Seminar 3

Problema celor mai mici pătrate

Acest seminar este dedicat metodelor numerice pentru rezolvarea unei probleme numerice foarte importante întâlnită în multe aplicaţii, aşa numita problemă a celor mai mici pătrate. Problema celor mai mici pătrate constă, în mare, în minimizarea normei euclidiene a unui vector în funcţie de unele variabile. Când tipul dependenţei este unul liniar avem de-a face cu problema liniară a celor mai mici pătrate.

3.1 Preliminarii

Fie un sistem de m ecuaţii liniare cu n necunoscute şi $m > n$. Un astfel de sistem poate fi scris într-o formă concisă:

$$Ax = b, \qquad A \in \mathbb{R}^{m\times n}, \quad b \in \mathbb{R}^m, \quad x \in \mathbb{R}^n.$$

În general acest sistem nu are soluţie. Problema celor mai mici pătrate (CMMP) constă în găsirea vectorului $x \in \mathbf{R}^n$ astfel încât $\|b - Ax\|$ să fie minim, unde $A \in \mathbf{R}^{m\times n}$ şi $b \in \mathbf{R}^m$ sunt date şi $\|\cdot\|$ este norma euclidiană. Dacă un astfel de vector există, atunci el este numit (pseudo-)soluţia sistemului supradeterminat $Ax = b$. În geometria euclidiană a spaţiului liniar $\mathbb{R}^m$ problema CMMP este strâns legată de conceptul de ortogonalitate. Într-adevăr, dacă x^* este soluţia CMMP, atunci $\|b - Ax^*\|$ este distanţa minimă între punctul definit de către vectorul b şi punctele subspaţiului ImA şi această distanţă este definită pe perpendiculara la ImA, vezi fig. 3.1.

Dacă A are coloane independente atunci soluţia CMMP este

$$x = A^{\#}b = (A^TA)^{-1}A^Tb,$$

unde $A^{\#} = (A^TA)^{-1}A^T$ este *o inversă la stânga* numită *pseudo-inversa Moore-Penrose* a lui A. Ca şi în cazul determinat, această formulă nu este recomandată pentru calculul lui x. Cea mai bună metodă se bazează pe transformările ortogonale care sunt singurele transformări ce conservă ortogonalitatea şi norma euclidiană. Asemenea transformări pot reduce problema CMMP la rezolvarea unui sistem liniar superior triunghiular. Pentru detalii referitoare la problema CMMP a se vedea cursul şi bibliografia. Aici vom insista asupra unor recomandări privitoare la metodele numerice pentru rezolvarea problemei mai sus menţionate. De asemenea, vom arăta cum se pot rezolva pe calculator nişte probleme înrudite, cum ar fi calculul bazelor ortogonale pentru subspaţii liniare ale spaţiului vectorial R^m.

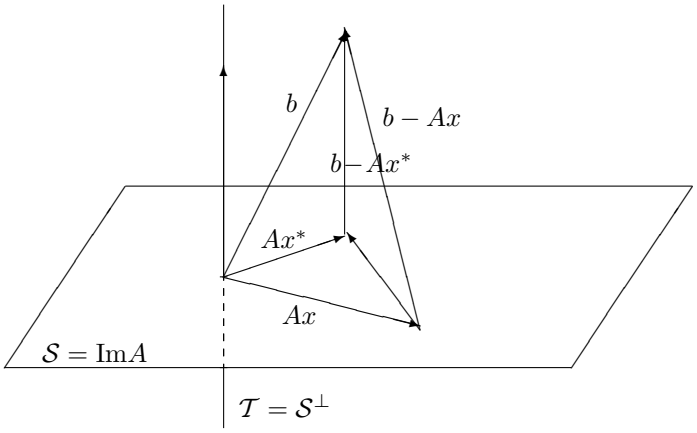

Figure 3.1: Ilustrarea geometrică a problemei celor mai mici pătrate.

1. Nu folosiţi formula de mai sus $x = (A^TA)^{-1}A^Tb$ pentru a calcula pseudo-soluţia sistemelor liniare supradeterminate. De asemenea, nu folosiţi rezolvarea aşa numitului sistem normal de ecuaţii $A^TAx = A^Tb$; există metode mai bune, i.e. metode numerice mai eficiente şi mai precise pentru rezolvarea problemei CMMP.

2. Cea mai bună metodă pentru rezolvarea unui sistem liniar supradeterminat este reducerea sa la un sistem triunghiular prin folosirea transformărilor; transformările ortogonale păstrează norma euclidiană şi, astfel, şi soluţia CMMP.

3. În mod normal, transformările ortogonale sunt construite ca o secvenţă de transformări ortogonale elementare; există două tipuri de transformări ortogonale elementare: *reflectori* (transformări Householder) şi *rotaţii* (transformări Givens).

4. Procedura principală folosită pentru rezolvarea problemei CMMP este triangularizarea ortogonală (vezi algoritmul ORTHT) a matricei coeficienţilor A.

5. Triangularizarea ortogonală a unei matrice A duce imediat la aşa numita factorizare QR a lui A, i.e. $A = Q * R$ unde Q este ortogonală iar R este superior triunghiulară.

6. Fie $a_1, a_2, \cdots, a_n$ o bază a subspaţiului liniar $\mathcal{S} \subset \mathbb{R}^m$ şi definim matricea $A = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \in \mathbb{R}^{m\times n}$. Dacă $A = Q * R$ este factorizarea QR a matricei A, atunci coloanele lui $Q(:, 1 : n)$ formează o bază ortonormală a subspaţiului $\mathcal{S} = \mathrm{Im}A$. De asemenea, coloanele matricei $Q(:, n + 1 : m)$ formează o bază ortonormală a subspaţiului $\mathcal{T} = \mathrm{Ker}A^T$, complementul ortonormal al lui $\mathcal{S}$ în $\mathbb{R}^m$ (vezi problema rezolvată 3.9).

7. Nu calculaţi pseudo-inversele matricelor fară o cerinţă explicită de a face acest lucru. Astfel de expresii precum $\alpha = c^TA^{\#}b$ pot fi calculate fară calculul lui $A^{\#}$.

8. Când un sistem liniar are mai multe necunoscute decât ecuaţii ($m < n$) se numeşte sistem liniar *subdeterminat*. În general, un sistem liniar subdeterminat $Ax = b$ are un număr infinit de soluţii. A rezolva un astfel de sistem în sensul CMMP înseamnă a

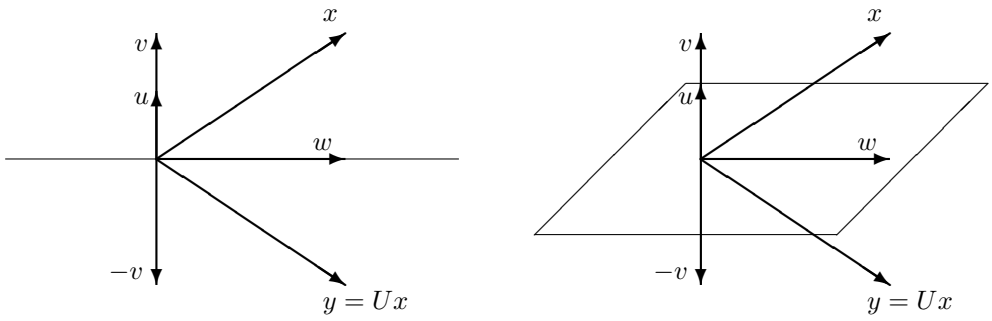

Figure 3.2: Interpretarea geometrică a unei reflecţii ortogonale în $\mathbb{R}^2$ (a) şi $\mathbb{R}^3$ (b).

calcula soluţia *normală*, care este soluţia de normă euclidiană minimă, i.e. x^* astfel încât

$$\|x^*\| = \min_{Ax=b} \|x\|.$$

Dacă A are linii independente atunci soluţia normală este unică şi este dată de

$$x^* = A^T * (A * A^T)^{-1} b,$$

unde $A^+ = A^T * (A * A^T)^{-1}$ este *o inversă la dreapta* numită *pseudo-inversa normală* a lui A. Desigur, această formulă nu este o metodă numerică bună pentru a calcula soluţia normală. Metodele recomandate se bazează pe transformări ortogonale şi reducerea sistemului la unul inferior triunghiular (vezi cursul pentru detalii).

3.2 Probleme rezolvate

3.2.1 Transformări ortogonale

Problema 3.1 Fie un reflector elementar $U = I_m - 2uu^T$, $\|u\| = 1$. Fie $x \in \mathbb{R}^m$ un vector dat şi $x = v + w$ o descompunere a lui x, unde v este coliniar cu u (i.e. $v = \gamma u$) şi w este ortogonal faţă de u (i.e. $u^T w = 0$). Arătaţi că $Ux = -v + w$ şi daţi o interpretare geometrică numelui *reflector* pentru matricea U.

Rezolvare. Avem $Ux = x - 2uu^T x = v + w - 2uu^T(v + w) = v - 2uu^T v + w - 2uu^T w = v - 2uu^T v + w$. Dacă $\gamma = 0$, i.e. $v = 0$, atunci $Ux = w = -v + w$. Dacă $\gamma \neq 0$, atunci $u^T v = \gamma u^T u = \gamma$ and $2uu^T v = 2\frac{v}{\gamma}\gamma = 2v$ şi, astfel, $Ux = -v + w$. Interpretarea geometrică este aceea că vectorul $y = Ux$ este imagine (i.e. o *reflecţie*) a lui x într-o "(hiper)oglindă" (definită de subspaţiul liniar perpendicular pe vectorul u). Vezi fig. 3.2 pentru $m = 2$ şi $m = 3$.

Problema 3.2 a. Arătaţi că orice matrice ortogonală triunghiulară este diagonală.

b. Arătaţi că orice matrice ortogonală poate fi scrisă ca un produs de reflectori elementari.

Rezolvare. a. Dacă o matrice ortogonală $Q \in \mathbb{R}^{m\times m}$ este superior triunghiulară, atunci inversa sa este Q^T şi este superior triunghiulară. Deci, Q este în acelaşi timp superior şi inferior triunghiulară, deci Q este diagonală. Cazul în care Q este inferior triunghiulară se tratează similar. În mod evident o matrice diagonală ortogonală are elementele diagonale egale cu 1 sau cu -1. **b.** Fie Q o matrice ortogonală şi să-i aplicăm algoritmul de triunghiularizare ortogonală ORTHT. Vom avea

$$U_{m-1}\cdots U_2U_1Q = R$$

unde U_k este un reflector elementar sau $U_k = I_m$. Evident, putem alege reflectori elementari folosiţi astfel încât toţi $R(k,k)$ să fie pozitivi. Dar matricea superior triunghiulară R este ortogonală ca un produs de matrice ortogonale. Conform punctului a) R este diagonală şi are elemente diagonale pozitive, i.e. $R = I_m$. Astfel,

$$Q = U_1^{-1}U_2^{-1}\cdots U_{m-1}^{-1}R = U_1U_2\cdots U_{m-1},$$

i.e. Q este un produs de reflectori elementari. În concluzie, orice transformare ortogonală poate fi exprimată ca o secvenţă de reflectori elementari, i.e. o secvenţă de transformări Householder.

Problema 3.3 a. Se consideră vectorul $x \in \mathbf{R}^n$ cu norma euclidiană egală cu 1. Găsiţi o matrice $Y \in \mathbf{R}^{n\times(n-1)}$ cu coloane ortogonale (i.e. care să satisfacă $Y^TY = I_{n-1}$), astfel încât $x^TY = 0$ (i.e. matricea $[\,x\ Y\,]$ să fie ortogonală).

b. Se consideră doi vectori ortogonali daţi $x_1, x_2 \in \mathbf{R}^n$ de normă euclidiană unitate. Găsiţi o matrice $Y \in \mathbf{R}^{n\times(n-2)}$ cu coloane ortogonale (i.e. care să satisfacă $Y^TY = I_{n-2}$) astfel încât $x_1^TY = 0$ şi $x_2^TY = 0$(i.e. matricea $[\,x_1\ x_2\ Y\,]$ să fie ortogonală).

c. Generalizaţi cele două proceduri de mai sus prin calculul unei matrice de completare $Y \in \mathbf{R}^{n\times(n-k)}$ cu coloane ortogonale la o matrice dată cu coloane ortogonale $X \in \mathbb{R}^{n\times k}$, i.e. astfel încât matricea $[\,X\ Y\,]$ să fie ortogonală.

Rezolvare. a. Vom construi o matrice ortogonală Q având x_1 ca prima sa coloană. Astfel, soluţia va fi $Y = Q(:, 2:n)$. Fie $U_1 = I_n - uu^T/\beta$ reflectorul care introduce zerouri pe poziţiile $2:n$ ale lui x, i.e. pentru care $(U_1x)(2:n) = 0$. După cum este cunoscut (vezi cursul) un astfel de reflector poate fi definit de către vectorul u cu $u_1 = x_1 - \|x\| = x_1 - 1$, $u(2:n) = x(2:n)$ şi $\beta = \|u\|^2/2$. Vom avea $u^Tx = (x_1-1)x_1 + \sum_{i=2}^n x_i^2 = 1 - x_1$ şi $\beta = u^Tu/2 = \frac{1}{2}\left[(x_1-1)^2 + \sum_{i=2}^n x_i^2\right] = 1 - x_1$. Prin urmare $U_1x = x - \frac{1}{\beta}uu^Tx = x - \frac{1}{1-x_1}(1-x_1)u = e_1$ şi deci $x = U_1^Te_1 = U_1e_1$. În concluzie, matricea Q este reflectorul U_1 mai sus definit şi matricea $Y = Q(:, 2:n)$ poate fi calculată după cum urmează:

1. $u_1 = x_1 - 1$
2. $\beta = 1 - x_1$
3. **pentru** $i = 2:n$

1. $u_i = x_i$
4. **pentru** $i = 1 : n$
   1. **pentru** $j = 1 : n - 1$
      1. $Y(i, j) = -u_i u_{j+1}/\beta$
5. **pentru** $i = 2 : n$
   1. $Y(i, i-1) = 1 + Y(i, i-1)$

b. După cum am procedat mai sus, vom construi o matrice ortogonală Q având $\pm x_1$ şi $\pm x_2$ ca primele sale coloane. Deci soluţia va fi $Y = Q(:, 3 : n)$. Definim matricea $X = [\, x_1 \; x_2 \,]$ şi fie $X = QR$ factorizarea QR a lui X, unde $R = \begin{bmatrix} R' \\ 0 \end{bmatrix}$ cu $R' \in \mathbb{R}^{2\times 2}$ este superior triunghiulară. Dar $R^T R = R'^T R' = X^T Q Q^T X = I_2$. Astfel R' va fi atât superior triunghiulară cât şi ortogonală, i.e. diagonală (vezi problema 2) cu elementele diagonale ± 1. Prin urmare $R = [\, \pm e_1 \;\; \pm e_2 \,]$. În mod evident, toate coloanele lui $Y = Q(:, 3 : n)$ sunt ortogonale cu x_1 şi x_2 iar Y poate fi calculată după cum urmează:

1. Se calculeaza factorizarea QR a matricei X, i.e. $X = [\, x_1 \; x_2 \,] = QR$
2. $Y = Q(:, 3 : n)$

c. Schema de calcul de mai sus poate fi aplicată oricărei matrice $X \in \mathbb{R}^{n\times k}$ cu coloanele ortogonale. Dacă $X = QR$ este factorizarea QR a lui X, atunci $R = [\, \pm e_1 \;\; \pm e_2 \;\; \cdots \;\; \pm e_k \,]$. Prin urmare $Y = Q(:, k + 1 : n)$.

Problema 3.4 Fie $x, y \in \mathbf{R}^n$ doi vectori cu norme euclidiene egale, $\|x\| = \|y\| = \rho$. Construiţi algoritmi pentru:

a. calculul unei matrice ortogonale $Q \in \mathbf{R}^{n\times n}$ astfel încât $Qx = y$. **b.** calculul unei matrice ortogonale $W \in \mathbf{R}^{n\times n}$ astfel încât $Wx \perp y$.

Rezolvare. a. Fie U_1 şi V_1 doi reflectori astfel încât $U_1 x = -\text{sign}(x_1)\rho e_1$ şi $V_1 y = -\text{sign}(y_1)\rho e_1$. Prin urmare, $\text{sign}(x_1) U_1 x = \text{sign}(y_1) V_1 y = -\rho e_1$ şi $\text{sign}(y_1)\text{sign}(x_1) V_1 U_1 x = y$, unde $Q = \text{sign}(y_1)\text{sign}(x_1) V_1 U_1$ este în mod evident o matrice ortogonală. Pentru calculul $V_1 U_1$ fie $U_1 = I_n - uu^T/\beta$ şi $V_1 = I_n - vv^T/\gamma$. Vom avea, $V_1 U_1 = (I_n - vv^T/\gamma)(I_n - uu^T/\beta) = I_n - uu^T/\beta - vv^T/\gamma + \tau uv^T$, where $\tau = v^T u/(\beta\gamma)$. Un algoritm pentru calculul lui Q este:

1. $\rho = \sqrt{\sum_{i=1}^n x_i^2}$
2. $\sigma = \text{sign}(x_1)\rho$
3. $u_1 = x_1 + \sigma$
4. $\beta = u_1 \sigma$
5. **pentru** $i = 2 : n$
   1. $u_i = x_i$
6. $\sigma = \text{sign}(y_1)\rho$
7. $v_1 = y_1 + \sigma$
8. $\gamma = v_1 \sigma$
9. **pentru** $i = 2 : n$
   1. $v_i = y_i$
10. $\tau = (\sum_{i=1}^n v_i u_i)/(\beta\gamma)$
11. **pentru** $i = 1 : n$
    1. **pentru** $j = 1 : n$

1. $Q(i,j) = -u_i u_j/\beta - v_i v_j/\gamma + \tau v_i u_j$
2. $Q(i,i) = 1 + Q(i,i)$

b. Prima coloană a lui V_1 are aceeaşi direcţie cu y ($V_1 e_1 = \pm y/\rho$). Considerăm o altă coloană w a matricei V_1 şi fie vectorul $z = \rho w$; este evident că $z \perp y$ şi $\|x\| = \|z\| = \rho$. Atunci, folosim algoritmul de la punctul **a** pentru calculul unei matrice ortogonale W astfel încât $Wx = z$; prin urmare $Wx \perp y$.

3.2.2 Triangularizarea ortogonală. Factorizarea QR.

Problema 3.5 Fie $H \in \mathbf{R}^{(n+1)\times n}$ o matrice superior Hessenberg ($h_{ij} = 0$ pentru $i > j+1$). Construiţi un algoritm eficient pentru triangularizarea ortogonală a lui H. Folosiţi transformările Householder şi Givens şi comparaţi cele două versiuni. Calculaţi factorizarea QR a unei matrice superior Hessenberg.

Rezolvare. Adaptarea algoritmului $ORTHT$ pentru triangularizarea ortogonală a lui H cu reflectori Householder $U_k = I - (u_k u_k^T)/\beta_k$ ia în considerare faptul că la fiecare pas trebuie sa transformăm în zero un singur element subdiagonal. Prin urmare, vectorul u_k ce îl defineşte pe U_k va avea forma $u_k^T = [0 \dots u_{kk}\ u_{k+1,k}\ 0 \dots 0]$. Algoritmul este:

1. **pentru** $k = 1 : n$
   1. $\sigma \leftarrow \text{sign}(h_{kk})\sqrt{h_{kk}^2 + h_{k+1,k}^2}$
   2. $u_{kk} \leftarrow h_{kk} + \sigma$
   3. $u_{k+1,k} \leftarrow h_{k+1,k}$
   4. $\beta_k \leftarrow u_{kk}\sigma$
   5. $h_{kk} \leftarrow r_{kk} = -\sigma$
   6. **pentru** $j = k+1 : n$
      1. $\tau \leftarrow (u_{kk}h_{kj} + u_{k+1,k}h_{k+1,j})/\beta_k$
      2. $h_{kj} \leftarrow h_{kj} - \tau u_{kk}$
      2. $h_{k+1,j} \leftarrow h_{k+1,j} - \tau u_{k+1,k}$

Numărul de flopi necesar pentru acest algoritm este de $N_{fl} \approx 4n^2$ şi este nevoie şi de n extrageri de radical. Algoritmul de mai sus calculează matricea superior triunghiulară $R = U_n \cdots U_2 U_1 Z H$. Prin urmare factorizarea QR a lui H va fi $H = U_1 U_2 \cdots U_n R = QR$, unde $Q = U_1 U_2 \cdots U_n$. Pentru calculul lui Q putem folosi una dintre următoarele două scheme de calcul:

1. $Q = I_{n+1}$
2. **pentru** $k = 1 : n$
   1. $Q \leftarrow QU_k$

sau

1. $Q = I_{n+1}$
2. **pentru** $k = n : -1 : 1$
   1. $Q \leftarrow U_k Q$

care trebuie să exploateze structura superioară Hessenberg a lui Q (de demonstrat).

Folosind sintaxa $[U, beta] = ORTHT\_H(H)$ pentru algoritmul de triangularizare a unei matrice superior Hessenberg şi prima schemă de calcul, factorizarea QR a lui H poate fi calculată prin următorul algoritm.

1. $[U, beta] = ORTHT\_H(H)$
2. **pentru** $i = 1 : n+1$
   1. **pentru** $j = 1 : n+1$
      1. $q_{ij} = 0$
   2. $q_{ii} = 1$
2. **pentru** $k = 1 : n$
   1. **pentru** $i = 1 : k+1$
      1. $\tau = (q_{ik}u_{kk} + q_{i,k+1}u_{k+1,k})/\beta_k$
      2. $q_{ik} \leftarrow q_{ik} - \tau u_{kk}$
      3. $q_{i,k+1} \leftarrow q_{i,k+1} - \tau u_{k,k+1}$

Numărul suplimentar necesar de flopi este de $N_{fl} \approx 4n^2$.

Desigur, problema poate fi rezolvată şi prin folosirea rotaţiilor Givens. În mod evident la fiecare pas va fi folosită o singură rotaţie pentru a elimina unicul element subdiagonal nenul. Prin urmare, următorul algoritm va calcula matricea superior triunghiulară: $R = P_{n,n+1} \cdots P_{23}P_{12}H$.

1. **for** $k = 1 : n$
   1. $\rho \leftarrow \sqrt{h_{kk}^2 + h_{k+1,k}^2}$
   2. $c_k \leftarrow h_{kk}/\rho$
   3. $s_k \leftarrow h_{k+1,k}/\rho$
   4. $h_{kk} \leftarrow r_{kk} = \rho$
   5. $h_{k+1,k} \leftarrow 0$
   6. **pentru** $j = k+1 : n$
      1. $\tau \leftarrow c_k h_{kj} + s_k h_{k+1,j}$
      2. $h_{k+1,j} \leftarrow -s_k h_{kj} + c_k h_{k+1,j}$
      3. $h_{kj} \leftarrow \tau$

Numărul de flopi necesar acestui al doilea algoritm este de $N_{fl} \approx 3n^2$ şi n extrageri de radical. Aşadar, a doua variantă este mai bună (situaţia inversă se întâlneşte în cazul matricelor generale).

Factorizarea QR a lui H bazată pe algoritmul de mai sus va calcula în plus matricea $Q = P_{12}^T P_{23}^T \cdots P_{n,n+1}^T$. Folosind sintaxa $[c, s, R] = ORTHT\_H\_ROT(H)$ pentru algoritmul de triangularizare a unei matrice superior Hessenberg cu rotaţii şi schema de calcul

1. $Q = I_{n+1}$
2. **pentru** $k = 1 : n$
   1. $Q \leftarrow QP_{k,k+1}^T$

factorizarea QR a lui H poate fi calculată cu ajutorul următorului algoritm:

1. $[c, s, R] = ORTHT\_H\_ROT(H)$
2. **pentru** $i = 1 : n+1$

1. **pentru** $j = 1 : n+1$
   1. $q_{ij} = 0$
   2. $q_{ii} = 1$
2. **pentru** $k = 1 : n$
   1. **pentru** $i = 1 : k+1$
      1. $\tau = q_{ik}c_k + q_{i,k+1}s_k$
      2. $q_{i,k+1} \leftarrow -q_{ik}s_k + q_{i,k+1}c_k$
      3. $q_{i,k} \leftarrow \tau$

Numărul suplimentar necesar de flopi este de $N_{fl} \approx 3n^2$.

Problema 3.6 Fie A în $\mathbf{R}^{m\times n}$, $m > n$, o matrice de rang n a cărei factorizare QR este cunoscută ($A = QR$, unde matricele $Q \in \mathbf{R}^{m\times m}$ şi $R \in \mathbf{R}^{m\times n}$ sunt cunoscute).

a. Fie $\tilde{A} = [A\ z]$, cu $z \in \mathbf{R}^m$ un vector dat. Construiţi un algoritm eficient pentru calculul factorizării QR a lui $\tilde{A}$.

b. Fie $\tilde{A} = \begin{bmatrix} w^T \\ A \end{bmatrix}$, cu $w \in \mathbf{R}^n$ un vector dat. Arătaţi că toate coloanele lui $\tilde{A}$ sunt liniar independente şi construiţi un algoritm pentru calculul factorizării QR a lui $\tilde{A}$.

Rezolvare. a. Fie $B = Q^T\tilde{A} = [Q^TA \quad Q^Tz] = [R\ y] \in \mathbf{R}^{m\times(n+1)}$. B este deja superior triunghiulară în primele sale n coloane, i.e. în afara ultimei sale coloane $y = Q^Tz$. Fie $U_n \in \mathbf{R}^{m\times m}$ reflectorul Householder pentru care $(U_ny)_i = 0$, pentru $i > n$. Atunci, $U_nB = [R\ U_ny] = \tilde{R}$ este superior triunghiulară; $U_nQ^T\tilde{A} = \tilde{R}$ implică faptul că $\tilde{A} = QU_n\tilde{R}$ şi, prin urmare, $\tilde{Q} = QU_n$. Astfel putem calcula $\tilde{Q}$ cu ajutorul următoarei scheme

1. $y = Q^Tz$
2. Se calculează reflectorul $U_n \in \mathbf{R}^{m\times m}$ astfel încât $(U_ny)(n+1:m) = 0$
3. $\tilde{Q} = QU_n$

Detaliile sunt lăsate studentului.

b. Dacă matricea $\tilde{A}$ nu ar avea coloanele liniar independente atunci ar exista un vector nenul x astfel încât $\tilde{A}x = \begin{bmatrix} w^T \\ A \end{bmatrix} x = \begin{bmatrix} w^Tx \\ Ax \end{bmatrix} = 0$, i.e. $Ax = 0$, i.e. rangul lui A este mai mic decât n. Contradicţie. Să definim acum matricea ortogonală $Z = \begin{bmatrix} 1 & 0 \\ 0 & Q^T \end{bmatrix}$. Atunci, matricea

$$H = Z\tilde{A} = \begin{bmatrix} 1 & 0 \\ 0 & Q^T \end{bmatrix} \begin{bmatrix} w^T \\ QR \end{bmatrix} = \begin{bmatrix} w^T \\ R \end{bmatrix}$$

va fi superior Hessenberg. Factorizarea QR $H = \hat{Q}\hat{R}$ a acestei matrice poate fi calculată ca în problema 3.5. Prin urmare, $\tilde{A} = Z^TH = \begin{bmatrix} 1 & 0 \\ 0 & Q \end{bmatrix} \hat{Q}\hat{R} = \tilde{Q}\tilde{R}$, unde $\tilde{Q} = \begin{bmatrix} 1 & 0 \\ 0 & Q \end{bmatrix} \hat{Q}$ şi $\tilde{R} = \hat{R}$ definesc factorizarea QR a lui $\tilde{A}$. Detaliile şi scrierea algoritmului formal sunt lăsate în sarcina studentului.

Problema 3.7 Fie $A \in \mathbb{R}^{m\times n}$ o matrice dată. Scrieţi un algoritm eficient pentru calculul unei matrice ortogonale V astfel încât $VA = L$ să fie inferior triunghiulară (trapezoidală)

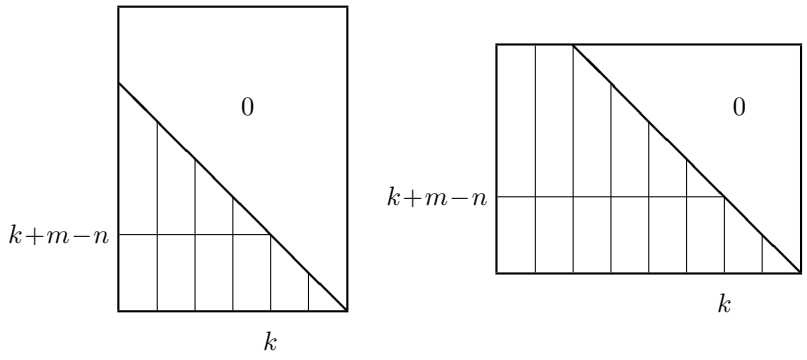

Figure 3.3: Matricea inferior triunghiulară (trapezoidală) $m \times n$, relativ la diagonala sud-est, cu $m > n$ (a) and $m < n$ (b).

relativ la diagonala care se sfârşeşte în colţul din dreapta-jos al matricei A (i.e. elementele acestei diagonale sunt $A(i,j)$ cu $i - j = m - n$) care va fi numită în continuare *diagonala sud-est*, vezi fig.3.3. În acest context calculaţi aşa numita factorizare QL a lui A, i.e. o matrice ortogonală Q şi o matrice inferior triunghiulară (trapezoidală) (relativ la diagonala sud-est) L astfel încât $A = Q * L$.

Rezolvare. Pentru a rezolva această problemă avem nevoie de aşa numiţii reflectori modificaţi care pot să introducă zerouri pe primele poziţii ale unui vector dat. Un reflector modificat de ordin m şi indice k este definit prin

$$V_k = I_m - v_k v_k^T/\beta_k \in \mathbb{R}^{m\times m}, \quad v_k = \begin{bmatrix} v_{1k} \\ \vdots \\ v_{kk} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^m, \quad \beta_k = \|v_k\|^2/2.$$

Se observă uşor că dacă $x \in \mathbb{R}^m$ este un vector dat cu $\sigma = \|x(1:k)\| \neq 0$ şi

$$v_{kk} = x_k \pm \sigma, \quad v_{ik} = x_i, \;\; i = k-1:-1:1,$$

atunci $\beta_k = v_{kk}\sigma$ şi

$$y = V_k x \quad \text{are} \quad y(1:k-1) = 0, \;\; y_k = -(\pm\sigma), \;\; y(k+1:m) = x(k+1:m).$$

Semnul lui σ este ales astfel încât în calculul lui v_{kk} operaţia să fie o adunare nu o scădere.

Acum putem introduce zerouri în A deasupra diagonalei sud-est. Pentru a păstra zerourile introduse în coloanele anterioare trebuie să desfăşurăm procesul de calcul în ordinea inversă a coloanelor, i.e. calculând

$$A \leftarrow L = V_s V_{s+1} \cdots V_m A$$

unde $s = \max(1, n-m+2) + m - n$, precum în următoarea schemă de calcul:

1. $p = \max(1, n - m + 2)$
2. **pentru** $k = n : -1 : p$
   1. Se calculează reflectorul modificat $V_{k+m-n} \in \mathbf{R}^{m\times m}$ astfel încât $(V_{k+m-n}A)(1 : k + m - n - 1) = 0$
   2. $A \leftarrow V_{k+m-n}A$.

La pasul curent k din schema de mai sus, instrucţiunea 2.2

- lasă coloanele $> k$ neschimbate,
- coloana k este transformată în zerouri deasupra diagonalei sud-est şi elementul $k + m - n, k$ al diagonalei sud-est devine $-\sigma = \pm\|A(1 : k + m - n)\|$,
- coloanele $j < k$ devin

  $$A(:.j) \leftarrow V_{k+m-n}A(:, j) = (I_m - v_{k+m-n}v_{k+m-n}^T/\beta_{k+m-n})A(:, j) = A(:, j) - v_{k+m-n}\tau,$$

  unde

  $$\tau = v_{k+m-n}^T A(:\, j)/\beta_{k+m-n} = \left(\sum_{i=1}^{k+m-n} v_{i,k+m-n}a_{ij}\right)/\beta_{k+m-n};$$

  fiind modificate doar primele $k - m + n$ elemente ale coloanei k.

Prin urmare, putem detalia schema de calcul de mai sus pentru a obţine următorul algoritm pentru triangularizarea inferioară a unei matrici date A de dimensiune $m \times n$. Matricea inferior triunghiulară L suprascrie matricea A. Pentru claritate, vectorii v vor fi stocaţi într-o matrice V şi scalarii β într-un vector b.

1. $p = \max(1, n - m + 2)$
2. **pentru** $k = n : -1 : p$
   1. $q = k + m - n$
   2. $\sigma = \text{sign}(a_{qk})\sqrt{\sum_{i=1}^{q} a_{ik}^2}$
   3. $v_{qk} = a_{qk} + \sigma$
   4. **pentru** $i = 1 : q - 1$
      1. $v_{ik} = a_{ik}$
   5. $b_q \leftarrow \beta_q = v_{qk} * \sigma$
   6. $a_{qk} = \sigma$
   7. **pentru** $i = q - 1 : -1 : 1$
      1. $a_{ik} = 0$
   8. **pentru** $j = 1 : k - 1$
      1. $\tau = \left(\sum_{i=1}^{q} v_{iq}a_{ij}\right)/\beta_q$
      2. **pentru** $i = 1 : q$
         1. $a_{ij} \leftarrow a_{ij} - \tau v_{iq}$

Efortul de calcul necesar algoritmului de mai sus este de $N_{fl} \approx 2mn^2 - 2n^3/3$ şi n extrageri de radical când $m > n$ şi de $N_{fl} \approx$ şi $m - 1$ extrageri de radical atunci când $n < m$. Dacă $m = n$ efortul de calcul este acelaşi cu cel al algoritmului $ORTHT$.

Pentru calculul factorizării QL este nevoie de acumularea transformărilor. Din $L = V_sV_{s+1}\cdots V_mA$ avem

$$A = V_mV_{m-1}\cdots V_sL = QL, \quad \text{unde} \quad Q = V_mV_{m-1}\cdots V_s.$$

Introducem sintaxa $[V, b, L] = ORTHT\_L(A)$ pentru algoritmul de mai sus şi calculăm matricea ortogonală Q cu ajutorul schemei:

1. $Q = I_m$
2. **pentru** $k = s : m$
   1. $Q \leftarrow V_k Q$

Factorizarea QL este calculată cu ajutorul următorului algoritm:

1. $[V, b, L] = ORTHT\_L(A)$
2. **pentru** $i = 1 : m$
   1. **pentru** $j = 1 : m$
      1. $q_{ij} = 0$
   2. $q_{ii} = 1$
3. $s = \max(1, n - m + 2) + m - n$
4. **for** $k = s : m$
   1. **pentru** $j = 1 : k$
      1. $\tau = \left(\sum_{i=1}^{k} v_{ik} q_{ij}\right) / \beta_k$
      2. **pentru** $i = 1 : k$
         1. $q_{ij} \leftarrow q_{ij} - \tau v_{ik}$

Calculul matricei Q necesită suplimentar $N_{fl} \approx \frac{4}{3}(m^3 - s^3)$ flopi.

Observaţie. Reducerea ortogonală de mai sus la o formă inferior triunghiulară (relativă la diagonala sud-est) şi factorizarea QL corespunzătoare pot rezolva toate problemele CMMP, fiind o alternativă de aproximativ acelaşi cost la metodele prezentate în curs (vezi problema 3.20).

Problema 3.8 Fie $A \in \mathbb{R}^{m\times n}$ o matrice monică şi $A = Q * R$ factorizarea sa QR. Demonstraţi că: coloanele matricei $Q' = Q(:, 1 : n) \in \mathbf{R}^{m\times n}$ formează o bază ortogonală pentru ImA şi coloanele matricei $Q'' = Q(:, n + 1 : m) \in \mathbf{R}^{m\times(m-n)}$ formează o bază ortogonală pentru subspaţiul KerA^T.

Rezolvare. Prin definiţie Im$A = \{y \in \mathbb{R}^m \mid \exists x \in \mathbb{R}^n \text{ a.i. } y = Ax\}$. Pentru a demonstra că coloanele lui $Q' = Q(:, 1 : n) \in \mathbf{R}^{m\times n}$ formează o bază ortogonală pentru ImA vom arăta că ImA =ImQ' demonstrând că ImA $\subset$ImQ' şi ImQ' $\subset$ImA. Intr-adevăr, factorizarea QR poate fi scrisă:

$$A = Q * R = \begin{bmatrix} Q' & Q'' \end{bmatrix} * \begin{bmatrix} R' \\ 0 \end{bmatrix} = Q' * R'$$

unde matricea superior triunghiulară $R' \in \mathbb{R}^{n\times n}$ este nesingulară datorită faptului că A este monică. Pentru toţi $y \in$ ImA avem $y = Ax = Q'R'x = Q'z$, i.e. $y \in$ ImQ'. Prin urmare ImA $\subset$ImQ'. Invers, pentru toţi $y \in$ ImQ' avem $y = Q'x = Q'R'(R')^{-1}x = A(R')^{-1}x = Az$, i.e. $y \in$ ImA. Astfel ImQ' $\subset$ImA.

Prin definiţie Ker$A^T = \{y \in \mathbb{R}^m \mid A^T y = 0\}$. Pentru a demonstra că coloanele lui $Q'' = Q(:, n + 1 : m) \in \mathbf{R}^{m\times(m-n)}$ formează o bază ortogonală pentru KerA^T vom arăta că KerA^T =ImQ' arătând că KerA^T $\subset$ImQ'' şi ImQ'' $\subset$KerA^T. Pentru toţi $y \in$ KerA^T avem $A^T y = 0$, i.e. $R^T Q^T y = \begin{bmatrix} R'^T & 0 \end{bmatrix} \begin{bmatrix} Q'^T \\ Q''^T \end{bmatrix} y = R'^T Q'^T y = 0$. Dar R' este nesingulară

şi, deci, $Q'^T y = 0$, i.e. $y \in \text{Ker}Q'^T = \text{Im}Q''$. Prin urmare $\text{Ker}A^T \subset \text{Im}Q''$. Altfel, dacă $y \in \text{Im}Q''$, atunci $y \in \text{Ker}Q'^T$, i.e. $Q'^T y = 0$. Deci, $R'^T Q'^T y = R^T Q^T y = A^T y = 0$, i.e. $y \in \text{Ker}A^T$. Astfel $\text{Im}Q'' \subset \text{Ker}A^T$. Demonstraţia este completă.

3.2.3 Rezolvarea sistemelor liniare supradeterminate

Problema 3.9 Fie un sistem cu m ecuaţii şi o necunoscută $a\xi = b$, unde $a, b \in \mathbb{R}^m$ sunt date. Rezolvaţi acest sistem folosind a) reflectori, b) rotaţii, c) pseudo-inversa. Daţi o interpretare geometrică pentru cazul în care $m = 2$.

Rezolvare. a) Este suficient un singur reflector elementar U_1 astfel încât $U_1 a = \rho e_1$. Atunci soluţia CMMP va fi $\xi = (U_1 b)(1)/\rho$. Algoritmul este:

1. $\sigma \leftarrow \text{sign}(a_1)\sqrt{\sum_{i=1}^m a_i^2}$
2. $u_1 = a_1 + \sigma$
3. $\beta = u_1 \sigma$
4. **pentru** $i = 2 : m$
   1. $u_i = a_i$
5. $\rho = -\sigma$
6. $\xi = \dfrac{b_1 - u_1 \dfrac{\sum_{i=1}^m u_i b_i}{\beta}}{\rho}$

Numărul de flopi necesar pentru algoritmul de mai sus este de $N_{fl} \approx 4m$ şi o singură extragere de radical.

b) Pentru a triangulariza matricea a avem nevoie de o secvenţă de $m-1$ rotaţii,

$$P_{1,m} \cdots P_{13} P_{12} a = \rho e_1.$$

După aceea trebuie să aplicăm aceste rotaţii vectorului b şi în final soluţia CMMP va fi $\xi = b_1/\rho$. Algoritmul este:

1. **pentru** $k = 2 : m$
   1. $\rho \leftarrow \sqrt{a_1^2 + a_k^2}$
   2. $c \leftarrow a_1/\rho$
   3. $s \leftarrow a_k/\rho$
   4. $a_1 \leftarrow \rho$
   5. $a_k \leftarrow 0$
   6. $\tau \leftarrow b_1$
   7. $b_1 \leftarrow cb_1 + sb_k$
   8. $b_k \leftarrow -s\tau + cb_k$
2. $\xi = b_1/\rho$

Acest al doilea algoritm are nevoie de $N_{fl} \approx 11m$ flopi şi $m-1$ extrageri de radical, mult mai mult decât primul.

c) În acest caz particular formula

$$\xi = (a^T a)^{-1} a^T b = \frac{a^T b}{a^T a} = \frac{\sum_{i=1}^m a_i b_i}{\sum_{i=1}^m a_i^2}$$

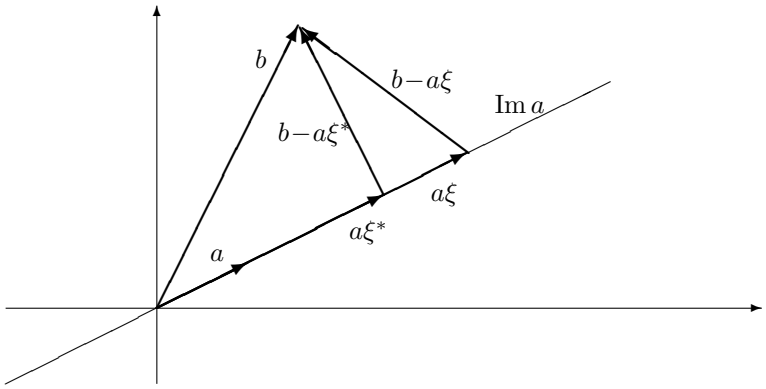

Figure 3.4: Interpretarea geometrică a problemei CMMP pentru $m = 2$ şi $n = 1$.

poate fi aplicată cu succes, având nevoie doar de $N_{fl} \approx 4m$ flopi.

Interpretarea geometrică este dată în figura 3.4 şi exprimă faptul cunoscut că soluţia CMMP ξ^A minimizează lungimea euclidiană a vectorului $b - a\xi$, i.e.

$$||b - a\xi^A|| = min_{\xi \in \mathbb{R}} ||b - a\xi||$$

.

Problema 3.10 Fie $H \in \mathbf{R}^{(n+1)\times n}$ o matrice monică superior Hessenberg dată ($h_{ij} = 0$ for $i > j + 1$) şi $b \in \mathbf{R}^{n+1}$ un vector dat. Construiţi un algoritm eficient pentru rezolvarea sistemului liniar supradeterminat $Hx = b$.

Rezolvare. Vom folosi algoritmul $ORTH\_H\_ROT$ (vezi problema rezolvată 3.5) pentru reducerea rezolvării sistemului supradeterminat $Hx = b$ la rezolvarea unui sistem determinat superior triunghiular. Detaliând transformările ortogonale ale vectorului b şi folosind $UTRIS$ pentru rezolvarea unui sistem superior triunghiular, algoritmul va fi:

1. $[c, s, R] = ORTH\_H\_ROT(H)$
2. **pentru** $k = 1 : n$
   1. $\tau \leftarrow b_k$
   2. $b_k \leftarrow c_k b_k + s_k b_{k+1}$
   3. $b_{k+1} \leftarrow -s_k \tau + c_k b_{k+1}$
3. $x = UTRIS(R(1 : n, :), b(1 : n))$

Efortul de calcul este de $N_{fl} \approx 4n^2$ flopi şi n extrageri de radical.

3.2.4 Rezolvarea sistemelor liniare subdeterminate

Problema 3.11 Fie o ecuaţie liniară cu $n > 1$ necunoscute

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = \gamma,$$

sau, intr-o formă concisă, $a^T x = \gamma$, unde $a \in \mathbb{R}^n$ şi $\gamma \in \mathbb{R}$ sunt date. Rezolvaţi această ecuaţie în sensul CMMP folosind: a) reflectori, b) rotaţii, c) pseudo-inversa. Daţi o interpretare geometrică pentru $m = 2$.

Rezolvare. a) Este suficient un singur reflector elementar U_1 astfel încât $U_1 a = \rho e_1$ (vezi problema 3.9 pentru algoritm), i.e. $a^T U_1 = \rho e_1^T$. Ecuaţia $a^T x = \gamma$ poate fi scrisă ca $a^T U_1 U_1 x = \gamma$ şi devine $\rho e_1^T y = \gamma$, unde $y = U_1 x$. Soluţia normală, i.e. soluţia de normă euclidiană minimă, a acestei ultime ecuaţii este, evident, $y = \frac{\gamma}{\rho} e_1$. Atunci soluţia de normă euclidiană minimă a ecuaţiei iniţiale este

$$x = U_1 y = \frac{\gamma}{\rho} U_1 e_1 = \frac{\gamma}{\rho}(I_n - \frac{uu^T}{\beta})e_1 = \frac{\gamma}{\rho}(e_1 - \frac{uu_1}{\beta}).$$

Definind scalarii $\tau = \frac{\gamma}{\rho}$, $\mu = \frac{u_1}{\beta}$ şi $\nu = \tau * \mu$ algoritmul va fi:

1. $\sigma \leftarrow \text{sign}(a_1)\sqrt{\sum_{i=1}^n a_i^2}$
2. $u_1 = a_1 + \sigma$
3. $\beta = u_1 \sigma$
4. **for** $i = 2 : n$
   1. $u_i = a_i$
5. $\rho = -\sigma$
6. $\mu = \frac{u_1}{\beta}$
7. $\nu = \tau * \mu$
8. $x_1 = \tau * (1 - u_1 * \mu)$
9. **pentru** $i = 2 : n$
   1. $x_i = -\nu u_i$

Numărul de flopi necesari acestui algoritm este de $N_{fl} \approx 3n$ şi de o extragere de radical.

b) Precum în problema 3.9, vom folosi secvenţa P_{12}, P_{13}, ... $P_{1,n}$ de $n-1$ rotaţii plane, astfel încât $P_{1,n} \cdots P_{13} P_{12} a = \rho e_1$ or $a^T P_{12}^T P_{13}^T \cdots P_{1,n}^T = \rho e_1^T$. Ca în cazul anterior, ecuaţia $a^T x = \gamma$ poate fi scrisă

$$a^T P_{12}^T P_{13}^T \cdots P_{1,n}^T P_{1,n} \cdots P_{13} P_{12} x = \gamma$$

şi devine $\rho e_1^T y = \gamma$, unde $y = P_{1,n} \cdots P_{13} P_{12} x$. Soluţia de normă euclidiană minimă a acestei ultime ecuaţii este, în mod evident, $y = \frac{\gamma}{\rho} e_1$. Soluţia de normă euclidiană minimă pentru prima ecuaţie va fi

$$x = P_{12}^T P_{13}^T \cdots P_{1,n}^T y = \frac{\gamma}{\rho} P_{12}^T P_{13}^T \cdots P_{1,n}^T e_1.$$

Algoritmul este:

1. **pentru** $k = 2 : n$
   1. $\rho \leftarrow \sqrt{a_1^2 + a_k^2}$
   2. $c_k \leftarrow a_1/\rho$
   3. $s_k \leftarrow a_k/\rho$

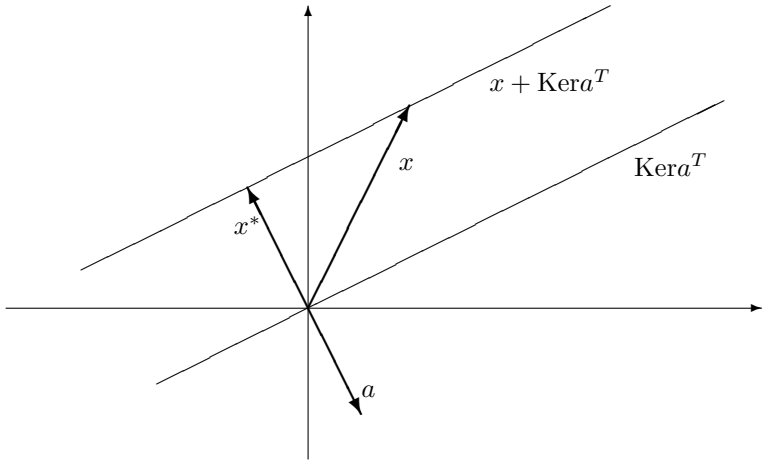

Figure 3.5: Interpretarea geometrică a soluţiei normale pentru $m = 1$ şi $n = 2$.

4. $a_1 \leftarrow \rho$
5. $a_k \leftarrow 0$

2. $x_1 = \dfrac{\gamma}{\rho}$
3. **pentru** $i = 2 : n$
   1. $x_i = 0$
4. **pentru** $k = n : -1 : 2$
   1. $\tau = x_1$
   2. $x_1 = c_k \leftarrow c_k x_1 - s_k x_k$
   3. $x_k \leftarrow s_k \tau + c_k x_k$

Acest al doilea algoritm necesită $N_{fl} \approx 11n$ flopi şi $n - 1$ extrageri de radical, mult mai mult decât primul. Prin urmare nu este recomandat.

c) În acest caz particular, formula

$$x = a^{\#}\gamma = a(a^T a)^{-1}\gamma, \quad \text{i.e.} \quad x_i = \frac{a_i \gamma}{a^T a}, \quad i = 1 : n$$

poate fi aplicată cu succes prin algoritmul:

1. $\tau = \dfrac{\gamma}{\sum_{i=1}^{n} a_i^2}$
2. **pentru** $i = 1 : n$
   1. $x_i = a_i * \tau$

având nevoie de numai $N_{fl} \approx 3n$ flopi.

Interpretarea geometrică a soluţiei normale pentru cazul în care $n =$ este dată în fig. 3.5, unde $x \in \mathbb{R}^2$ este o soluţie arbitrară, $x + \mathrm{Ker}\, a^T$ este setul complet de soluţii iar x^* este soluţia normală.

Problema CMMP pentru rezolvarea sistemelor liniare supradeterminate $Ax = b$ constă în minimizarea funcţiei $g(x) = \|Ax - b\|^2$ iar problema CMMP pentru rezolvarea sistemelor

liniare nedeterminate $Ax = b$ constă în minimizarea funcţiei $h(x) = \|x\|^2$ cu restricţiile $Ax = b$. Să combinăm cele două funcţii în problema următoare.

Problema 3.12 Fie $A \in \mathbb{R}^{m\times n}$ şi $b \in \mathbb{R}^m$ date şi să considerăm funcţia pătratică

$$f : \mathbb{R}^n \to \mathbb{R}_+, \qquad f(x) = \|Ax - b\|^2 + \|x\|^2.$$

Arătaţi că f are un minim unic în x^* şi construiţi un algoritm pentru calculul său şi al lui $\rho = f(x^*)$.

Rezolvare. Funcţia

$$f(x) = \|b - Ax\|^2 + \|x\|^2 = \sum_{i=1}^{m} \left[\left(\sum_{j=1}^{n} a_{ij}x_j - b_i \right)^2 + x_i^2 \right]$$

este continuă şi diferenţiabilă. Punctele de extrem trebuie să satisfacă relaţia

$$\frac{\partial f(x)}{\partial x_k} = 0, \quad k = 1 : n.$$

Dar

$$\frac{\partial f(x)}{\partial x_k} = \sum_{i=1}^{n} 2 \left[\frac{\partial f(x)}{\partial x_k} \left(\sum_{j=1}^{n} a_{ij}x_j - b_i \right) \right] \left(\sum_{j=1}^{n} a_{ij}x_j - b_i \right) + 2x_k =$$

$$= \sum_{i=1}^{n} 2a_{ik} \left(\sum_{j=1}^{n} a_{ij}x_j - b_i \right) + 2x_k \quad k = 1 : n,$$

i.e. punctele de extrem x^* trebuie să satisfacă sistemul liniar

$$(A^TA + I)x^* = A^Tb.$$

Matricea $A^TA + I$ fiind pozitiv definită (de demonstrat) este nesingulară, astfel posibilul punct de extrem este unic. Prin calculul celei de-a doua diferenţiale se poate observa uşor că soluţia $x^* = (A^TA + I_n)^{-1}A^Tb$ a sistemului liniar de mai sus este într-adevăr unicul punct de minim al lui f. Valoarea minimă a lui f este

$$\rho = f(x^*) = \|Ax^* - b\|^2 + \|x^*\|^2 = \|A(A^TA + I_n)^{-1}A^Tb - b\|^2 + \|(A^TA + I_n)^{-1}A^Tb\|^2$$

Pentru calculul lui x^* o metodă directă este:

1. $B = A^TA + I$
2. $d = A^Tb$
3. Se calculează factorizarea Cholesky $B = LL^T$
4. $y = LTRIS(L, d)$
5. $x^* = UTRIS(L^T, y)$.

O metodă mai bună constă în a observa că

$$B = A^T A + I = \begin{bmatrix} A^T & I_n \end{bmatrix} \begin{bmatrix} A \\ I_n \end{bmatrix} = C^T C, \qquad C = \begin{bmatrix} A \\ I_n \end{bmatrix} \in \mathbb{R}^{(m+n)\times n},$$

$$A^T b = \begin{bmatrix} A^T & I_n \end{bmatrix} \begin{bmatrix} b \\ 0 \end{bmatrix} = C^T d, \qquad d = \begin{bmatrix} b \\ 0 \end{bmatrix} \in \mathbb{R}^{m+n},$$

unde matricea C este în mod evident una monică. Deci x^* este unica (pseudo-)soluţie CMMP a sistemului liniar supra determinat $Cx = d$ care poate fi rezolvat prin metode standard (desigur, exploatarea formei specifice a matricei C este insistent recomandată). Detaliile sunt lăsate în sarcina studenţilor.

3.3 Probleme propuse

Problema 3.13 Fie un vector dat $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ în planul $\mathbb{R}^2$.

a. Pentru un vector dat $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \in \mathbb{R}^2$ calculaţi reflectorul $U = I_2 - 2\dfrac{uu^T}{\|u\|}$ definit de către ele şi vectorul $y = Ux$. Apoi, calculaţi un vector u astfel încât $y_2 = 0$ şi desenaţi vectorii x, u şi y. Daţi o interpretare geometrică celor două desene.

b. Pentru un număr real $c < 1$ dat fie $s = \sqrt{1 - c^2}$ şi considerăm rotaţia $P = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$ şi vectorul $y = Px$. Calculaţi c astfel încât $y_2 = 0$. Pentru vectorul $x = \begin{bmatrix} 5 \\ 12 \end{bmatrix}$ cu $c = 1/2$ desenaţi vectorii x şi y. Daţi o interpretare geometrică celor două desene.

Problema 3.14 Rezolvaţi în sens CMMP următoarele două sisteme liniare supradeterminate de 3 ecuaţii şi o necunoscută

$$\begin{cases} 3\xi = 7 \\ 4\xi = 7 \\ 12\xi = 10 \end{cases}, \qquad \begin{cases} 6\xi = 14 \\ 4\xi = 7 \\ 12\xi = 10 \end{cases}.$$

Explicaţi de ce cele două (pseudo-)soluţii sunt diferite.

Problema 3.15 Fie $A \in \mathbb{R}^{m\times m}$ o matrice dată şi $u \in \mathbb{R}^m$ un vector dat. Considerând reflectorul $U = I_m - 2\dfrac{uu^T}{\|u\|^2}$ construiţi un algoritm eficient pentru calculul matricei $A \leftarrow B = UAU$.

Problema 3.16 **a.** Adaptaţi algoritmul original de triangularizare $ORTHT$ pentru matricea $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$, unde $A_1 \in \mathbf{R}^{n\times n}$ este superior triunghiulară şi $A_2 \in \mathbf{R}^{(m-n)\times n}$. **b.** Aceeaşi problemă în cazul în care şi A_2 este superior triunghiulară.

Problema 3.17 Fie $A \in \mathbb{R}^{m\times n}$ o matrice cu $m < n$. Calculaţi o triangularizare ortogonală la dreapta, i.e. o matrice ortogonală $V \in \mathbb{R}^{n\times n}$ astfel încât matricea $L = AV$ să fie inferior triunghiulară. Folosind acest rezultat, calculaţi factorizarea LQ a lui A, i.e. calculaţi o matrice inferior triunghiulară L şi o matrice ortogonală Q astfel încât $A = LQ$.

Problema 3.18 Fie $a, b \in \mathbf{R}^n$ doi vectori necoliniari. Construiţi un algoritm pentru calculul proiecţiei lui b pe direcţia lui a.

Problema 3.19 Fie dată matricea $A \in \mathbf{R}^{m\times n}$ de rang maximal. **a.** Construiţi algoritmi eficienţi pentru calculul bazei ortogonale pentru subspaţiile liniare $\mathrm{Im}A \subset \mathbb{R}^m$, $\mathrm{Ker}A^T \subset \mathbb{R}^m$, $\mathrm{Im}A^T \subset \mathbb{R}^n$ şi $\mathrm{Ker}A \subset \mathbb{R}^n$.

b. Dacă, în plus, sunt daţi şi vectorii $b \in \mathbb{R}^m$ şi $c \in \mathbb{R}^n$, construiţi algoritmi eficienţi pentru calculul proiecţiilor ortogonale ale lui b pe $\mathrm{Im}A$, $\mathrm{Ker}A^T$ şi pentru proiecţiile lui c pe $\mathrm{Im}A^T$, $\mathrm{Ker}A$.

Problema 3.20 Fie $A \in \mathbf{R}^{m\times n}$ o matrice dată şi $b \in \mathbb{R}^m$ un vector dat. Construiţi algoritmi eficienţi pentru rezolvarea problemei CMMP folosind factorizarea QL (vezi problema rezolvată 3.7).

3.4 Bibliografie

1. B. Jora, B. Dumitrescu, C. Oară, Numerical Methods, UPB, Bucharest, 1995.
2. G.W. Stewart, Introduction to Matrix Computations, Academic Press, 1973.
3. G. Golub, Ch. Van Loan, Matrix Computations, 3-rd edition, John Hopkins University Press, 1998.
4. G.W. Stewart, Matrix Algorithms, vol.1: Basic Decompositions, SIAM, 1999.
5. G.W. Stewart, Matrix Algorithms, vol.2: Eigensystems, SIAM, 2001.
6. B. Dumitrescu, C. Popeea, B. Jora, Metode de calcul numeric matriceal. Algoritmi fundamentali, ALL, Bucureşti, 1998.

3.5 Programe MATLAB

În această secţiune sunt date programele MATLAB pentru implementarea algoritmilor prezentaţi în acest seminar. Programele au fost testate şi pot fi obţinute de la autorul lor menţionat în comentariile ataşate.

Problema 3.4

```
1:  function [Q]=p34(x,y)
2:  %-------------------------------------------
3:  %Acesta este un algoritm pentru calculul matricei Q
4:  %din problema 3.4.
5:  %Primeste ca date de intrare vectorii x si y cu norme euclidiene
6:  %egale.
7:  %Qx=y
8:  %Apelul:
9:  % [Q]=p34(x,y)
10: %Stamatescu Grigore Mai 2006
11: %-------------------------------------------
12: if norm(x)~=norm(y),
13:     error('Cei doi vectori au norme euclidiene distincte!');
14: end
15:
16: ro=0; gamma=0; beta=0; sigma=0; tau=0; s=0;
17: n=length(x);
18: for i=1:n
19:     ro=ro+x(i)^2;
20: end
21: ro=sqrt(ro);
22: sigma=sign(x(1))*ro;
23: u(1)=x(1)+sigma;
24: beta=u(1)*sigma;
25: for i=2:n
26:     u(i)=x(i);
27: end
28: sigma=sign(y(1))*ro;
29: v(1)=y(1)+sigma;
30: gamma=v(1)*sigma;
31: for i=2:n
32:     v(i)=y(i);
33: end
34: for i=1:n
35:     s=s+v(i)*u(i);
36: end
37: tau=s/(beta*gamma);
38: for i=1:n
39:     for j=1:n
```

```
40:             Q(i,j)=-u(i)*u(j)/beta - v(i)*v(j)/gamma + tau*v(i)*u(j);
41:         end
42:         Q(i,i)=1+Q(i,i);
43:     end
```

```
 1:     function [U,beta,R]= ORTHT_H(H)
 2:     %-----------------------
 3:     %Algoritmul calculeaza matricea R a factorizarii QR a matricei
 4:     %superior Hessenberg H folosind transformarile Householder.
 5:     %Apelul:
 6:     % [U,beta,R]=ORTHT_H(H)
 7:     %Stamatescu Grigore, Mai 2006
 8:     %-----------------------
 9:     [m,n]=size(H);
10:     if m~=n+1,
11:         error('Matricea trebuie sa fie de dimensiune (n+1)x n!');
12:     end
13:
14:     for i=1:m
15:         for j=1:n
16:             if i>j+1,
17:                 if H(i,j)~=0,
18:                     error('Matricea nu este superior Hessenberg');
19:                 end
20:             end
21:         end
22:     end
23:     for k=1:n
24:         sigma= sign(H(k,k))*sqrt( H(k,k)^H(k,k) + H(k+1,k)*H(k+1,k) );
25:         U(k,k)=H(k,k) + sigma;
26:         U(k+1,k)=H(k+1,k);
27:         beta(k)=U(k,k)*sigma;
28:         R(k,k)=-sigma;
29:         H(k,k)=R(k,k);
30:         H(k+1,k)=0;
31:         R(k+1,k)=0;
32:         for j=k+1:n
33:             tau=(U(k,k)*H(k,j) + U(k+1,k)*H(k+1,j))/beta(k);
34:             H(k,j)=H(k,j)-tau*U(k,k);
35:             R(k,j)=H(k,j);
36:             H(k+1,j)=H(k+1,j)-tau*U(k+1,k);
37:             R(k+1,j)=H(k+1,j);
38:         end
39:     end
```

```
1:  function [Q]=QR1(H)
2:  %-------------------------
3:  %Calculam matricea Q a factorizarii QR pentru matricea
4:  %superioara Hessenberg H.
5:  %Se folosesc transformarile Householder.
6:  %Apel:
7:  % [Q]=QR1(H)
8:  %Stamatescu Grigore, Mai 2006
9:  %-------------------------
10:
11: tau=0;
12: [U,beta,R]=ORTHT_H(H);
13: [m,n]=size(H);
14: for i=1:n+1
15:     for j=1:n+1
16:         Q(i,j)=0;
17:     end
18:     Q(i,i)=1;
19: end
20: for k=1:n
21:     for i=1:k+1
22:         tau = (Q(i,k)*U(k,k) + Q(i,k+1)*U(k+1,k))/beta(k);
23:         Q(i,k)=Q(i,k)-tau * U(k,k);
24:         Q(i,k+1)=Q(i,k+1)-tau * U(k+1,k);
25:     end
26: end
```

```
1:  function [c,s,R]= ORTHT_H_ROT(H)
2:  %-------------------------------
3:  %Alogritmul calculeaza matricea superior triunghiulara R
4:  %a factorizarii QR a mtricei H cu ajutorul rotatiilor
5:  %Givens.
6:  %Apelul:
7:  % [c,s,R]= ORTHT_H_ROT(H)
8:  %Stamatescu Grigore, Mai 2006
9:  %-------------------------------
10: [m,n]=size(H);
11: if m~=n+1,
12:     error('Matricea trebuie sa fie de dimensiune (n+1)x n!');
13: end
14:
15: for i=1:m
16:     for j=1:n
17:         if i>j+1,
18:             if H(i,j)~=0,
```

```
19:                         error('Matricea nu este superior Hessenberg');
20:                     end
21:                 end
22:             end
23:     end
24:
25:     for k=1:n
26:         ro=sqrt(H(k,k)*H(k,k)+H(k+1,k)*H(k+1,k));
27:         c(k)=H(k,k)/ro;
28:         s(k)=H(k+1,k)/ro;
29:         R(k,k)=ro;
30:         H(k,k)=R(k,k);
31:         H(k+1,k)=0;
32:         R(k+1,k)=0;
33:         for j=k+1:n
34:             tau=c(k)*H(k,j)+s(k)*H(k+1,j);
35:             H(k+1,j)=-s(k)*H(k,j)+c(k)*H(k+1,j);
36:             R(k+1,j)=H(k+1,j);
37:             H(k,j)=tau;
38:             R(k,j)=tau;
39:         end
40:     end
```

```
1:  function [Q]= QR2(H)
2:  %-------------------
3:  %Alogritmul calculeaza factorizarea QR a matricei
4:  %H superior Hessenberg folosind algoritmul de
5:  %triangularizare cu rotatii.
6:  %Apelul:
7:  % Q= QR2(H)
8:  %Stamatescu Grigore, Mai 2006
9:  %-------------------
10: [m,n]=size(H);
11: if m~=n+1,
12:     error('Matricea trebuie sa fie de dimensiune (n+1)x n!');
13: end
14:
15: for i=1:m
16:     for j=1:n
17:         if i>j+1,
18:             if H(i,j)~=0,
19:                 error('Matricea nu este superior Hessenberg');
20:             end
21:         end
22:     end
23: end
```

```
[c,s,R]=ORTHT_H_ROT(H);
for i=1:n+1
    for j=1:n+1
        Q(i,j)=0;
    end
    Q(i,i)=1;
end
for k=1:n
    for i=1:k+1
        tau=Q(i,k)*c(k)+Q(i,k+1)*s(k);
        Q(i,k+1)=-Q(i,k)*s(k)+Q(i,k+1)*c(k);
        Q(i,k)=tau;
    end
end
```

```
function [V,b,A]= ORTHT_L(A)
%-----------------------
%Algoritmul realizeaza triangularizarea inferioara a unei
%matrice. Matricea inferior triunghiulara va suprascrie
%matricea A.
%Apelul:
% [V,b,A]=ORTHT_L(A)
%Stamatescu Grigore, Mai 2006
%-----------------------
[m,n]=size(A);
p=1;
if n-m+2>1,
    p=n-m+2;
end
for k=n:-1:p
    q=k+m-n;
    suma=0;
    for i=1:q
        suma=suma+A(i,k)*A(i,k);
    end
    sigma=sign(A(q,k))*sqrt(suma);
    V(q,k)=A(q,k)+sigma;
    for i=1:q-1
        V(i,k)=A(i,k);
    end
    beta(q)=V(q,k)*sigma;
    b(q)=beta(q);
    A(q,k)=sigma;
    for i=q-1:-1:1
        A(i,k)=0;
    end
```

```
32:         for j=1:k-1
33:             s=0;
34:             for i=1:q
35:                 s=s+V(i,j)*A(i,j);
36:             end
37:             tau=s/beta(q);
38:             for i=1:q
39:                 A(i,j)=A(i,j)-tau*V(i,j);
40:             end
41:         end
42:     end
```

```
1:  function [x]= p310(H,b)
2:  %-------------------------
3:  %Algoritmul calculeaza solutia sistemului liniar supradeterminat
4:  %Hx=b.
5:  %(Rezolvarea problemei 3.10)
6:  %Apelul:
7:  % x= p310(H,b)
8:  %Stamatescu Grigore, Mai 2006
9:  %-------------------------
10: [m,n]=size(H);
11: if m~=n+1,
12:     error('Matricea trebuie sa fie de dimensiune (n+1)x n!');
13: end
14: t=length(b);
15: if m~=n+1,
16:     error('Vectorul b trebuie sa fie de lungime n+1!');
17: end
18: for i=1:m
19:     for j=1:n
20:         if i>j+1,
21:             if H(i,j)~=0,
22:                 error('Matricea nu este superior Hessenberg');
23:             end
24:         end
25:     end
26: end
27: [c,s,R]=ORTHT_H_ROT(H);
28: for k=1:n
29:     tau=b(k);
30:     b(k)=c(k)*b(k)+s(k)*b(k+1);
31:     b(k+1)=-s(k)*tau+c(k)*b(k+1);
32: end
33: x=UTRIS(R(1:n,:),b(1:n));
```

Problema 3.9 a

```
1:  function [xi]=p39a(a,b)
2:  %-----------------------
3:  %Acest algoritm calculeaza folosind reflectori
4:  %solutia xi a sistemului a*xi=b de m ecuatii si
5:  %o necunoscuta. (Problema 3.9. subpunctul a)
6:  %a,b sunt dati.
7:  %Apelul:
8:  % [xi]=p39a(a,b)
9:  %Stamatescu Grigore, Mai 2006
10: %-----------------------
11: m=length(a); n=length(b);
12:
13: %analizam daca vectorii au aceeasi dimensiune
14: if n~=m
15:     error('Vectorii trebuie sa aiba aceeasi dimensiune!');
16: end
17:
18: sigma=0; beta=0; ro=0; s=0;
19: for i=1:m
20:     s=s+a(i)^2;
21: end
22: sigma=sign(a(1))*sqrt(s);
23: u(1)=a(1)+sigma;
24: beta=u(1)*sigma;
25: for i=2:m
26:     u(i)=a(i);
27: end
28: ro=-sigma;
29: s=0;
30: for i=1:n
31:     s=s+u(i)*b(i);
32: end
33: xi=[b(1)-u(1)*s/beta]/ro;
```

Problema 3.9 b

```
1:  function [xi]=p39b(a,b)
2:  %-----------------------
3:  %Acest algoritm calculeaza folosind rotatii
4:  %solutia xi a sistemului a*xi=b de m ecuatii si
5:  %o necunoscuta. (Problema 3.9. subpunctul b)
6:  %a,b sunt dati.
7:  %Apelul:
8:  % [xi]=p39b(a,b)
9:  %Stamatescu Grigore, Mai 2006
```

```
%-----------------------
m=length(a); n=length(b);

%analizam daca vectorii au aceeasi dimensiune
if n~=m
    error('Vectorii trebuie sa aiba aceeasi dimensiune!');
end

ro=0;tau=0;
for k=2:m
    ro=sqrt(a(1)^2 + a(k)^2);
    c=a(1)/ro;
    s=a(k)/ro;
    a(1)=ro;
    a(k)=0;
    tau=b(1);
    b(1)=c*b(1)+s*b(k);
    b(k)=-s*tau+c*b(k);
end
xi=b(1)/ro;
```

Problema 3.11 a

```
function [x]=p311a(a,y)
%-----------------------
%Acest algoritm calculeaza folosind reflectori
%solutia x a ecuatiei liniare cu n>1 necunoscute
%(Problema 3.11. subpunctul a)
%Primeste ca date de intrare vectorul a si scalarul y.
%Apelul:
% [x]=p311a(a,y)
%Stamatescu Grigore, Mai 2006
%-----------------------
n=length(a);

suma=0;
for i=1:n
    suma= suma+a(i)^2;
end
sigma=sign(a(1))*sqrt(suma);
u(1)=a(1)+sigma;
beta=u(1)*sigma;
for i=2:n
    u(i)=a(i);
end
ro=-sigma;
tau=y/ro;
```

```
25:  miu=u(1)/beta;
26:  niu=tau*miu;
27:  x(1)=tau*(1-u(1)*miu);
28:  for i=2:n
29:       x(i)=-niu*u(i);
30:  end
```

Problema 3.11 b

```
1:   function [x]=p311b(a,y)
2:   %-------------------------
3:   %Acest algoritm calculeaza folosind rotatii
4:   %solutia x a ecuatiei liniare cu n>1 necunoscute
5:   %(Problema 3.11. subpunctul b)
6:   %Primeste ca date de intrare vectorul a si scalarul y.
7:   %Apelul:
8:   % [x]=p311b(a,y)
9:   %Stamatescu Grigore, Mai 2006
10:  %-------------------------
11:  n=length(a);
12:  ro=0;
13:  for k=2:n
14:       ro=sqrt(a(1)^2 + a(k)^2);
15:       c(k)=a(1)/ro;
16:       s(k)=a(k)/ro;
17:       a(1)=ro;
18:       a(k)=0;
19:  end
20:  x(1)=y/ro;
21:  for i=2:n
22:       x(i)=0;
23:  end
24:  for k=n:-1:2
25:       tau=x(1);
26:       c(k)=c(k)*x(1)-s(k)*x(k);
27:       x(1)=c(k);
28:       x(k)=s(k)*tau+c(k)*x(k);
29:  end
```

Problema 3.11 c

```
1:   function [x]=p311c(a,y)
2:   %-------------------------
3:   %Acest algoritm calculeaza folosind pseudo-inversa
4:   %solutia x a ecuatiei liniare cu n>1 necunoscute
5:   %(Problema 3.11. subpunctul c)
```

```
6:  %Primeste ca date de intrare vectorul a si scalarul y.
7:  %Apelul:
8:  % [x]=p311c(a,y)
9:  %Stamatescu Grigore, Mai 2006
10: %————————————————
11: n=length(a);
12: suma=0;
13: for i=1:n
14:       suma=suma+a(i)^2;
15: end
16: tau=y/suma;
17: for i=1:n
18:       x(i)=a(i)*tau;
19: end
```