METODE NUMERICE: Laborator #3
Transformări ortogonale: Householder și Givens. Algoritmul Gram-Schmidt. Polinoame ortogonale

Titulari curs: *Florin Pop, George-Pantelimon Popescu*

Responsabil Laborator: **Mihaela-Andreea Vasile, Florin Pop**

Obiective Laborator

Obiectivele laboratorului de transformări ortogonale sunt:

- Înțelegerea noţiunii de matrice ortogonală şi vector ortogonal;
- Aplicarea celor 2 metode de transformare ortogonală;
- Implementarea algoritmul Gram-Schmidt;
- Folosirea polinoamelor ortogonale.

Vectori Ortogonali. Matrici Ortogonale

Fie vectorii coloană $x, y \in R^n$ de forma $x = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^T$ și $y = \begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix}^T$. Definim:

1. Produsul scalar:
$$< x, y >= \sum_{i=1}^{n} x_i y_i = y^T x \tag{1}$$

2. Norma Euclidiană:
$$||x||_2 = \sqrt{< x, x >} = \sqrt{x^T x} = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2} \tag{2}$$

Vectorii $u_1, u_2, \dots, u_n \in R^n$ sunt **ortogonali** dacă $< u_i, u_j > = 0$, $\forall i \neq j$. Vectorii sunt **ortonormaţi** dacă sunt ortogonali şi $||u_i||_2 = 1$, $\forall i$. O matrice $H \in R^{n \times n}$ este **ortogonală** dacă coloanele sale sunt **vectori ortonormaţi**. Pentru o matrice H ortogonală: $H^T H = H H^T = I_n$. Avem urmatoarele proprietăţi pentru o matrice ortogonală:

1. $H^{-1} = H^T$;

2. $||Hx||_2 = ||x||_2$;

3. $||H||_2 = 1$;

4. $||HA||_2 = ||A||_2$;

5. $det(H) = \pm 1$.

Transformarea Householder

Metoda propusă iniţial de Alston Scott Householder este folosită pentru a transforma o matrice $A \in R^{n\times n}$ simetrică într-o matrice tridiagonală cu aceleaşi valori proprii. Prezentarea generală a metodei poate fi gasită la adresa: `http://mathfaculty.fullerton.edu/mathews/n2003/HouseholderMod.html`. Algoritmul de calcul este următorul:

Algorithm 1 Transformatrea Householder

procedure HOUSEHOLDER(A)
    $B = A$; ▷ $A \in R^{n\times n}$ simetrică.
    for $k = 1 \dots n$ **do**
        $s = \sqrt{\sum_{i=k+1}^{n} b_{ik}^2}$;
        if $s == 0$ **then**
            continue;
        end if
        $z = \frac{1}{2}\left(1 + sign(b_{k+1,k})\frac{b_{k+1,k}}{s}\right)$;
        for $j = 1 \dots k$ **do**
            $v_j = 0$;
        end for
        $v_{k+1} = \sqrt{z}$;
        for $j = k+2 \dots n$ **do**
            $v_j = \frac{sign(b_{k+1,k})b_{kj}}{2sv_{k+1}}$;
        end for
        $v = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}^T$;
        $H = I_n - 2vv^T$;
        Calculează: $A = HBH$;
        if $k == n-2$ **then**
            break;
        end if
        $B = A$;
    end for
    Return A;
end procedure

În cadrul acestui laborator vom defini, folosind reflectori elementari Householder, transformarea $R = HA$, unde:

1. $H = H_{n-1}H_{n-2}\dots H_2H_1$
2. $A = H^T R$

Un reflector elementar Householder H_p este de forma:

$$H_p = I_n - 2\frac{v_p v_p^T}{v_p^T v_p} \tag{3}$$

unde:

1. $a_p = \begin{bmatrix} a_{1p} & a_{2p} & \cdots & a_{pp} & \cdots & a_{np} \end{bmatrix}^T$ este coloana p din matricea A;

2. $\sigma_p = sign(a_{pp})\sqrt{\sum_{i=p}^{n} a_{ip}^2}$;

3. $v_p = \begin{bmatrix} 0 & 0 & \cdots & v_{pp} & \cdots & v_{np} \end{bmatrix}$ (vector Householder);

4. $v_{pp} = a_{pp} + \sigma_p$;

5. $v_{ip} = a_{ip}$, $\forall i > p$;

6. $\beta_p = \sigma_p v_{pp}$.

Folosind un reflector Householder putem aduce o matrice la forma superior triunghiulară, după metoda umrătoare:

```
function [Q, R] = HA(A)
    [m,n] = size(A);
    H = In;
    for p = 1:min(m-1, n)
        Hp = compute Householder reflector for column p in matrix A
        A = Hp A; ;all elements on column p, lines p+1:m will be 0
        H = Hp H; ;save all Householder transformations
    endfor
    Q = H';
    R = A;
endfunction
```

Datorită formei reflectorilor, înmulţirea A = $H_p A$ se efectuează astfel:

- Coloanele $1 : p-1$ din A rămân neschimbate
- Coloana p din A:
  - Elementele de pe liniile $1 : p-1$ rămân neschimbate;
  - $a_{pp} = -\sigma_p$;
  - $a_{ip} = 0, \forall i > p$;
- Coloanele $j = p+1 : n$ din A:
  - Elementele de pe liniile $1 : p-1$ rămân neschimbate;
  - $a_{ij} = a_{ij} - \tau_j \cdot v_{ip}$, $\forall i > p$, $\tau_j = \frac{\sum_{i=p}^{m} v_{ip} \cdot a_{ij}}{\beta_p}$

Transformarea Givens

Metoda Givens este folosită pentru a descompune o matrice $A \in R^{n \times n}$ astfel: A=$G^T R$, unde:

1. $G = G_{n-1,n-1}G_{n-2,n-1}G_{n-2,n-2} \ldots G_{1n} \ldots G_{13}G_{12}$

2. $R = GA$

O matrice de rotaţie Givens, G_{kl} este folosită pentru a elimina elementul $A(l, k)$ de sub diagonala principală (k < l), şi are forma:

$$G_{kl} = \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \cos\theta & \cdots & -\sin\theta & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \sin\theta & \cdots & \cos\theta & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

Pentru a determina matricea de rotaţie Givens, G_{kl}, vom folosi relaţiile:

1. $\rho = \sqrt{A(k,k)^2 + A(l,k)^2}$
2. $\sin\theta = s = -\frac{A(l,k)}{\rho}$
3. $\cos\theta = c = \frac{A(k,k)}{\rho}$

```
function [Q R b] = givens(A, b)
  % factorizarea QR a unei matrice A prin rotatori Givens
  % Intrari: A = matricea sistemului
  %          b = vectorul termenilor liberi
  % Iesiri: x = vectorul necunoscutelor
  %      Q = matricea factor ortogonala
  %      R = factorul superior triunghiular

    [m n] = size(A);
    Q = eye(m);
    for k = 1 : n
        for l = k+1 : m
            r = sqrt(A(k,k)^2 + A(l,k)^2);
            c = A(k,k)/r;
            s = -A(l,k)/r;

            t = c*A(k,k:n) - s*A(l,k:n);
            A(l,k:n) = s*A(k,k:n) + c*A(l,k:n);
            A(k,k:n) = t;

            u = c*b(k) - s*b(l);
            b(l) = s*b(k) + c*b(l);
            (k) = u;

            t = c*Q(k,1:m) - s*Q(l,1:m);
            Q(l,1:m) = s*Q(k,1:m) + c*Q(l,1:m);
            Q(k,1:m) = t;
        end
    end
    Q = Q';
    R = A;
end
```

Listing 1: Transformarea Givens

Datorită formei matricilor de rotaţie, înmulţirea x $= G_{kl} \cdot x$, x vector coloană, se efectuează astfel:

- $x(k) = c \cdot x(k) - s \cdot x(l)$;
- $x(l) = s \cdot x(k) + c \cdot x(l)$;
- Restul elementelor rămân neschimbate.

Algoritmul Gram-Schmidt

Vom considera relaţia $QR = A$ cu necunoscutele q_i şi $r_{ij}, (i <= j)$, unde:

- a_i este coloana i din A;
- q_i este coloana i din Q;
- $r_{ij}, i \leq j$, elementele din R de deasupra diagonalei principale (dacă $i > j$ atunci $r_{ij} = 0$).

$$\begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & r_{nn} \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix}$$

Pentru $j = 1 : n$, aplicăm formulele:

1. $r_{ij} = q_i^T a_j, \quad i = 1 : j-1$;
2. $aux = a_j - \sum_{i=1}^{j-1} r_{ij} q_i$;
3. $r_{jj} = ||aux||_2$;
4. $q_j = \frac{aux}{r_{jj}}$.

Algoritmul Gram-Schmidt Modificat

Algoritmul Gram-Schmidt clasic prezintă o stabilitate numerică slabă. Propunem varianta puțin modificată:

Pentru $i = 1 : n$, aplicăm formulele:

1. $r_{ii} = ||a_i||_2$;
2. $q_j = \frac{a_i}{r_{ii}}$;
3. Pentru $j = i+1 : n$, aplicăm formulele:
   (a) $r_{ij} = q_i^T a_j$;
   (b) $a_j = a_j - q_i r_{ij}$.

Polinoame ortogonale

Polinoamele $p_0, p_1, \ldots, p_n$ reprezintă o bază de polinoame ortogonale, dacă:

1. $< p_i, p_j >= 0, \quad \forall i \neq j$;
2. $||p_i|| = 1, \quad \forall i$;
3. $< p_i, p_j >= \int_a^b p_i(x) p_j(x) w(x) dx$.

Observaţie: Un polinom ortogonal este definit prin:

1. relaţia de recurenţă;
2. intervalul pe care este definit ([a, b]);
3. valorile iniţiale şi funcţia pondere ($w(x)$ - folosită la produsul scalar).

Proprietăţi

1. Orice polinom ortogonal are radacinile in [a,b] reale şi distincte;
2. Orice polinom ortogonal este ortogonal cu orice polinom de grad mai mic decât el.

Exemple de polinoame ortogonale

1. **Cebâşev**: $T_{n+1} - 2xT_n + T_{n-1} = 0, \quad T_0 = 1, \quad T_1 = x; \quad (-1, 1); \quad w(x) = \frac{1}{\sqrt{1-x^2}};$
2. **Legendre**: $(n+1)L_{n+1} - (2n+1)xL_n + nL_{n-1} = 0, \quad L_0 = 1, \quad L_1 = x; \quad [-1, 1]; \quad w(x) = 1;$
3. **Laguerre**: $G_{n+1} - (2n+1-x)G_n + n^2G_{n-1} = 0, \quad G_0 = 1, \quad G_1 = 1 - x; \quad [0, \infty); \quad w(x) = e^{-x};$
4. **Hermite**: $H_{n+1} - 2xH_n + 2nH_{n-1} = 0, \quad H_0 = 1, \quad H_1 = 2x; \quad (-\infty, \infty); \quad w(x) = e^{-x^2}.$

Problema 1

Rulaţi exemplul pentru Algoritmul Gram-Schmidt şi implementaţi Algoritmul Gram-Schmidt modificat.

```
function [Q, R] = Gram_Schmidt(A)
  [n n] = size(A);
  Q = zeros(n);
  R = zeros(n);

  for j = 1:n
    for i = 1:j-1
      R(i,j) = Q(:, i)' * A(:,j);
    endfor

    s = zeros(n, 1);
    for i = 1:j-1
      s = s + R(i, j) * Q(:, i);
    endfor
    %%% Echivalent pentru instructiunea for de mai sus:
    %%  s = Q(:, 1:j-1) * R(1:j-1, j);

    aux = A(:, j) - s;

    R(j, j) = norm(aux, 2);
    Q(:, j) = aux/ R(j, j);
  endfor
endfunction
```

Listing 2: Algoritmul Gram-Schmidt

Problema 2

Definiţi o funcţie pentru calculul coeficienţilor unui polinom ortogonal de grad n. Selecţia polinomului se face printr-un parametru şir de caractere care poate avea valorile: '*cebasev*', '*legendre*', '*laguerre*' sau '*hermite*', folosind relaţiile de recurenţă de mai sus. Funcţia va avea antetul:

```
function [p] = poliOrtogonal(nume_polinom, n)
```

Operații utile cu polinoame în Octave:

```
function test_poly
    %%% Coeficientii polinomului p
    p = [ 2 1 -1]

    %%% Afisarea polinomului
    polyout(p, 'x')

    %%% Coeficientii polinomului q
    q = [ 1 2]

    %%% Afisarea polinomului
    polyout(q, 'x')

    %%% Produsul dintre p si q
    r = conv(p,q)

    %%% Afisarea rezultatului
    polyout(r, 'x')
endfunction
```

Listing 3: Polinoame în Octave

Problema 3

Definiţi o funcţie pentru calculul valorii unui polinom ortogonal de grad n într-un punct dat. Selecţia polinomului se face printr-un parametru şir de caractere care poate avea valorile: '*cebasev*', '*legendre*', '*laguerre*' sau '*hermite*'. Funcţia va avea antetul:

```
function val = evalPoliOrtogonal(nume_polinom, n, a)
```

Hint: Folosiți funcția `polyval`.

Problema 4

Se consideră vectorii $u, v \in R^n$ ortonormaţi ($||u||_2 = 1$, $||v||_2 = 1$, $u^T v = v^T u = 0$). Se formează vectorul $x = u + v$.

1. Să se dea exemplu de doi vectori ortonormaţi;

2. Să se calculeze $||x||_2$;
3. Se formează matricea $H = I_n - xx^T$. Să se calculeze Hu, Hv şi $||H||_2$;
4. Dacă $A = uv^T$, calculaţi $B = H^{-n}AH^n$.

Problema 5

Implementaţi transformarea Householder pentru o matrice A.

1. Implementaţi o funcţie care primeşte un vector x şi un index p, şi calculează parametrii σ, v_p, β definiţi mai sus:
   `function [vp, sigma, beta] = GetHSReflector(x, p)`
2. Implementaţi o funcţie care primeşte un vector x, un index p şi parametrul sigma, şi calculează transformarea Householder aplicată asupra vectorului, considerând că acest vector a fost folosit la calculul parametrilor (coloana p din A).
   `function x = ApplyHSToPColumn(x, p, sigma)`
3. Implementaţi o funcţie care primeşte un vector oarecare x, un index p, vectorul Householder v_p şi parametrul beta, şi calculează transformarea Householder aplicată asupra vectorului(coloanele $p+1:n$ din A).
   `function x = ApplyHSToRandomColumn(x, vp, p, beta)`
4. Implementaţi funcţia `function [Q, R] = Householder(A)`, folosind funcţiile definite mai sus.

Problema 6

Să se determine descompunerea QR pentru matricea $A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$ folosind transformarea Givens.