układ jest źle uwarunkowany.

Spróbujemy jednak obliczyć liczbę znaczących miejsc po przecinku.

Przyjmując, że komputer przechowuje liczby rzeczywiste za pomocą 24-bitowej mantysy, epsilon maszynowy wynosi:

$$\varepsilon = 2^{-23} = 0{,}119209 * 10^{-6}$$

Obliczam iloczyn epsilona maszynowego i współczynnika uwarunkowania macierzy:

$$\begin{aligned}\varepsilon * cond(A) &= 0{,}119209 * 10^{-6} * 21118968335779856 \\ &= 2\,517\,571\,096\,339\,980{,}853904 * 10^{-6}\end{aligned}$$

Warunek poniżej wyznacza nam m – liczbę cyfr znaczących wyniku.

$$\varepsilon * cond(A) < 0.5 * 10^{-m}$$

$$2\,517\,571\,096\,339\,980{,}853904 * 10^{-6} < 0.5 * 10^{-m}$$

$$5\,035\,142\,192\,679\,961{,}707808 * 10^{-6} < 10^{-m}$$

$$\log_{10} 5\,035\,142\,192\,679\,961{,}707808 * 10^{-6} < \log_{10} 10^{-m}$$

$$\log_{10} 5\,035\,142\,192\,679\,961{,}707808 + \log_{10} 10^{-6} < \log_{10} 10^{-m}$$

$$\log_{10} 5\,035\,142\,192\,679\,961{,}707808 - 6 < -m$$

$$15.702011739561792761824 6 - 6 < -m$$

$$9.702011739561792761824 6 < -m$$

$$m < -9.702011739561792761824 6$$

$$m = -10$$

Kod programu potwierdzający obliczenia:

```
A = np.array([[0.1, 0.2, 0.3],
              [0.4, 0.5, 0.6],
              [0.7, 0.8, 0.9]])

condA = np.linalg.cond(A)

print("cond(A) = " + "{:.2f}".format(condA))

epsilon = 0.119209 / 1000000

condAeps = condA * epsilon

print("cond(A) * epsilon =", condAeps)

print("cond(A) * epsilon <= 0.5 * 10^(-m)")

m = condAeps * 0.5
m = np.log10(m)
m *= (-1)
m = np.floor(m)

print("m =", m)
```

Wynik:

```
cond(A) = 211189683357798560.00
cond(A) * epsilon = 2517571096.3399806
cond(A) * epsilon <= 0.5 * 10^(-m)
m = -10.0
```

Obliczenia nie powiodły się. Liczba m powinna wskazywać dodatnią liczbę miejsc po przecinku, które są zgodne z poprawnym wynikiem.

Potwierdza to fakt, że układ jest źle uwarunkowany i niewielka zmiana wartości współczynników znacząco wpływa na wynik.

Wykorzystana metoda:
http://nm.mathforcollege.com/mws/gen/04sle/mws_gen_sle_spe_adequacy.pdf

Zadanie 2.

Dana jest macierz $A = \begin{bmatrix} 1 & 1+\varepsilon \\ 1-\varepsilon & 1 \end{bmatrix}$

a) Ile wynosi wyznacznik A?

Wyznacznik macierzy 2x2 wyznaczam ze wzoru:

$$\det(M) = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} * a_{22} - a_{21} * a_{12}$$

$$\det(A) = \begin{vmatrix} 1 & 1+\varepsilon \\ 1-\varepsilon & 1 \end{vmatrix} = 1 * 1 - (1-\varepsilon) * (1+\varepsilon) = 1 - (1-\varepsilon^2) = \varepsilon^2$$

Czyli wyznacznik jest całkowicie zależny od epsilona maszynowego.

b) Dla jakiego zakresu wartości ε obliczony wyznacznik będzie równy zero w arytmetyce zmiennoprzecinkowej?

Niestety nie znam odpowiedzi.

c) Ile wynoszą macierze L i U będące wynikiem rozkładu LU macierzy A?

W metodzie LU dążymy do zapisania macierzy A jako iloczyn pewnej macierzy dolnej L i górnej U. Przy czym aby rozkład był jednoznaczny, zakładam, że elementy na głównej przekątnej macierzy L są równe 1.

$$A = L * U$$

$$\begin{bmatrix} 1 & 1+\varepsilon \\ 1-\varepsilon & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l & 1 \end{bmatrix} * \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}$$

Wykorzystam metodę Doolittle'a, w której wyznaczanie elementów macierzy L i U robi się naprzemiennie – raz wiersz macierzy U, raz kolumnę L.

1 wiersz macierzy U:

$$\begin{bmatrix} 1 & 1+\varepsilon \\ 1-\varepsilon & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l & 1 \end{bmatrix} * \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}$$

$$1 = u_{11} * 1 + 0 * 0 \rightarrow u_{11} = 1$$

$$1+\varepsilon = u_{12} * 1 + u_{22} * 0 \rightarrow u_{12} = 1+\varepsilon$$

Zaktualizowane macierze:

$$\begin{bmatrix} 1 & 1+\varepsilon \\ 1-\varepsilon & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1+\varepsilon \\ 0 & u_{22} \end{bmatrix}$$

1 kolumna macierzy U:

$$\begin{bmatrix} 1 & 1+\varepsilon \\ 1-\varepsilon & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1+\varepsilon \\ 0 & u_{22} \end{bmatrix}$$

$$1-\varepsilon = 1 * l + 0 * 1 \rightarrow l = 1-\varepsilon$$

Zaktualizowane macierze:

$$\begin{bmatrix} 1 & 1+\varepsilon \\ 1-\varepsilon & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1-\varepsilon & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1+\varepsilon \\ 0 & u_{22} \end{bmatrix}$$

2 wiersz macierzy U:

$$\begin{bmatrix} 1 & 1+\varepsilon \\ 1-\varepsilon & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1-\varepsilon & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1+\varepsilon \\ 0 & u_{22} \end{bmatrix}$$

$$1 = (1-\varepsilon) * (1+\varepsilon) + 1 * u_{22} \rightarrow u_{22} = \varepsilon^2$$

Ostatecznie macierze:

$$\begin{bmatrix} 1 & 1+\varepsilon \\ 1-\varepsilon & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1-\varepsilon & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1+\varepsilon \\ 0 & \varepsilon^2 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 \\ 1-\varepsilon & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} 1 & 1+\varepsilon \\ 0 & \varepsilon^2 \end{bmatrix}$$

d) Dla jakiego zakresu wartości ε obliczona macierz U będzie osobliwa w arytmetyce zmiennoprzecinkowej?

Aby macierz U była osobliwa jej wyznacznik musi wynosić 0.

Wyznacznik macierzy 2x2 wyznaczam ze wzoru:

$$\det(M) = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} * a_{22} - a_{21} * a_{12}$$

$$\det(U) = \begin{vmatrix} 1 & 1+\varepsilon \\ 0 & \varepsilon^2 \end{vmatrix} = 1 * \varepsilon^2 - 0 * (1+\varepsilon) = \varepsilon^2$$

Niestety nie znam odpowiedzi na pytanie.

Zadanie 3.

Pomnożenie obu stron układu równań liniowych $Ax = b$ przez nieosobliwą macierz diagonalna D daje nowy układ $DAx = Db$. Przeskalowanie wierszy układu w teorii nie zmienia rozwiązania układu.

Takie przeskalowanie ma jednak wpływ na współczynnik uwarunkowania macierzy oraz wybór elementów wiodących, a w konsekwencji może wpłynąć na dokładność rozwiązania w arytmetyce o skończonej precyzji (skalowanie może także wprowadzić błędy zaokrągleń, chyba, ze elementy macierzy D są potęgami 2, ogólnie: potęgami podstawy systemu używanego w arytmetyce zmiennoprzecinkowej).

Wybierz losową macierz A i wektor b dla których znane jest dokładne rozwiązanie x. Następnie wykonaj eksperymenty z różnymi macierzami D i zaobserwuj ich wpływ na współczynnik uwarunkowania macierzy DA i rozwiązania układu $DAx = Db$ otrzymane metodą *numpy.linalg.solve*.

Zbadaj macierze D, w których wartości bezwzględne elementów na przekątnej znacznie się różnią (chodzi o zasymulowanie układu z błędnie dobranymi jednostkami fizycznymi).

Porównaj residuum względne oraz błąd względny rozwiązania dla różnych skalowań. Jakie skalowanie daje małą dokładność? Czy residuum pozostaje małe w tym przypadku?

Wybrany układ równań w postaci macierzowej:

$$\begin{bmatrix} 1 & 2 & -1 \\ 3 & 4 & 1 \\ 2 & -2 & 3 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 9 \\ -1 \end{bmatrix}$$

Układ równań w postaci macierzy współczynników przekształcam za pomocą operacji elementarnych metodą Gaussa:

$$\left[\begin{array}{ccc|c} 1 & 2 & -1 & 5 \\ 3 & 4 & 1 & 9 \\ 2 & -2 & 3 & -1 \end{array}\right] \quad w2 - 3 * w1$$

$$\left[\begin{array}{ccc|c} 1 & 2 & -1 & 5 \\ 0 & -2 & 4 & -6 \\ 2 & -2 & 3 & -1 \end{array}\right] \quad w3 - 2 * w1$$

$$\left[\begin{array}{ccc|c} 1 & 2 & -1 & 5 \\ 0 & -2 & 4 & -6 \\ 0 & -6 & 5 & -11 \end{array}\right] \quad w3 * (-1)$$

$$\left[\begin{array}{ccc|c} 1 & 2 & -1 & 5 \\ 0 & -2 & 4 & -6 \\ 0 & 6 & -5 & 11 \end{array}\right] \quad w3 + 3 * w2$$

$$\left[\begin{array}{ccc|c} 1 & 2 & -1 & 5 \\ 0 & -2 & 4 & -6 \\ 0 & 0 & 7 & -7 \end{array}\right]$$

Otrzymałam macierz w postaci schodkowej, na podstawie której układam równania zaczynając od ostatniego wiersza.

$$\begin{cases} 7z = -7 \\ -2y + 4z = -6 \\ x + 2y - z = 5 \end{cases}$$

$$\begin{cases} z = -1 \\ -2y - 4 = -6 \\ x + 2y + 1 = 5 \end{cases}$$

$$\begin{cases} z = -1 \\ y = 1 \\ x + 2 + 1 = 5 \end{cases}$$

$$\begin{cases} z = -1 \\ y = 1 \\ x = 2 \end{cases}$$

Ostateczny wynik:

$$\begin{cases} x = 2 \\ y = 1 \\ z = -1 \end{cases}$$

Kod programu to testowania dla różnych macierzy diagonalnych D:

```
import numpy as np

A = np.array([[1, 2, -1],
              [3, 4, 1],
              [2, -2, 3]])

B = np.array([5, 9, -1])


x = np.linalg.solve(A, B)
print("Wynik:", x)

condA = np.linalg.cond(A)
print("cond(A) =", condA)

############################

D1 = np.array([[1, 0, 0],
               [0, 2/7, 0],
               [0, 0, 1/3]])

x1 = np.linalg.solve(np.dot(D1, A), np.dot(D1, B))
print("\nWynik:", x1)

condD1A = np.linalg.cond(np.dot(D1, A))
print("cond(D1*A) =", condD1A)

############################
```

```
D2 = np.array([[20, 0, 0],
               [0, 40, 0],
               [0, 0, 80]])

x2 = np.linalg.solve(np.dot(D2, A), np.dot(D2, B))
print("\nWynik:", x2)

condD2A = np.linalg.cond(np.dot(D2, A))
print("cond(D2*A) =", condD2A)

############################

D3 = np.array([[2, 0, 0],
               [0, -4048, 0],
               [0, 0, 1024]])

x3 = np.linalg.solve(np.dot(D3, A), np.dot(D3, B))
print("\nWynik:", x3)

condD3A = np.linalg.cond(np.dot(D3, A))
print("cond(D3*A) =", condD3A)

############################

D4 = np.array([[0.00002, 0, 0],
               [0, 0.5, 0],
               [0, 0, 0.009]])

x4 = np.linalg.solve(np.dot(D4, A), np.dot(D4, B))
print("\nWynik:", x4)

condD4A = np.linalg.cond(np.dot(D4, A))
print("cond(D4*A) =", condD4A)
```

Wynik działania:

```
Wynik: [ 2.  1. -1.]
cond(A) = 9.266373996015885

Wynik: [ 2.  1. -1.]
cond(D1*A) = 8.419118491838415

Wynik: [ 2.  1. -1.]
cond(D2*A) = 25.380362499613142

Wynik: [ 2.  1. -1.]
cond(D3*A) = 15481.391400628769

Wynik: [ 2.  1. -1.]
cond(D4*A) = 191213.3252282354
```

Widać, że wartość wskaźnika uwarunkowania w różnych przypadkach potrafiła znacząco zmienić wartość.

Niestety nie udało mi się znaleźć takiej macierzy, aby wyniki różniły się od siebie.