Metody Obliczeniowe w Nauce i Technice
Natalia Bratek, 30.04.2024
Laboratorium 08

Układy równań – metody bezpośrednie

1. Treści zadań

1.1 Napisz program, który:

1. Jako parametr pobiera rozmiar układu równań n
2. Generuje macierz układu A(nxn) i wektor wyrazów wolnych b(n)
3. Rozwiązuje układ równań Ax=b na trzy sposoby:
   a) poprzez dekompozycję LU macierzy A: A=LU;
   b) poprzez odwrócenie macierzy A: $x=A^{-1}b$, sprawdzić czy $AA^{-1}=I$ i $A^{-1}A=I$ (macierz jednostkowa)
   c) poprzez dekompozycję QR macierzy A: A=QR.
4. Sprawdzić poprawność rozwiązania (tj., czy A**x**=b)
5. Zmierzyć całkowity czas rozwiązania układu.
6. Porównać czasy z trzech sposobów: poprzez dekompozycję LU, poprzez *odwrócenie macierzy* i poprzez *dekompozycję QR*

1.2 Zadanie domowe: Narysuj wykres zależności całkowitego czasu rozwiązywania układu (LU, QR, odwrócenie macierzy) od rozmiaru układu równań. Wykonaj pomiary dla 5 wartości z przedziału od 10 do 100.

Uwaga: można się posłużyć funkcjami z biblioteki numerycznej dla danego języka programowania.

2. Rozwiązania

2.1 Zadanie pierwsze

```
void print_vector(gsl_vector *v) {
    for (size_t i = 0; i < v->size; i++) {
        printf("%g ", gsl_vector_get(v, i));
    }
    printf("\n");
}
```

Rys.1 Funkcja print_vector

Funkcja print_vector służy do wypisania elementów wektora z biblioteki GSL na standardowe wyjście. Przechodzi przez wszystkie elementy wektora przy użyciu pętli for, pobiera każdy element za pomocą funkcji gsl_vector_get i wypisuje go.

```
void fill_matrix_and_vector(gsl_matrix *A, gsl_vector *b, int n) {
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            gsl_matrix_set(A, i, j, drand48() * 10.0);
        }
        gsl_vector_set(b, i, drand48() * 10.0);
    }
}
```

Rys.2 Funkcja fill_matrix_and_vector

Funkcja fill_matrix_and_vector inicjalizuje macierz i wektor, używając losowych wartości z zakresu 0 do 10. Używa do tego celu dwóch pętli: jednej dla macierzy i drugiej dla wektora, aby wypełnić je losowymi liczbami.

```c
void solve_lu(gsl_matrix *A, gsl_vector *b, gsl_vector *x) {
    int s;
    gsl_matrix *LU = gsl_matrix_alloc(A->size1, A->size2);
    gsl_matrix_memcpy(LU, A);
    gsl_permutation *p = gsl_permutation_alloc(A->size1);

    gsl_linalg_LU_decomp(LU, p, &s);
    gsl_linalg_LU_solve(LU, p, b, x);

    gsl_matrix_free(LU);
    gsl_permutation_free(p);
}
```

Rys.3 solve_lu

Funkcja solve_lu używa metody dekompozycji LU do rozwiązania układu równań liniowych. Najpierw tworzy kopię macierzy wejściowej, wykonuje dekompozycję na macierze trójkątne L i U, a następnie używa tej dekompozycji do rozwiązania układu równań. Na koniec zwalnia zaalokowaną pamięć.

```c
void solve_inverse(gsl_matrix *A, gsl_vector *b, gsl_vector *x) {
    int s;
    gsl_matrix *invA = gsl_matrix_alloc(A->size1, A->size2);
    gsl_matrix *LU = gsl_matrix_alloc(A->size1, A->size2);
    gsl_permutation *p = gsl_permutation_alloc(A->size1);

    gsl_matrix_memcpy(LU, A);
    gsl_linalg_LU_decomp(LU, p, &s);
    gsl_linalg_LU_invert(LU, p, invA);
    gsl_blas_dgemv(CblasNoTrans, 1.0, invA, b, 0.0, x);

    gsl_matrix_free(invA);
    gsl_matrix_free(LU);
    gsl_permutation_free(p);
}
```

Rys.4 solve_inverse

Funkcja solve_inverse rozwiązuje układ równań liniowych Ax = b poprzez obliczenie odwrotności macierzy A i używa tej odwrotności do obliczenia rozwiązania x. Proces obejmuje dekompozycję LU macierzy A, obliczenie jej odwrotności, a następnie mnożenie odwrotności przez wektor b w celu uzyskania rozwiązania. Po zakończeniu, zwalnia zaalokowane zasoby.

```c
void solve_qr(gsl_matrix *A, gsl_vector *b, gsl_vector *x) {
    gsl_matrix *QR = gsl_matrix_alloc(A->size1, A->size2);
    gsl_vector *tau = gsl_vector_alloc(A->size1);

    gsl_matrix_memcpy(QR, A);
    gsl_linalg_QR_decomp(QR, tau);
    gsl_linalg_QR_solve(QR, tau, b, x);

    gsl_matrix_free(QR);
    gsl_vector_free(tau);
}
```

Rys.5 solve_qr

Funkcja solve_qr rozwiązuje układ równań liniowych *Ax=b* używając metody dekompozycji QR. Najpierw tworzy kopię macierzy *A* w nowej macierzy *QR* i wykonuje dekompozycję QR tej macierzy. Następnie używa wyników dekompozycji do rozwiązania układu równań. Po obliczeniu rozwiązania zwalnia zaalokowane zasoby, czyli pamięć przydzieloną dla macierzy *QR* i wektora tau.

```c
void check_solution(gsl_matrix *A, gsl_vector *x, gsl_vector *b) {
    gsl_vector *Ax = gsl_vector_alloc(b->size);
    gsl_blas_dgemv(CblasNoTrans, 1.0, A, x, 0.0, Ax);

    gsl_vector_sub(Ax, b);

    double norm = gsl_blas_dnrm2(Ax);
    printf("Norma różnicy Ax-b: %g\n", norm);
    if (norm < 1e-10) {
        printf("Rozwiązanie jest poprawne.\n");
    } else {
        printf("Rozwiązanie jest niepoprawne.\n");
    }

    gsl_vector_free(Ax);
}
```

Rys.6 check_solution

Funkcja check_solution sprawdza, czy rozwiązanie *x* układu równań liniowych *Ax=b* jest poprawne. W tym celu najpierw oblicza produkt macierzy *A* i wektora *x*, a następnie porównuje wynik z wektorem *b*, obliczając normę różnicy. Jeśli norma jest mniejsza niż ustalony próg (1e-10), uznaje rozwiązanie za poprawne, w przeciwnym razie za niepoprawne. Na koniec zwalnia zaalokowaną pamięć dla wektora wynikowego.

```c
int main() {
    int n = 10;
    gsl_matrix *A = gsl_matrix_alloc(n, n);
    gsl_vector *b = gsl_vector_alloc(n);
    gsl_vector *x = gsl_vector_alloc(n);

    fill_matrix_and_vector(A, b, n);

    printf("Metoda LU:\n");
    clock_t start = clock();
    solve_lu(A, b, x);
    clock_t end = clock();
    printf("Rozwiązanie: ");
    print_vector(x);
    printf("Czas: %f sekund\n", (double)(end - start) / CLOCKS_PER_SEC);
    check_solution(A, x, b);

    printf("Metoda odwrotności macierzy:\n");
    start = clock();
    solve_inverse(A, b, x);
    end = clock();
    printf("Rozwiązanie: ");
    print_vector(x);
    printf("Czas: %f sekund\n", (double)(end - start) / CLOCKS_PER_SEC);
    check_solution(A, x, b);

    printf("Metoda QR:\n");
    start = clock();
    solve_qr(A, b, x);
    end = clock();
    printf("Rozwiązanie: ");
    print_vector(x);
    printf("Czas: %f sekund\n", (double)(end - start) / CLOCKS_PER_SEC);
    check_solution(A, x, b);

    gsl_matrix_free(A);
    gsl_vector_free(b);
    gsl_vector_free(x);

    return 0;
}
```

Rys.7 main

Funkcja main inicjuje macierz *A* i wektory *b* oraz *x*, a następnie stosuje trzy różne metody rozwiązania układu równań liniowych *Ax=b*: metodę LU, metodę odwracania macierzy oraz metodę QR. Dla każdej metody mierzy czas wykonania, wypisuje obliczone rozwiązanie oraz sprawdza jego poprawność. Na zakończenie zwalnia zaalokowaną pamięć.

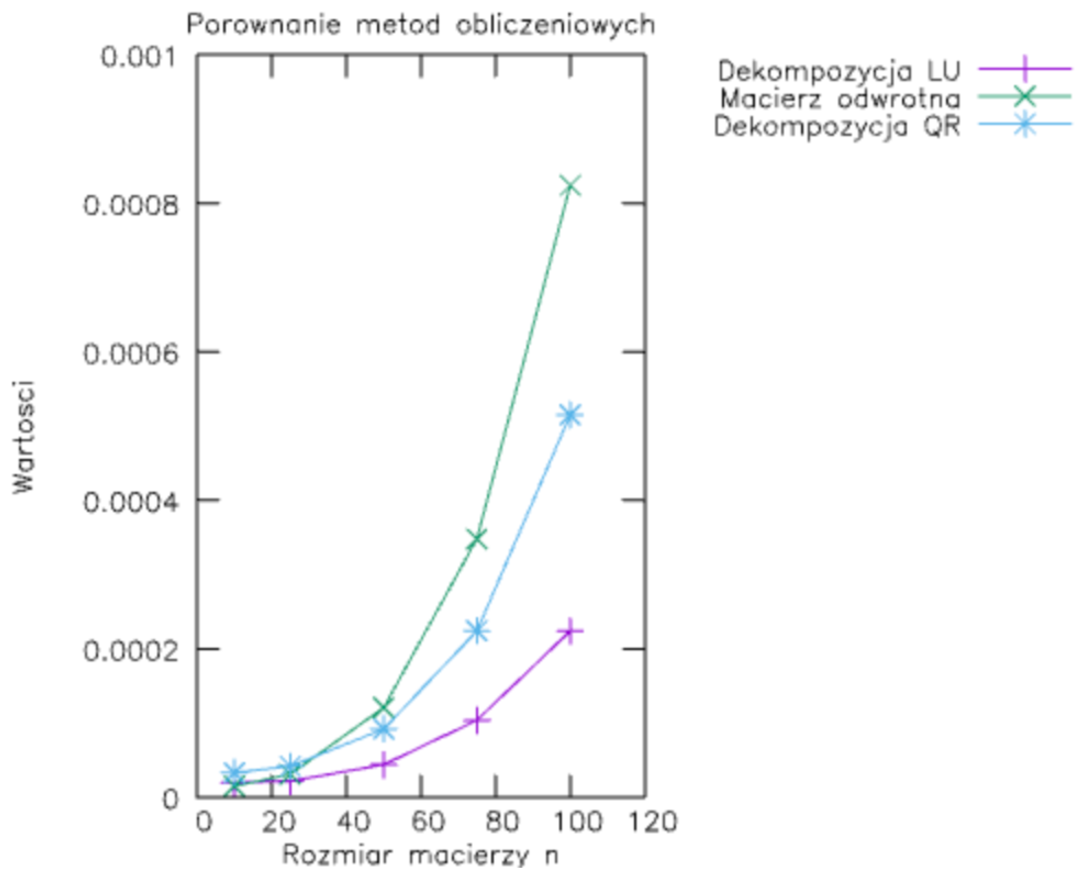
Metoda	n=10	n=25	n=50	n=75	n=100
Dekompozycja LU	0.000020	0.000022	0.000044	0.000104	0.000224
Macierz odwrotna	0.000015	0.000031	0.000121	0.000348	0.000824
Dekompozycja QR	0.000033	0.000042	0.000092	0.000224	0.000515

Tab.1 Wyniki

Tabela przedstawia wyniki dla pięciu pomiarów dla metod: dekompozycji LU, metody macierzy odwrotnej, dekompozycji QR.

2.2 Zadanie drugie

Do wygenerowania wykresu z danymi z tabeli skorzystałam z Gnuplot w wersji online. Wykres przedstawia wartości dla różnych metod w zależności od rozmiaru n.



Rys.8 Wykres

Poniżej umieściłam skrypt, który generuje wykres.

Script:

```
# Ustawienie tytułu wykresu i etykiet osi
set title "Porownanie metod obliczeniowych"
set xlabel "Rozmiar macierzy n"
set ylabel "Wartosci"

# Ustawienie zakresu osi x i y
set xrange [0:120]
set yrange [0:0.001]

# Ustawienie stylu danych
set style data linespoints

# Ustawienie legendy
set key outside right top

# Plotowanie danych
plot "-" using 1:2 title 'Dekompozycja LU' with linespoints, \
     "-" using 1:2 title 'Macierz odwrotna' with linespoints, \
     "-" using 1:2 title 'Dekompozycja QR' with linespoints

# Dane dla Dekompozycji LU
10 0.000020
25 0.000022
50 0.000044
75 0.000104
100 0.000224
e
# Dane dla Macierzy odwrotnej
10 0.000015
25 0.000031
50 0.000121
75 0.000348
100 0.000824
e
# Dane dla Dekompozycji QR
10 0.000033
25 0.000042
50 0.000092
75 0.000224
100 0.000515
e
```

Rys.9 Skrypt

Wnioski:

- Wartości dla wszystkich metod zwiększają się w miarę wzrostu rozmiaru macierzy *n*, ponieważ większe macierze wymagają więcej obliczeń.
- Dekompozycja LU pokazuje umiarkowany wzrost wartości, ale staje się znacząco wyższa dla bardzo dużych macierzy (*n*=100).
- Macierz odwrotna wykazuje najszybszy wzrost wartości, co sugeruje, że ta metoda jest najmniej efektywna z punktu widzenia obliczeniowego, szczególnie dla dużych macierzy. Jest to najczęściej związane z wysoką złożonością obliczeniową potrzebną do obliczenia macierzy odwrotnej.
- Dekompozycja QR również pokazuje wzrost, ale wartości są ogólnie niższe niż dla macierzy odwrotnej, co wskazuje na większą efektywność tej metody w porównaniu z obliczeniem macierzy odwrotnej, choć mniej efektywną niż Dekompozycja LU do *n*=75.
- Dekompozycja LU i QR są ogólnie bardziej efektywne niż obliczenia związane z macierzą odwrotną, zwłaszcza przy obsłudze większych macierzy.

3. Bibliografia

- https://pl.wikipedia.org/wiki/Rozk%C5%82ad_QR
- https://pl.wikipedia.org/wiki/Metoda_LU
- https://pl.wikipedia.org/wiki/Macierz_odwrotna