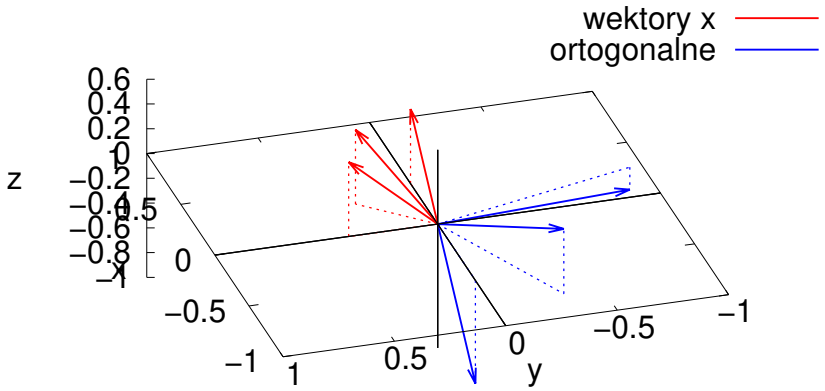
UARL: ortogonalizacja bazy wektorowej przy użyciu rozkładu QR

Tomasz Chwiej

26 listopada 2024

1 Wstęp: macierzowa ortogonalizacja bazy wektorowej, rozkład wektora w bazie



Rysunek 1: Trzy wektory bazowe nieortogonalne (czerwony) oraz po ortogonalizacji bazy (niebieski).

Dowolna macierz $\mathbb{R}^{n\times n} : A$ stanowi operator działający na wektory w przestrzeni $\mathbb{R}^n : \vec{x}$, działanie to polega na przemnożeniu wektora przez macierz $\mathbb{R}^n : \vec{y} = G\vec{x}$, co może odpowiadać obrotowi wektora oraz zmianie jego długości. Problem taki można łatwo przedstawić w 3 wymiarach przy użyciu współrzędnych kartezjańskich. W $\mathbb{R}^3$ możemy zdefiniować macierz obrotu wokół osi "z" o kąt ϕ

$$G = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

i przy jej pomocy oraz zaproponowanego przez nas wektora $\vec{x}_0 = [a, b, c]$ możemy wygenerować trzy wektory

$$\vec{x}_0 \tag{2}$$

$$\vec{x}_1 = G\vec{x}_0 \qquad \text{- wektor obrócony} \tag{3}$$

$$\vec{x}_2 = G^2\vec{x}_0 = G\vec{x}_1 \qquad \text{- wektor dwukrotnie obrócony} \tag{4}$$

Przykład takiej trójki wektorów (kolor czerwony) pokazany jest na rysunku 1, widać że czerwone wektory są liniowo niezależne (tworzą bazę) i nieortogonalne. W wielu zagadnieniach numerycznych

zachodzi konieczność ortogonalizacji bazy wektorowej np. w celu poprawy stabilności numerycznej algorytmu (lub uproszczenia zapisu równań macierzowych). W ramach projektu dokonamy ortogonalizacji bazy przy użyciu rozkładu QR, gdzie Q jest macierzą ortogonalną a R macierzą trójkątną górną. Rozkład QR odpowiada ortogonalizacji Grama-Schmidta. Najpierw utwórzmy macierz A, w której kolejnych kolumnach znajdą się wektory bazy pierwotnej:

$$A = [\vec{x}_0, \vec{x}_1, \vec{x}_2] \tag{5}$$

Dzięki zapisowi macierzowemu możemy łatwo policzyć iloczyny skalarne wektorów $\vec{x}_i$

$$S = A^T A \qquad \text{(T-transpozycja macierzy)} \tag{6}$$

Jeśli wektory bazowe $\vec{x}_i$ są ortogonalne to macierz S jest diagonalna (jednostkowa, gdy są unormowane do 1) w przeciwnym wypadku elementy pozadiagonalne będą różne od zera. Macierz A możemy zapisać jako iloczyn Q i R

$$A = QR \quad \rightarrow \quad AR^{-1} = Q \tag{7}$$

Wystarczy więc przemnożyć macierz A przez R^{-1} aby uzyskać macierz Q, której kolumny są do siebie wzajemnie ortogonalne, to jest nasza nowa baza.

Jeśli dysponujemy pewnym wektorem $\vec{y}$ i chcemy znaleźć jego rzuty na kierunki nieortogonalnych wektorów bazowych $[\vec{x}_0, \vec{x}_1, \vec{x}_2]$ to zapisujemy go w postaci kombinacji liniowej, którą można wyrazić w postaci macierzowej

$$\vec{y} = \sum_{i=0}^{n-1} c_i \vec{x}_i \qquad \vec{x}_k^T \cdot / \quad \text{(k-dowolne)} \tag{8}$$

$$\vec{x}_k^T \vec{y} = \sum_{i=0}^{n-1} c_i \vec{x}_k^T \vec{x}_i \tag{9}$$

$$\text{dla wszystkich k:} \quad A^T \vec{y} = A^T A \vec{c} \qquad (A^T)^{-1} \cdot / \tag{10}$$

$$A\vec{c} = \vec{y} \tag{11}$$

Ostatnie wyrażenie to układ równań, który możemy rozwiązać przy użyciu rozkładu QR

$$A\vec{c} = \vec{y} \tag{12}$$

$$QR\vec{c} = \vec{y} \qquad Q^T \cdot / \qquad (Q^T Q = I) \tag{13}$$

$$R\vec{c} = Q^T \vec{y} = \vec{b} \tag{14}$$

Czyli po wyznaczeniu $\vec{b} = Q^T \vec{y}$ wystarczy zastosować postępowanie odwrotne z macierzą R. W projekcie do znalezienia rozkładu QR wykorzystamy bibliotekę numeryczną GSL.

2 Zadania do wykonania

1. utworzyć macierz obrotu G (wzór 1) dla kąta $\phi = \pi/4$
2. utworzyć wektor $\vec{x}_0 = [0.4, 0.0, 0.6]$ oraz wektory $\vec{x}_1 = R\vec{x}_0$ oraz $\vec{x}_2 = R\vec{x}_1$; wektory zapisać do pliku
3. utworzyć macierz $A = [\vec{x}_0, \vec{x}_1, \vec{x}_2]$
4. utworzyć macierz iloczynów skalarnych $S = A^T A$, macierz S zapisać do pliku
5. znaleźć rozkład QR macierzy A (GSL)
6. wektory kolumnowe Q (nowej bazy) zapisać do pliku

7. sprawdzić ortogonalność nowej bazy obliczając iloczyn $D = Q^T Q$, macierz D zapisać do pliku
8. dla wektora $\vec{y} = [-3, 3, 7]$ znaleźć jego rzut na bazę nieortogonalną w postaci współczynników wektora $\vec{c}$ (GSL)
9. w raporcie, oprócz przedstawienia i analizy danych zebranych w pliku z wynikami, proszę także sporządzić wykres pokazujący wektory bazy pierwotnej i po jej ortogonalizacji

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 QR wyznaczymy przy użyciu procedury

```
gsl_linalg_QR_decomp(gsl_matrix *A,gsl_vector *tau)
```

UWAGA: po jej wykonaniu macierz A zostanie nadpisana, a w jej miejsce zakodowane zostaną macierze Q i R (oraz w wektorze *tau*) ze względu na oszczędność pamięci. Aby uzyskać jawne postaci Q i R musimy użyć procedury

```
gsl_linalg_QR_unpack(const gsl_matrix * A, const gsl_vector * tau,
                     gsl_matrix * Q, gsl_matrix * R)
```

Rozwiązanie układu równań $A\vec{c} = QR\vec{c} = \vec{y}$ (wzór 12) uzyskamy korzystając z procedury

```
gsl_linalg_QR_Rsolve(const gsl_matrix * A, const gsl_vector * y, gsl_vector * c)
```

lub jeśli wykonaliśmy mnożenie $\vec{b} = Q^T\vec{y}$ to rozwiązane dla $R\vec{c} = \vec{b}$ (wzór 14) dostarczy nam procedura

```
gsl_linalg_R_solve(const gsl_matrix * R, const gsl_vector * b, gsl_vector * c)
```

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

```
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");
        }
}
```