UARL: numeryczne testowanie własności macierzy idempotentnej i inwolucji, rozkład LU

Tomasz Chwiej

26 listopada 2024

1 Wstęp

1.1 Macierz idempotentna

Załóżmy, że dysponujemy dwoma wektorami w $\mathbb{R}^n$: $\vec{u}$ oraz $\vec{v}$, takimi że ich iloczyn skalarny (wewnętrzny) jest równy

$$c = \vec{v}^T\vec{u} = \vec{u}^T\vec{v} = \begin{bmatrix} u_0 & \dots & u_{n-1} \end{bmatrix} \begin{bmatrix} v_0 \\ \vdots \\ v_{n-1} \end{bmatrix} = 1, \qquad \left(c = \sum_{i=0}^{n-1} u_i v_i \right) \tag{1}$$

Iloczyn zewnętrzny (tensorowy) tych wektorów generuje macierz w $\mathbb{R}^n$

$$A = \vec{u}\vec{v}^T = \begin{bmatrix} u_0 \\ \vdots \\ u_{n-1} \end{bmatrix} \begin{bmatrix} v_0 & \dots & v_{n-1} \end{bmatrix} = \begin{bmatrix} u_0v_0 & \dots & u_0v_{n-1} \\ \vdots & \ddots & \vdots \\ u_{n-1}v_0 & \dots & u_{n-1}v_{n-1} \end{bmatrix}, \qquad (A_{i,j} = u_i v_j) \tag{2}$$

Policzmy iloczyn macierzowy AA

$$AA = \vec{u}\underbrace{\vec{v}^T\vec{u}}_{=1}\vec{v}^T = \vec{u}\vec{v}^T = A \tag{3}$$

$$AA = A \tag{4}$$

Macierz o własności $AA = A$ nazywamy macierzą idempotentną, ma ona szczególne własności

$$Det(A) = c \in \mathbb{R} \tag{5}$$

$$Det(AA) = Det(A)Det(A) = (Det(A))^2 = c^2 \tag{6}$$

$$Det(A) = Det(AA) \quad \rightarrow \quad c^2 = c \quad \rightarrow \quad c(c-1) = 0 \quad \rightarrow \quad Det(A) = 0 \vee 1 \tag{7}$$

Przypadkiem szczególnym jest macierz jednostkowa $Det(I) = 1$, natomiast pozostałe macierze idempotentne mają $Det(A) = 0$. Porównując postać iloczynu skalarnego (wzór 1) z diagonalą macierzy A (równanie 2) od razu zauważamy, że ślad macierzy

$$Tr(A) = \sum_{i=0}^{n-1} A_{i,i} = 1 \tag{8}$$

1.2 Macierz inwolucji

Jeśli dysponujemy macierzą idempotentną A to przy jej pomocy możemy utworzyć macierz inwolucji P (I-macierz jednostkowa)

$$P = 2A - I \tag{9}$$

o własnościach

$$PP = (2A - I)(2A - I) = 4(A^2 - A) + I = I \tag{10}$$
$$PP = I = PP^{-1} \quad \rightarrow \quad P = P^{-1} \tag{11}$$

oraz

$$Det(P) = \pm 1 \tag{12}$$

Szczególna postać macierzy P pozwala łatwo znaleźć rozwiązanie układu równań, niezerowy wyznacznik gwarantuje istnienie rozwiązania nietrywialnego

$$P\vec{x} = \vec{b} \quad \rightarrow \quad \vec{x} = P^{-1}\vec{b} \quad \rightarrow \quad \vec{x} = P\vec{b} \tag{13}$$

które możemy uzyskać też poprzez zastosowanie rozkładu LU. W ramach projektu stworzymy macierze A i P oraz numerycznie sprawdzimy ich własności (kolor czerwony).

2 Zadania do wykonania

1. utworzyć $n = 5$ elementowe wektory $\vec{u}$ oraz $\vec{v}$, elementy definiujemy następująco

$$u_i = i + 1, \qquad i = 0, \ldots, n - 1 \tag{14}$$
$$v_i = (i + 1)^2 - 0.8, \qquad i = 0, \ldots, n - 1 \tag{15}$$

2. unormować wektory $\vec{u}$ i $\vec{v}$ tak aby iloczyn skalarny $\vec{u}^T\vec{v} = 1$

$$\vec{u}^T\vec{v} = s \neq 1 \quad \rightarrow \quad \vec{u} := \frac{\vec{u}}{s} \qquad \text{(normalizacja wektora u)} \tag{16}$$

3. utworzyć macierz idempotentną $A = \vec{u}\vec{v}^T$ (wzór 2), zapisać ją do pliku
4. policzyć iloczyn AA oraz sprawdzić czy $AA - A = 0$?, macierz $AA - A$ zapisać do pliku
5. znaleźć rozkład LU macierzy A przy użyciu procedury GSL (*gsl_linalg_LU_decomp*()), do pliku zapisać elementy diagonalne macierzy U (U_{ii}) oraz wyznacznik macierzy A

$$Det(A) = Det(LU) = \underbrace{Det(L)}_{=1} Det(U) = Det(U) = \prod_{i=0}^{n-1} U_{ii} \tag{17}$$

6. policzyć ślad macierzy A $Tr(A)$, wynik zpisać do pliku
7. utworzyć macierz inwolucji $P = 2A - I$ i zapisać jej postać do pliku
8. sprawdzić czy $PP = I$?, iloczyn PP zapisać do pliku
9. znaleźć rozkład LU macierzy P, obliczyć wyznacznik $Det(P)$ i zapisać go do pliku
10. dla wektora $\vec{b} = [1, 1, 1, 1, 1]$ stanowiącego prawą stronę układu równań $P\vec{x} = \vec{b}$ znaleźć jego rozwiązanie ($\vec{x}$) stosując rozkład LU (najpierw znaleźć rozkład LU macierzy P a później wywołać *gsl_linalg_LU_solve*()) i porównać z rozwiązaniem $\vec{x} = P\vec{b}$, oba rozwiązania zapisać do pliku
11. w raporcie proszę przeanalizować uzyskane wyniki w kontekście własności macierzy A i P

3 Obsługa biblioteki GSL

- Aby wykorzystać GSL-a musimy zlokalizować bibliotekę, np. przy użyciu polecenia *locate* (LINUX)

```
locate gsl |grep gsl_linalg.h
locate gsl |grep libgsl.a
```

Pierwsza instrukcja poda ścieżkę do katalogu z plikami nagłówkowymi (np. /usr/include), a druga do katalogu zawierającego skompilowaną bibliotekę (np. /usr/lib/x86_64-linux-gnu/). Kompilacja kodu C (C++) na Taurusie z uwzględnieniem ścieżek oraz bibliotek

```
gcc -L/usr/lib/x86_64-linux-gnu -I/usr/include  kod.c   -lm -lgsl -lgslcblas
g++ -L/usr/lib/x86_64-linux-gnu -I/usr/include  kod.cpp   -lm -lgsl -lgslcblas
```

Uwaga: podczas kompilacji ważna jest kolejność w jakiej podajemy używane biblioteki - te są odczytywane od prawej do lewej, zmiana kolejności może spwodować błąd

- W programie należy dołączyć plik nagłówkowy zawierający definicje procedur GSL-a z algebry liniowej

```
#include<gsl/gsl_linalg.h>
```

- Pakiet wymaga użycia macierzy i wektorów własnego typu, które tworzymy następująco

```
gsl_vector *x=gsl_vector_alloc(int n)
gsl_matrix *A=gsl_matrix_alloc(int n, int n)
```

gdzie: n jest liczbą elelemntów w wektorze oraz liczbą kolumn/wierszy w macierzy. Zapis wartości liczbowych w komórkach oraz ich odczyt wygląda następująco

```
gsl_vector_set(gsl_vector *a,int i,double value)
gsl_matrix_set(gsl_matrix *A,int i,int j,double value)
double b=gsl_vector_get(gsl_vector *a,int i)
double b=gsl_matrix_get(gsl_matrix *A,int i,int j)
```

- Rozkład LU wyznaczymy przy użyciu procedury

```
gsl_linalg_LU_decomp(gsl_matrix *A, gsl_vector *p, int *signum)
```

po jej wykonaniu dolna macierz trójkatna (poniżej diagonali) zostanie nadpisana elementami maicerzy L, a górna (od diagonali w górę) elementami macierzy U. Po znalezieniu rozkładu możemy przystapić do rozwiązania układu równań liniowych $A\vec{x} = \vec{b}$ (A zawiera teraz w sobie rozkład LU)

```
gsl_linalg_LU_solve(const gsl_matrix *A, const gsl_permutation *p,
                    const gsl_vector * b,gsl_vector * x)
```

4 Przydatne procedury (przyśpieszające wykonanie projektu)

- mnożenie wektora przez macierz: $\vec{y} = A\vec{x}$, $y_i = \sum_{j=0}^{n-1} A_{i,j} \cdot x_j, \quad i = 0, 1, \ldots, n-1$

```
for(i=0;i<n; i++){
        y[i]=0.   - zerujemy komórkę w której gromadzimy wyniki
        for(j=0;j<n; j++){
                y[i]+=A[i][j]*x[j]
        }
}
```

- mnożenie dwóch macierzy $C = A \cdot B$, $C_{i,j} = \sum_{k=0}^{n-1} A_{i,k} B_{k,j}, \quad i, j = 0., 1, \ldots, n-1$

```
for(i=0;i<n; i++){
  for(j=0;j<n; j++){
    //zerujemy komórkę w której gromadzimy wynik sumowania (iloczyn skalarny)
    C[i][j]=0
    for(k=0;k<n; k++){
                C[i][j]+=A[i][k]*B[k][j]
    }
  }
}
```

- mnożenie $A^T A$, $C_{i,j} = \sum_{k=0}^{n-1} A_{k,i} A_{k,j}, \quad i, j = 0., 1, \ldots, n-1$

```
for(i=0;i<n; i++){
  for(j=0;j<n; j++){
    //zerujemy komórkę w której gromadzimy wynik sumowania (iloczyn skalarny)
    C[i][j]=0
    for(k=0;k<n; k++){
            //zmieniliśmy kolejność indeksów w pierwszej macierzy
                C[i][j]+=A[k][i]*A[k][j]
    }
  }
}
```

- mnożenie macierzy kwadratowej i trójkątnej dolnej $B = A\mathcal{L}$ (w $\mathcal{L}$ elementy powyżej przekątnej są zerami więc je pomijamy)

```
for(i=0;i<n; i++){
  for(j=0;j<n; j++){
    //zerujemy komórkę w której gromadzimy wynik sumowania (iloczyn skalarny)
    C[i][j]=0
    for(k=0;k<n; k++){
                if(k>=j)C[i][j]+=A[i][k]*ℒ[k][j]
    }
  }
}
```

- zapis macierzy (w postaci tablicy 2D, to nie jest tablica GSL-a tylko C) do pliku (fp!=NULL) lub wypisanie na ekran (fp=NULL)

```
FILE *fp

if(fp==NULL){
```

```
        for(i=0;i<n;i++){
                for(j=0;j<n;j++){
                        printf("%12.3f\t",S[i][j]);
                }
                printf("\n");
        }
}else if(fp!=NULL){
        for(i=0;i<n;i++){
                for(j=0;j<n;j++){
                        fprintf(fp,"%12.3f\t",S[i][j]);
                }
                fprintf(fp,"\n");
        }
}
```