

Introducere în calcul științific

Suport de curs și laborator

pentru studenții anului al II-lea

Ionel-Dumitrel Ghiba

Facultatea de Matematică, Universitatea “Alexandru Ioan Cuza” din Iași

dumitrel.ghiba@uaic.ro



ALEXANDRU IOAN CUZA
UNIVERSITY of IAȘI

Prefață

Calculul științific se află la intersecția dintre matematică, algoritmi și calculul efectiv pe calculator. El pornește de la o întrebare simplă: *cum transformăm un model matematic într-un rezultat numeric în care putem avea încredere?* Răspunsul nu constă doar în aplicarea unei formule. Trebuie să alegem o reprezentare adecvată a problemei, un algoritm suficient de eficient, să înțelegem erorile introduse de aritmetica în virgulă mobilă și să putem decide dacă rezultatul calculat este relevant pentru problema inițială.

Acest curs este conceput ca o primă introducere în aceste idei. Algebra liniară numerică apare în mare parte a cursului, deoarece numeroase probleme de calcul științific se reduc, exact sau după discretizare, la operații cu vectori și matrice: rezolvarea sistemelor liniare, aproximarea datelor, problema celor mai mici pătrate, calculul valorilor și vectorilor proprii, optimizarea unor funcții pătratice, aproximarea de rang mic prin descompunerea în valori singulare sau reducerea dimensiunii datelor prin analiza componentelor principale.

Scopul cursului nu este memorarea unei colecții de algoritmi. Pentru fiecare metodă ne interesează patru întrebări: *de unde provine metoda?*, *de ce funcționează?*, *cât costă?* și *cât de mult putem avea încredere în rezultatul obținut?* Aceste întrebări vor însoți metodele prezentate pe parcursul cursului.

De la formulă la algoritm

În matematică, două formule echivalente descriu același obiect. Pe calculator, două proceduri algebrice echivalente pot avea costuri și comportamente numerice foarte diferite. De exemplu, pentru sistemul $Ax = b$, formula $x = A^{-1}b$ este corectă matematic, dar calcularea explicită a inversei este în general o alegere numerică mai slabă decât rezolvarea sistemului printr-o factorizare adecvată. În calculul științific contează, așadar, nu numai formula, ci și modul în care ea este transformată într-un algoritm.

Manualul este destinat studenților anului al II-lea și presupune cunoștințe de bază de algebră liniară și analiză. Expunerea combină demonstrația matematică cu interpretarea numerică, exemple complete calculate, exerciții, sarcini de programare și mini-proiecte.

Cum se citește o secțiune

La prima lectură urmăriți ideea metodei și rolul fiecărei ipoteze. La a doua lectură refaceți demonstrațiile și calculele. În final, implementați algoritmul și testați-l pe exemple pentru care cunoașteți răspunsul. O metodă numerică este înțeleasă cu adevărat atunci când puteți explica de ce funcționează, când poate eșua, ce eroare produce și cum verificați rezultatul.

Cum lucrăm la seminar și laborator

În acest manual, calculul manual și programarea nu sunt activități separate. O implementare este considerată înțeleasă numai dacă studentul poate explica formulele din care provine, ordinea operațiilor, ipotezele în care algoritmul funcționează și un criteriu prin care rezultatul este verificat.

Pentru fiecare algoritm nou se recomandă aceeași succesiune. Mai întâi se execută cu pixul pe hârtie unul sau două cazuri mici, fără a sări etape. Apoi se scrie pseudocodul și se identifică dependențele dintre pași. Numai după aceea se trece la MATLAB. Funcțiile MATLAB de bibliotecă sunt folosite în primul rând ca referință pentru verificare; scopul laboratorului este construirea algoritmului, nu înlocuirea lui printr-o singură comandă.

Ce înseamnă “a explica un algoritm”?

Studentul trebuie să poată răspunde la întrebări precum: de ce această buclă începe de la indicele $k + 1$? de ce este necesar pivotul nenul? ce mărime scade de la o iterație la alta? de ce folosim QR și nu ecuațiile normale? ce se întâmplă dacă o valoare singulară este foarte mică? Un cod care produce un număr, dar ale cărui etape nu pot fi justificate, nu este încă o soluție completă.

Proiectele au nivel mediu și urmăresc integrarea noțiunilor. Nu se cere construirea unui sistem software complex. Se cere însă ca modelul, algoritmul, implementarea MATLAB, verificarea și interpretarea rezultatului să formeze un argument coerent.

Bibliografie de bază și materiale didactice consultate

Expunerea folosește rezultate standard din calcul științific și algebră liniară numerică și a fost calibrată didactic și prin consultarea unor suporturi moderne de curs. Principalele repere sunt:

- Lucian Maticiuc, *Introducere în MATLAB*, Iași, 2019, material disponibil online la https://www.math.uaic.ro/~maticiuc/didactic/MATLAB_Course.pdf.
- A. Quarteroni, R. Sacco, F. Saleri, *Numerical Mathematics*, Springer.
- W. Gander, M. J. Gander, F. Kwok, *Scientific Computing – An Introduction Using Maple and MATLAB*, Springer.
- A. Beck, *Introduction to Nonlinear Optimization – Theory, Algorithms, and Applications with MATLAB*, SIAM.
- L. Page, S. Brin, R. Motwani, T. Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*, Stanford Digital Library Technologies Project, 1998/1999.
- Y. Koren, R. Bell, C. Volinsky, “Matrix Factorization Techniques for Recommender Systems”, *Computer*, 42(8), 30–37, 2009.
- J. Calder, *Math 5467: Introduction to the Mathematics of Image and Data Analysis*, University of Minnesota, School of Mathematics, 2022.
- J. Gallier, J. Quaintance, *Linear Algebra for Computer Vision, Robotics, and Machine Learning*, University of Pennsylvania, 2025.
- O. Çeliker, *MIT 6.057: Introduction to MATLAB*, IAP 2019.

Organizarea și evaluarea cursului

Evaluarea la această disciplină este pe parcurs și este alcătuită din 2 componente de evaluare continuă. Evaluarea continuă constă în

- 70% verificarea înțelegerii rezolvărilor unor probleme la laborator, asemănătoare cu cele din fișele propuse săptămânal. Această verificare se va face cerând la laborator ca studenții să rezolve individual în 20 minute o sarcină de laborator. Aceste activități au caracter continuu și vor fi notați, desigur, doar studenții care le vor îndeplini. Nu se repetă, fiind considerate activități curente de laborator. **Nu se repetă pe parcursul semestrului sau în sesiunea de restanțe!**
- 30% verificarea cunoștințelor teoretice printr-un test scurt de 20 minute în ultima săptămână, la începutul ultimului curs. **Se poate repeta în sesiunea de restanțe.**

Ce se evaluează

- Accentul nu este pus numai pe obținerea unui rezultat numeric. Studentul trebuie să poată justifica alegerea metodei, să explice ipotezele teoretice, să estimeze costul și eroarea și să interpreteze rezultatul unui experiment numeric.
- Nu se punctează prezența, însă absentarea de la orele de laborator vor conduce la imposibilitatea îndeplinirii sarcinilor de lucru, neîndeplinirea lor notându-se cu nota 1/10.
- Prezența la curs nu este obligatorie, însă multe explicații teoretice și de îndeplinire a sarcinilor de laborator vor fi date doar la curs și vor fi foarte utile, evaluarea insistând pe aceste aspecte comunicate doar în sala de curs.

Notatii

Unele litere sunt folosite cu sensuri diferite în capitole diferite. Pentru a evita confuziile, convențiile principale sunt reunite mai jos.

Simbol	Sens	Capitol
P	matrice de permutare	metode directe / pivotare
P	matrice de splitting sau preconditionare	metode iterative
P_L	proiector ortogonal pe subspațiul L	PCA
P	matrice stocastică de tranziție	PageRank
U	factor superior triunghiular în LU	metode directe
U	matricea vectorilor singulari stânga	SVD
$r = b - Ax$	reziduul pentru sisteme liniare	convenție folosită în manual

Sensul unui simbol este precizat din nou acolo unde poate apărea o confuzie.

Cuprins

Prefață	i
Cum lucrăm la seminar și laborator	ii
Bibliografie de bază și materiale didactice consultate	iii
Organizarea și evaluarea cursului	iv
Notății	v
1 Introducere în calcul științific	1
1.1 Ce este calculul științific?	1
1.2 De ce apar vectorii și matricile în aplicații?	1
1.3 Problema model: sistemul liniar	4
1.4 Erori, stabilitate și condiționare	4
1.5 Cum este organizat cursul?	4
Test de capitol	7
2 Recapitulare: matrice, valori proprii și pozitiva definire	8
Extensie didactică: interpretarea spectrală a matricelor	11
Fișă de seminar: geometria matricelor simetrice	13
Test de capitol	18
3 Metode directe pentru sisteme liniare	19
3.1 Rezolvarea sistemelor triunghiulare	19
3.2 Eliminarea gaussiană și factorizarea LU	23
3.3 Forma compactă a factorizării LU	33
3.4 Factorizarea LDM^T	36
3.5 Pivotare	40
3.6 Factorizarea Cholesky	47
3.7 Matrice cu structură de bandă	52
Fișă de seminar și laborator: metode directe în ansamblu	57

Test de capitol	62
4 Norme, condiționare și analiza erorii	63
Aritmetica în virgulă mobilă	63
Extensie didactică: norme, perturbații și numărul de condiționare	73
4.1 Îmbunătățirea acurateței eliminării gaussiene	76
Fișă de seminar: eroare, condiționare și stabilitate	80
Test de capitol	85
5 Metode iterative pentru sisteme liniare	86
Extensie didactică: metode iterative ca probleme de punct fix	89
5.1 Jacobi, Gauss-Seidel și metode relaxate	92
5.2 Convergența metodelor Jacobi și Gauss-Seidel	96
Fișă de seminar și laborator: metode iterative	102
Test de capitol	107
6 Sisteme dreptunghiulare și aproximații	108
Extensie didactică: ce înseamnă un sistem dreptunghiular?	108
6.1 “Soluția” sistemelor supradeterminate	109
6.2 Problema celor mai mici pătrate și ajustarea datelor	114
6.3 Ajustarea datelor (data fitting)	117
Fișă de seminar: sisteme dreptunghiulare și modelarea datelor	122
Test de capitol	126
7 Regularizarea problemei celor mai mici pătrate și eliminarea zgomotului	127
Extensie didactică: regularizarea ca informație a priori	130
Test de capitol	135
8 Factorizarea QR	136
Extensie didactică: ortogonalizarea ca instrument numeric	139
8.1 Sisteme dreptunghiulare rezolvate cu factorizarea QR	142
Fișă de seminar și laborator: factorizarea QR	146
Test de capitol	150
9 Metode de descreștere și metoda gradientului	151
Extensie didactică: un sistem SPD ca problemă de minimizare	153
9.1 Metoda direcțiilor de descreștere	155
9.2 Metoda gradientului	159
9.3 Numărul de condiționare pentru funcții pătratice. Rescalarea problemei.	163
9.4 Rescalare	166

Fișă de seminar: metode de descreștere și gradient	174
Extensie: metoda gradientului conjugat	175
Test de capitol	179
10 Metoda Gauss–Newton pentru cele mai mici pătrate neliniare	180
Extensie didactică: Gauss–Newton ca secvență de probleme LS liniare	182
Fișă de seminar și laborator: Gauss–Newton	184
10.1 Problema celor mai mici pătrate pentru modele neliniare: ajustarea cercurilor și problema localizării	185
Test de capitol	194
11 Descompunerea în valori singulare și pseudoinversa	195
Extensie didactică: geometria și puterea descompunerii SVD	200
Fișă de seminar: SVD și pseudoinversa	202
Test de capitol	206
12 Aplicații SVD în compresia imaginilor	207
12.1 Descompunerea în valori singulare (SVD) și tensorii de ordinul al doilea	207
12.2 Compresia imaginilor în tonuri de gri	213
12.3 Compresia imaginilor color în spațiul RGB	217
12.4 Compresia imaginilor color în spațiul YCbCr	222
Test de capitol	228
13 Analiza componentelor principale (PCA)	229
13.1 Motivație: cum descriem date de dimensiune mare prin puține direcții?	229
13.2 Centrare, varianță și covarianță	229
13.3 Cel mai bun subspațiu afin în sensul celor mai mici pătrate	230
13.4 Principiul Rayleigh–Ritz și alegerea direcțiilor principale	232
13.5 PCA ca maximizare a varianței	234
13.6 Legătura exactă dintre PCA și SVD	234
13.7 Reducerea dimensiunii și reconstrucția	235
13.8 Câte componente păstrăm? Varianța explicată	236
13.9 Exemplu complet calculat	237
13.10Centrare sau standardizare?	238
13.11Algoritmul PCA în MATLAB	238
13.12Aplicație: o colecție de imagini ca nor de puncte	240
13.13Limitări și situații în care PCA trebuie folosită cu atenție	241
13.14Exerciții	242
13.15Rezumatul capitolului	243

Test de capitol	244
14 Aplicații integratoare: PageRank și sisteme de recomandare	245
14.1 De ce introducem aceste aplicații abia acum?	245
14.2 PageRank ca problemă de calcul științific	245
14.2.1 De la un graf orientat la o matrice stocastică	245
14.2.2 Modelul surferului aleator și teleportarea	246
14.2.3 Forma spectrală	247
14.2.4 Exemplu complet cu patru pagini	247
14.3 Un model de recomandare prin factorizare de rang mic	250
14.4 Ce noțiuni din curs au fost folosite?	252
Test de capitol	253
În loc de încheiere	254
Tabel de referință: costuri dominante	254
Fișă finală: compresia imaginilor ca proiect de calcul științific	255

Introducere în calcul științific

1.1 Ce este calculul științific?

Calculul științific studiază transformarea unui model matematic într-un calcul efectiv, controlabil și reproductibil. Într-o problemă reală nu este suficient să știm că o soluție există sau că poate fi scrisă formal printr-o formulă. Trebuie să construim un algoritm, să estimăm costul lui, să înțelegem efectul erorilor de rotunjire și al perturbațiilor din date și, în final, să putem verifica dacă rezultatul calculat este relevant pentru problema inițială.

Un exemplu elementar este sistemul liniar $Ax = b$. Formula $x = A^{-1}b$ este corectă atunci când A este inversabilă, dar nu descrie metoda numerică pe care ar trebui să o folosim. În practică, alegerea depinde de structură: un sistem algebric având o matrice triunghiulară se rezolvă prin substituție, un sistem algebric având o matrice generală prin eliminare Gauss și factorizare LU cu pivotare, un sistem algebric având o matrice simetrică pozitiv definită (adică $A = A^T$ și $x^T Ax > 0$ pentru orice $x \neq 0$) prin Cholesky, iar o problemă foarte mare cu o matrice rară – adică având majoritatea elementelor nule – poate favoriza o metodă iterativă. Aceeași problemă matematică poate conduce astfel la algoritmi foarte diferiți.

Trei niveluri care nu trebuie confundate

În calculul științific avem permanent *problema matematică*, *algoritmul* și *implementarea*. Problema poate fi bine sau prost condiționată; algoritmul poate fi stabil sau instabil; implementarea poate fi corectă sau poate conține erori. Un rezultat numeric bun cere înțelegerea tuturor celor trei niveluri menționate.

1.2 De ce apar vectorii și matricile în aplicații?

De exemplu, după discretizarea unei ecuații diferențiale, valorile necunoscute ale unei temperaturi, deplasări sau presiuni devin componentele unui vector, iar relațiile dintre ele sunt descrise de o matrice. În ajustarea datelor experimentale, liniile unei matrice conțin evaluările unui model în punctele măsurate. În procesarea imaginilor, o imagine în tonuri de gri este chiar o matrice de intensități, iar o imagine color poate fi privită ca un ansamblu de trei astfel de matrici. În probleme de rețea, matricile descriu conexiunile dintre noduri. În sisteme de recomandare, matricile pot codifica interacțiunile dintre utilizatori și obiecte.

Pentru a vedea mai concret cum apar aceste matrice, să considerăm câteva situații simple. Dacă discretizăm problema de ordinul întâi

$$u'(x) = g(x), \quad u(0) = 0,$$

pe o rețea uniformă $x_i = ih$, atunci aproximația cu diferență înainte

$$\frac{u_i - u_{i-1}}{h} \simeq g(x_i), \quad i = 1, \dots, n,$$

conduce, pentru necunoscutele $u_1, \dots, u_n$, la sistemul

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ -1 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 \\ 0 & \cdots & -1 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = h \begin{pmatrix} g(x_1) \\ g(x_2) \\ \vdots \\ g(x_n) \end{pmatrix}.$$

Astfel, matricea apare direct din relațiile dintre valorile vecine ale soluției pe rețea.

În ajustarea unei drepte $y \simeq a + bt$ la datele (t_i, y_i) , fiecare observație furnizează relația $a + bt_i \simeq y_i$. Reunind toate relațiile obținem

$$\begin{pmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_m \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \simeq \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix},$$

deci matricea apare direct prin evaluarea funcțiilor 1 și t în punctele măsurate.

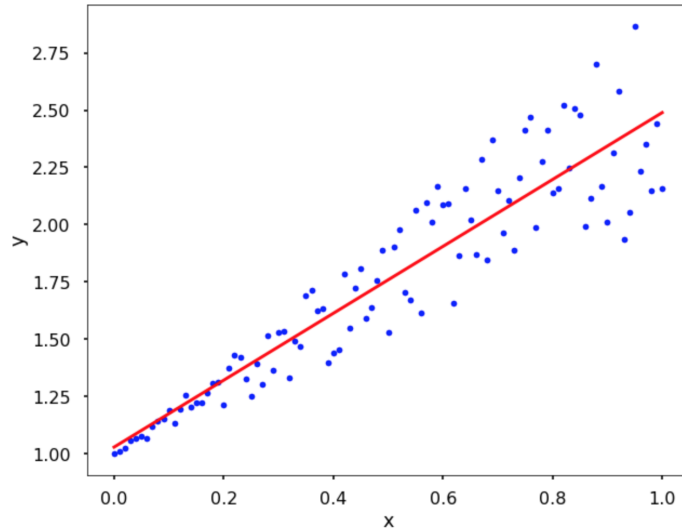


Figura 1.1: Date experimentale și o dreaptă de ajustare. Fiecare observație contribuie cu o relație liniară aproximativă pentru parametrii dreptei.

Într-o rețea cu n noduri, o descriere elementară este matricea de adiacență $A = (a_{ij})$, unde $a_{ij} = 1$ dacă nodurile i și j sunt conectate și $a_{ij} = 0$ în caz contrar. Astfel, chiar structura conexiunilor este codificată într-o matrice.

Aceste exemple au o caracteristică comună: dimensiunea problemei poate fi mare, datele pot conține zgomot, iar structura matricei este adesea la fel de importantă ca valorile numerice ale elementelor sale. De aceea, cursul nu tratează algebra liniară doar ca pe un set de identități, ci ca pe un limbaj pentru construirea și analiza algoritmilor.

Un exemplu practic important: o imagine ca matrice

O imagine în tonuri de gri cu $m \times n$ pixeli poate fi reprezentată printr-o matrice $A \in \mathbb{R}^{m \times n}$. Elementul a_{ij} descrie intensitatea pixelului din poziția (i, j) : valori mici corespund unor tonuri mai închise, iar valori mari unor tonuri mai deschise. Dacă această matrice admite o aproximare bună de rang mic, ea poate fi descrisă cu mult mai puține date. Mai târziu vom demonstra că descompunerea în valori singulare furnizează, într-un sens precis, cea mai bună aproximație de rang fix. Pentru moment este suficient să observăm ordinul de mărime: o imagine 4000×3000 conține 12 000 000 de valori, în timp ce o reprezentare de rang 50 necesită aproximativ $50(4000 + 3000 + 1)$ valori, adică aproximativ 350 000.

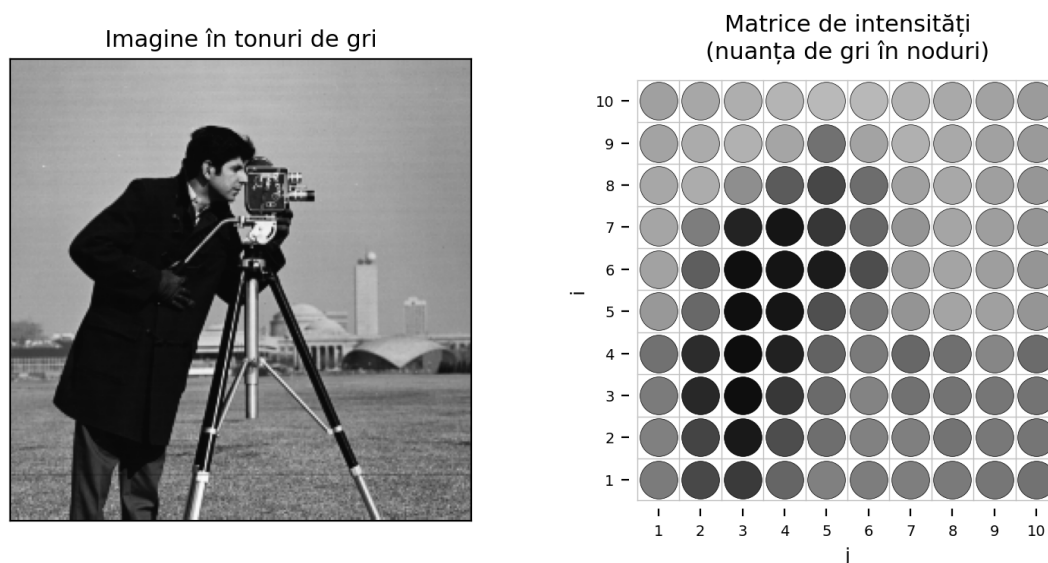


Figura 1.2: O imagine alb-negru și o reprezentare matricială grosieră. Fiecărui nod din grilă îi corespunde o nuanță de gri, deci o valoare a matricei de intensități.

Acest exemplu arată de ce noțiuni aparent abstracte precum rangul, ortogonalitatea și valorile singulare au consecințe practice directe. În capitolele finale vom reveni la imagini și vom construi efectiv algoritmi MATLAB de compresie.

Alte aplicații care vor reapărea în curs

Problema celor mai mici pătrate apare în calibrare, ajustarea unor modele la date și identificarea parametrilor. Regularizarea permite stabilizarea problemelor sensibile la zgomet. Metodele de gradient și Gauss–Newton apar în optimizare și ajustări neliniare. Matricile rare și metodele iterative apar în discretizări ale ecuațiilor diferențiale și în probleme de rețea de dimensiuni foarte mari.

Spre sfârșitul manualului vom reveni la mai multe dintre aceste noțiuni prin algoritmul PageRank pentru analiza rețelelor. El va fi introdus numai după ce vom avea la dispoziție normele induse, raza spectrală, metodele iterative, analiza convergenței și reprezentarea rară a matricelor. În acel moment vom putea nu doar să scriem o iterație, ci să demonstrăm de ce converge, să explicăm de ce este eficientă și să o implementăm corect în MATLAB. În aceeași parte finală vom discuta și un model simplificat de recomandare bazat pe factorizări de rang mic și pe problema celor mai mici pătrate.

1.3 Problema model: sistemul liniar

Unul dintre ideile centrale ale cursului este sistemul $Ax = b$, unde $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$ și $b \in \mathbb{R}^m$. În cazul pătratic, $m = n$, iar dacă A este inversabilă, soluția este unică. Din punct de vedere teoretic putem scrie $x = A^{-1}b$ sau putem invoca regula lui Cramer. Din punct de vedere numeric, întrebarea relevantă este însă cum obținem x cu un cost rezonabil și cu o eroare controlată.

Regula lui Cramer ilustrează bine diferența dintre formulă și algoritm. Calcularea determinantilor prin dezvoltare Laplace are un cost care crește factorial cu dimensiunea. Eliminarea Gauss și factorizarea LU au, pentru matrice dense, cost de ordin n^3 și permit apoi rezolvarea eficientă pentru mai mulți termeni liberi. Pentru matrici rare sau structurate costul poate fi redus substanțial.

Direct sau iterativ?

Metodele directe, precum LU , Cholesky și QR , produc în aritmetică exactă rezultatul după un număr finit de pași. Metodele iterative construiesc o succesiune $x^{(0)}, x^{(1)}, \dots$ care trebuie să tindă la soluție și sunt potrivite pentru multe probleme foarte mari și rare. Alegerea nu se face numai după dimensiunea matricei: contează simetria, pozitivă definită, raritatea, condiționarea și numărul de sisteme care trebuie rezolvate cu aceeași matrice.

1.4 Erori, stabilitate și condiționare

Calculatorul nu efectuează operații cu numere reale exacte. Numerele sunt reprezentate cu precizie finită, iar fiecare operație poate introduce o eroare de rotunjire. În plus, datele unei probleme provin adesea din măsurători și au propriile erori. Din acest motiv vom distinge două întrebări.

Prima este o proprietate a problemei: *cât de mult se modifică soluția dacă datele se modifică puțin?* Aceasta este problema condiționării. A doua este o proprietate a algoritmului: *cât de mult amplifică procedura numerică erorile inevitabile ale calculului?* Aceasta este problema stabilității. Un algoritm foarte bun nu poate transforma o problemă extrem de prost condiționată într-una insensibilă, dar un algoritm instabil poate degrada inutil chiar și o problemă bine condiționată.

1.5 Cum este organizat cursul?

Ordinea capitolelor urmărește dependența logică dintre noțiuni. Începem cu recapitularea matricelor simetrice, a valorilor proprii și a pozitivei definiri. Continuăm cu metode directe pentru sisteme liniare: substituție, eliminare Gauss, pivotare, factorizările LU , LDL^T și Cholesky, precum și matricile cu structură de bandă. Abia apoi introducem normele, raza spectrală, condiționarea și analiza erorilor.

Pe această bază studiem metodele iterative Jacobi, Gauss–Seidel, JOR și SOR și demonstrăm criterii de convergență. Urmează sistemele dreptunghiulare, problema celor mai mici pătrate, regularizarea și factorizarea QR . Problemele liniare sunt apoi reinterpretate ca probleme de optimizare, ceea ce conduce la metode de descreștere, metoda gradientului și metoda Gauss–Newton. Ultimele capitole dezvoltă descompunerea în valori singulare, pseudoinversa și aproximațiile de rang mic, cu aplicații în compresia imaginilor. Pe această bază introducem analiza componentelor principale (PCA), unde aceleași instrumente spectrale sunt folosite pentru

reducerea dimensiunii, aproximarea datelor prin subspații afine și analiza variației. Numai după parcurgerea acestor instrumente revenim la aplicații integratoare de tipul analizei rețelelor și al recomandării.

Ce trebuie să poată face studentul

La finalul cursului, studentul trebuie să poată trece în ambele direcții între matematică și algoritm: să deducă pașii unei metode din relațiile matematice, dar și să explice matematic un algoritm scris în MATLAB. La problemele mici, pașii trebuie să poată fi executați cu pixul pe hârtie; la problemele mai mari, implementarea trebuie însoțită de criterii de oprire, verificări prin reziduu, estimări de eroare și o discuție privind costul și stabilitatea.

Întrebări de început

1. De ce formula $x = A^{-1}b$ nu specifică încă un algoritm numeric?
2. Dați un exemplu în care structura matricei ar trebui să schimbe succesiunea de calcule folosită pentru rezolvarea sistemului.
3. De ce trebuie să distingem între problema matematică, algoritm și implementare?
4. De ce o matrice rară nu ar trebui, în general, stocată ca și cum toate elementele sale ar fi nenule?
5. Ce informații despre date, dimensiune și structură ar trebui notate înainte de a începe implementarea MATLAB?

Sarcini de aprofundare și orientare

Probleme de început: model, date și calcul

1. Alegeți una dintre aplicațiile discutate în capitol (procesare de imagini, recomandări, analiza rețelelor, ajustarea datelor). Identificați separat: datele de intrare, necunoscutele, rezultatul urmărit și etapa în care apare calculul numeric.
2. Pentru un sistem liniar simplu $Ax = b$, explicați diferența dintre a scrie o formulă matematică pentru soluție și a descrie o succesiune efectivă de operații care o calculează.
3. Dați câte un exemplu de matrice diagonală, triunghiulară, simetrică și rară. Pentru fiecare, indicați ce informație structurală este vizibilă înainte de efectuarea calculelor.
4. Comparați stocarea unei matrice 1000×1000 dense cu stocarea unei matrice de aceeași dimensiune având numai trei diagonale nenule. Numărați elementele care trebuie memorate în fiecare caz.
5. Construiți un exemplu în care aceeași problemă poate fi formulată cu 10, 100 și 1000 de necunoscute. Discutați ce se schimbă în volumul de calcul și ce rămâne neschimbat în model.
6. Formulați, în maximum cinci rânduri, diferența dintre model matematic, date, algoritm și implementare.

Laborator de orientare: aceeași structură la dimensiuni diferite

- Pași obligatorii de lucru.** 1. Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric.
2. Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă

este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Obiectiv. Observați rolul structurii și al dimensiunii înainte de introducerea instrumentelor de analiză a erorilor.

1. Construiți în MATLAB matricea tridiagonală cu 2 pe diagonală și -1 pe diagonalele vecine, pentru $n = 10, 100, 1000$.
2. Stocați matricea atât dens, cât și cu `sparse` și comparați memoria raportată de `whos`.
3. Reprezentați structura cu `spy(A)` și explicați ce informație oferă imaginea.
4. Alegeți vectorul $x_\star = \mathbf{1}$, formați $b = Ax_\star$ și folosiți `A\b` pentru a obține un vector calculat $\hat{x}$.
5. Comparați direct componentele lui $\hat{x}$ cu cele ale lui $x_\star$ și raportați cea mai mare diferență absolută dintre componente, fără a introduce încă noțiuni de normă sau condiționare.
6. Măsurați timpul pentru forma densă și forma rară și explicați diferențele observate.

Test de capitol

Test – Introducere în calcul științific

1. Analiză și justificare. Pentru fiecare dintre situațiile următoare precizați dacă afirmația se referă în primul rând la *problema matematică*, la *algoritm* sau la *implementare* și justificați răspunsul:

- a) o mică modificare a datelor produce o modificare mare a soluției;
- b) două programe care implementează aceeași metodă furnizează rezultate diferite deoarece unul conține o eroare de indexare;
- c) pentru aceeași problemă, o metodă exploatează faptul că matricea este triunghiulară, iar alta tratează matricea ca fiind generală.

2. Selectați afirmația/afirmațiile corecte și justificați.

- a) Pentru un sistem $Ax = b$, formula $x = A^{-1}b$ este:
 - A) întotdeauna un algoritm numeric complet;
 - B) o formulă matematică ce nu precizează în sine succesiunea efectivă de calcule;
 - C) o formulă care exploatează automat faptul că A poate fi rară sau structurată;
 - D) aplicabilă și dacă A nu este inversabilă.
- b) O implementare numerică serioasă trebuie să țină seama de:
 - A) structura datelor;
 - B) dimensiunea problemei;
 - C) verificarea rezultatului;
 - D) numai numărul de linii ale programului.

Recapitulare: matrice, valori proprii și pozitiva definire

Matrice simetrice pozitiv definite

Pozitiva definire a unei matrice

- Spunem că matricea $A \in \mathbb{R}^{n \times n}$ este simetrică dacă $A = A^T$,
- Spunem că matricea simetrică $A \in \mathbb{R}^{n \times n}$ este **pozitiv semidefinită**, notăm $A \succeq 0$, dacă $\langle Ax, x \rangle \geq 0$ pentru orice $x \in \mathbb{R}^n$, unde $\langle \cdot, \cdot \rangle$ reprezintă produsul scalar standard din $\mathbb{R}^n$.
- Spunem că matricea simetrică $A \in \mathbb{R}^{n \times n}$ este **pozitiv definită**, notăm $A \succ 0$, dacă $\langle Ax, x \rangle > 0$ pentru orice $x \in \mathbb{R}^n$, $x \neq 0$.

Valori proprii și vectori proprii

Fie $A \in \mathbb{R}^{n \times n}$. Un vector nenul $v \in \mathbb{C}^n$ se numește **vector propriu** pentru A dacă există $\lambda \in \mathbb{C}$ astfel încât

$$Av = \lambda v.$$

Scalarul λ se numește **valoarea proprie** corespunzătoare vectorului propriu v . În general, o matrice reală poate avea valori proprii complexe. Pentru o matrice simetrică reală, toate valorile proprii sunt însă reale.

Teoremă 2.0.1. Dacă $A \in \mathbb{R}^{n \times n}$ este simetrică și $Av = \lambda v$ pentru un vector $v \in \mathbb{C}^n \setminus \{0\}$, atunci $\lambda \in \mathbb{R}$.

Demonstrație. Pe $\mathbb{C}^n$ folosim produsul scalar hermitian standard

$$\langle z, w \rangle_{\mathbb{C}^n} = z^H w = \sum_{j=1}^n \bar{z}_j w_j,$$

unde $z^H = \bar{z}^T$ notează transpusa conjugată (sau adjuncta hermitiană) a vectorului z . Mai general, pentru o matrice complexă M , notația $M^H = \overline{M}^T$ reprezintă transpusa conjugată a lui M . Deoarece A este reală și simetrică, avem $A^H = A$.

Din $Av = \lambda v$ rezultă

$$v^H Av = \lambda v^H v.$$

Scalarul $v^H Av$ este real: într-adevăr,

$$\overline{v^H Av} = v^H A^H v = v^H Av.$$

Cum $v^H v = \|v\|_2^2 > 0$, obținem

$$\lambda = \frac{v^H Av}{v^H v} \in \mathbb{R}.$$

□

Teoremă 2.0.2 (Teorema spectrală pentru matrice reale simetrice). *Fie $A \in \mathbb{R}^{n \times n}$ simetrică. Atunci există o matrice ortogonală $Q \in \mathbb{R}^{n \times n}$ și o matrice diagonală reală $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ astfel încât*

$$A = Q\Lambda Q^T.$$

Echivalent, $\mathbb{R}^n$ admite o bază ortonormală formată din vectori proprii ai lui A .

Demonstrație. Demonstrăm prin inducție după n . Pentru $n = 1$ afirmația este evidentă. Presupunem rezultatul adevărat în dimensiunea $n - 1$. Polinomul caracteristic al lui A are o rădăcină în $\mathbb{C}$, deci A are un vector propriu complex $v \neq 0$. Teorema precedentă arată că valoarea proprie corespunzătoare λ este reală. Scriind $v = x + iy$, cu $x, y \in \mathbb{R}^n$, din $Av = \lambda v$ obținem $Ax = \lambda x$ și $Ay = \lambda y$; cel puțin unul dintre x, y este nenul. Există deci un vector propriu real unitar q_1 .

Considerăm subspațiul ortogonal $q_1^\perp$. Acesta este invariant la acțiunea lui A : dacă $z \perp q_1$, atunci, folosind simetria,

$$q_1^T Az = (Aq_1)^T z = (\lambda q_1)^T z = 0,$$

deci $Az \in q_1^\perp$. Restricția lui A la $q_1^\perp$ este din nou simetrică. Prin ipoteza de inducție există o bază ortonormală $q_2, \dots, q_n$ a lui $q_1^\perp$ formată din vectori proprii ai restricției, deci și ai lui A . Matricea $Q = (q_1 | \dots | q_n)$ este ortogonală, iar $Q^T A Q$ este diagonală cu valorile proprii pe diagonală. Rezultă $A = Q\Lambda Q^T$. □

Valorile proprii ale unei matrice simetrice $A \in \mathbb{R}^{n \times n}$ vor fi notate cu

$$\lambda_1(A) \geq \lambda_2(A) \geq \dots \geq \lambda_n(A).$$

Cea mai mare valoare proprie va fi notată cu $\lambda_{\max}(A) = \lambda_1(A)$ și cea mai mică valoare proprie va fi notată cu $\lambda_{\min}(A) = \lambda_n(A)$.

Teoremă 2.0.3 (Pozitiva definire și valorile proprii). *Fie $A \in \mathbb{R}^{n \times n}$ simetrică. Atunci A este pozitiv semidefinită dacă și numai dacă toate valorile sale proprii sunt nenegative, iar A este pozitiv definită dacă și numai dacă toate valorile sale proprii sunt strict pozitive.*

Demonstrație. Prin teorema spectrală, $A = Q\Lambda Q^T$, cu Q ortogonală și $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$. Pentru $x \in \mathbb{R}^n$ punem $y = Q^T x$. Deoarece Q este ortogonală, $x \neq 0$ dacă și numai dacă $y \neq 0$, iar

$$x^T Ax = y^T \Lambda y = \sum_{i=1}^n \lambda_i y_i^2.$$

Dacă toate $\lambda_i \geq 0$, suma este nenegativă pentru orice x , deci A este pozitiv semidefinită; dacă toate $\lambda_i > 0$, suma este strict pozitivă pentru orice $x \neq 0$, deci A este pozitiv definită. Reciproc, alegând $x = q_i$, coloana a i -a a lui Q , obținem $x^T Ax = \lambda_i$. Prin urmare pozitivă semidefinire implică $\lambda_i \geq 0$, iar pozitivă definire implică $\lambda_i > 0$ pentru orice i . □

Teoremă 2.0.4 (Criteriul lui Sylvester). *Fie $A \in \mathbb{R}^{n \times n}$ simetrică și notați $A_k = A(1:k, 1:k)$. Atunci A este pozitiv definită dacă și numai dacă*

$$\det A_k > 0, \quad k = 1, \dots, n.$$

Demonstrație. Dacă A este pozitiv definită, atunci fiecare submatrice principală A_k este pozitiv definită: pentru $z \in \mathbb{R}^k \setminus \{0\}$ completăm z cu zerouri până la un vector $x \in \mathbb{R}^n$ și avem $z^T A_k z = x^T A x > 0$. Valorile proprii ale lui A_k sunt deci pozitive, iar $\det A_k$, produsul lor, este pozitiv.

Pentru reciproca folosim inducția după n . Cazul $n = 1$ este imediat. Presupunem că toți minorii principali conducători ai lui A sunt pozitivi. Atunci, pentru blocul $B = A_{n-1}$, toți minorii principali conducători sunt pozitivi; prin ipoteza de inducție, B este pozitiv definită și, în particular, inversabilă. Scriem

$$A = \begin{pmatrix} B & c \\ c^T & a \end{pmatrix}.$$

Cantitatea

$$s = a - c^T B^{-1} c$$

se numește complementul Schur al blocului B în A . Pentru a vedea de unde apare, efectuăm o eliminare pe blocuri:

$$\begin{pmatrix} I & 0 \\ -c^T B^{-1} & 1 \end{pmatrix} \begin{pmatrix} B & c \\ c^T & a \end{pmatrix} = \begin{pmatrix} B & c \\ 0 & s \end{pmatrix}.$$

Matricea din stânga cu care am înmulțit are determinantul 1, iar matricea din dreapta este triunghiulară pe blocuri. Prin urmare,

$$\det A = \det B s = \det B (a - c^T B^{-1} c).$$

Din ipoteză, $\det A > 0$ și $\det B > 0$, deci

$$s = a - c^T B^{-1} c > 0.$$

Rămâne să arătăm că această informație implică pozitivă definire a lui A . Pentru $x = (y, t)^T \in \mathbb{R}^n$ avem

$$x^T A x = y^T B y + 2t c^T y + at^2.$$

Completând pătratul și folosind simetria lui B , obținem

$$\begin{aligned} (y + tB^{-1}c)^T B (y + tB^{-1}c) + s t^2 &= y^T B y + 2t c^T y + t^2 c^T B^{-1} c \\ &\quad + t^2 (a - c^T B^{-1} c) \\ &= x^T A x. \end{aligned}$$

Deoarece B este pozitiv definită, primul termen este nenegativ, iar $s > 0$. Dacă $t \neq 0$, termenul st^2 este strict pozitiv. Dacă $t = 0$ și $x \neq 0$, atunci $y \neq 0$, iar $y^T B y > 0$. Așadar

$$x^T A x > 0 \quad \text{pentru orice } x \neq 0,$$

deci A este pozitiv definită. □

O matrice $A \in \mathbb{R}^{n \times n}$ se numește diagonal dominantă pe linii dacă

$$|a_{ii}| \geq \sum_{j=1, j \neq i}^n |a_{ij}|, \quad \text{pentru } i = 1, \dots, n. \quad (2.1)$$

O matrice $A \in \mathbb{R}^{n \times n}$ se numește diagonal dominantă pe coloane dacă

$$|a_{ii}| \geq \sum_{j=1, j \neq i}^n |a_{ji}|, \quad \text{pentru } i = 1, \dots, n, \quad (2.2)$$

Dacă inegalitățile sunt stricte, spunem că A este strict diagonal dominantă (pe linii, respectiv, pe coloane).

Teoremă 2.0.5. *O matrice simetrică strict diagonal dominantă cu elemente strict pozitive pe diagonală este pozitiv definită.*

Demonstrație. Fie $A = (a_{ij})$ simetrică, cu $a_{ii} > \sum_{j \neq i} |a_{ij}|$ pentru fiecare i . Pentru $x \in \mathbb{R}^n$ avem, prin simetrie,

$$x^T Ax = \sum_{i=1}^n a_{ii} x_i^2 + 2 \sum_{1 \leq i < j \leq n} a_{ij} x_i x_j.$$

Folosind $2|x_i x_j| \leq x_i^2 + x_j^2$, obținem

$$2a_{ij} x_i x_j \geq -2|a_{ij}| |x_i x_j| \geq -|a_{ij}|(x_i^2 + x_j^2).$$

Prin însumare,

$$x^T Ax \geq \sum_{i=1}^n \left(a_{ii} - \sum_{j \neq i} |a_{ij}| \right) x_i^2.$$

Fiecare coeficient dintre paranteze este strict pozitiv. Pentru $x \neq 0$ există cel puțin un $x_i \neq 0$, deci partea dreaptă este strict pozitivă. Prin urmare A este pozitiv definită. $\square$

Reciproca nu este valabilă. De exemplu,

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 5 \end{pmatrix}$$

este simetrică și pozitiv definită, deoarece minorii principali conducători sunt $1 > 0$ și $\det A = 1 > 0$, dar prima linie nu este diagonal dominantă: $|a_{11}| = 1 < 2 = |a_{12}|$. Așadar, diagonal-dominanța strictă cu diagonală pozitivă este o condiție suficientă, nu necesară, pentru pozitivă definire.

Extensie didactică: interpretarea spectrală a matricelor

Pentru metodele numerice, valorile proprii nu sunt doar obiecte algebrice. Ele descriu direcții în care acțiunea unei matrice are o formă simplă și vor reapărea în capitolele următoare în analiza mai multor algoritmi.

Matrice simetrică pozitiv definită

Pentru o matrice simetrică A , condiția $x^T Ax > 0$ pentru orice $x \neq 0$ înseamnă că forma pătratică asociată are curbura pozitivă în orice direcție. Prin teorema spectrală, $A = Q\Lambda Q^T$, unde Q este ortogonală și $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$. Dacă scriem $y = Q^T x$, atunci $x^T Ax = \sum_i \lambda_i y_i^2$. De aici se vede imediat echivalența dintre pozitivă definire și condiția $\lambda_i > 0$ pentru toate valorile proprii.

Raportul Rayleigh $R_A(x) = \frac{x^T A x}{x^T x}$, definit pentru $x \neq 0$, verifică $\lambda_{\min}(A) \leq R_A(x) \leq \lambda_{\max}(A)$ pentru orice matrice simetrică. El permite astfel să legăm proprietățile geometrice ale formei pătratice de spectrul matricei. În cazul unei funcții pătratice $f(x) = \frac{1}{2}x^T A x - b^T x$, aceste două valori proprii extreme vor determina cât de alungite sunt curbele de nivel și, ulterior, cât de repede converge metoda gradientului.

Exemplu ghidat

Fie $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$. Vectorii $q_1 = (1, 1)^T/\sqrt{2}$ și $q_2 = (1, -1)^T/\sqrt{2}$ sunt vectori proprii pentru valorile proprii 3, respectiv 1. Pentru $x = \alpha q_1 + \beta q_2$ avem $x^T A x = 3\alpha^2 + \beta^2$. Pozitivitatea este transparentă în aceste coordonate. În plus, raportul dintre cea mai mare și cea mai mică valoare proprie este 3; pentru moment îl interpretăm numai ca pe o măsură geometrică a diferenței dintre cele două axe ale elipsei.

De ce contează ortogonalitatea?

Transformările ortogonale păstrează produsul scalar și lungimile euclidiene. Această proprietate geometrică va fi exploatată ulterior de factorizări numerice bazate pe transformări ortogonale; aici ne limităm la interpretarea ei algebrică și geometrică.

Laborator: spectru și formă pătratică

Pași obligatorii de lucru. **1.** Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Alegeți o matrice simetrică 2×2 pozitiv definită. Calculați valorile proprii cu `eig`, apoi reprezentați curbele de nivel ale lui $f(x) = \frac{1}{2}x^T A x$. Rotiți sistemul de coordonate cu matricea vectorilor proprii și verificați că în noile coordonate curbele de nivel sunt elipse cu axele aliniate cu axele coordonatelor. Modificați raportul $\lambda_{\max}/\lambda_{\min}$ și observați cum se alungesc elipsele.

Exerciții și verificarea înțelegerii

1. Demonstrați că o matrice simetrică A este pozitiv definită dacă și numai dacă toate valorile sale proprii sunt strict pozitive.
2. Pentru $A = \begin{pmatrix} 4 & -1 \\ -1 & 1 \end{pmatrix}$ verificați pozitivă definire prin două metode: criteriul minorilor principali și calculul valorilor proprii.
3. Arătați că, pentru $A = A^T$, maximul raportului Rayleigh este $\lambda_{\max}(A)$ și minimul este $\lambda_{\min}(A)$.
4. Construiți o matrice nesimetrică ale cărei valori proprii sunt pozitive, dar pentru care $x^T A x$ nu este pozitiv pentru orice $x \neq 0$. Explicați de ce ipoteza de simetrie este necesară în afirmația anterioară.

Mini-proiect: un model elementar cu arcuri

Etapele proiectului. **1. Modelare:** definiți toate mărimile și termenii specifici aplicației înainte de a-i folosi și formulați problema matematică. **2. Derivare:** justificați metoda și executați manual un exemplu mic. **3. Algoritm:** prezentați pseudocodul și analizați costul. **4. MATLAB:** scrieți o implementare proprie, modulară. **5. Experimente:** variați cel puțin un parametru și comparați cu o metodă de referință. **6. Validare:** folosiți numai criterii definite până în acel punct al cursului. **7. Explicație:** includeți o concluzie matematică, nu doar grafice sau tabele.

Considerăm n puncte materiale care se pot deplasa numai pe o dreaptă, cu deplasări $x_1, \dots, x_n$, legate succesiv prin arcuri identice. Pentru un arc, presupunem legea liniară $F = -k\delta$: forța este opusă alungirii δ și proporțională cu ea. Capetele lanțului sunt fixate. Scrierea echilibrului forțelor pentru fiecare punct conduce la un sistem $Kx = f$. În acest context, numim *matrice de rigiditate* matricea K ce leagă deplasările de forțele elastice. După împărțirea prin k , ea este tridiagonală, cu 2 pe diagonală și -1 pe diagonalele vecine. Deduceți această matrice din ecuațiile de echilibru, apoi studiați numeric valorile proprii pentru mai multe valori ale lui n și verificați pozitivă definire. Interpretarea fizică a vectorilor proprii este opțională; din punct de vedere matematic, urmăriți forma lor și variația valorilor proprii.

Rezolvare orientativă**Pozitiva definire prin două metode**

Pentru $A = \begin{pmatrix} 4 & -1 \\ -1 & 1 \end{pmatrix}$, criteriul minorilor principali dă $\Delta_1 = 4 > 0$ și $\Delta_2 = 4 - 1 = 3 > 0$, deci A este SPD. Valorile proprii sunt rădăcinile lui $\lambda^2 - 5\lambda + 3 = 0$, adică $\lambda_{1,2} = (5 \pm \sqrt{13})/2$, ambele pozitive. Cele două verificări exprimă aceeași proprietate, dar din perspective diferite. Pentru $x = (x_1, x_2)^T$, forma pătratică este $4x_1^2 - 2x_1x_2 + x_2^2$. Completând pătratul, obținem $3x_1^2 + (x_2 - x_1)^2 > 0$ pentru $x \neq 0$, o a treia demonstrație. În aplicații, alegerea demonstrației depinde de informația disponibilă: structură explicită, spectru sau factorizare.

Situație practică: energie elastică discretă

În modelul elementar cu arcuri descris mai sus, x_i reprezintă deplasările punctelor, iar f_i forțele exterioare. Matricea K care apare în relația de echilibru $Kx = f$ a fost numită matrice de rigiditate. Pentru trei deplasări interioare și arcuri identice (după o scalare cu constanta arcului), energia poate avea forma $E(x) = \frac{1}{2}x^T Kx - f^T x$, unde K este tridiagonală cu 2 pe diagonală și -1 pe vecini. Termenul $x^T Kx$ se poate rescrie ca sumă de pătrate ale diferențelor dintre deplasări și ale deplasărilor față de capetele fixate. De aici se vede direct că K este SPD. Condiția de staționaritate a acestei funcții pătratice conduce la $Kx = f$. Deoarece K este pozitiv definită, sistemul are soluție unică. Cholesky și, ulterior, metoda gradientului vor exploata exact această structură.

Fișă de seminar: geometria matricelor simetrice

Problema rezolvată 1: pozitivă definire prin trei criterii

Considerăm $A = \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}$. Putem verifica pozitivă definire în trei moduri. Prin criteriul Sylvester, minorii principali sunt $4 > 0$ și $\det A = 8 > 0$. Prin completare de pătrate, $x^T A x = 4x_1^2 + 4x_1x_2 + 3x_2^2 = (2x_1 + x_2)^2 + 2x_2^2 > 0$ pentru $x \neq 0$. Prin valori proprii, polinomul caracteristic este $\lambda^2 - 7\lambda + 8$, ale cărui rădăcini $(7 \pm \sqrt{17})/2$ sunt pozitive. Cele trei teste exprimă aceeași proprietate, dar sunt utile în contexte diferite.

Pentru calcul numeric, nu vom calcula de obicei valorile proprii doar pentru a decide dacă o matrice SPD admite Cholesky; algoritmul Cholesky însuși poate servi drept test practic. În demonstrații, însă, perspectiva spectrală explică direct geometria formei pătratice.

Problema rezolvată 2: o schimbare ortogonală de coordonate

Dacă $Q^T Q = I$, atunci produsul scalar este păstrat: $(Qx)^T (Qy) = x^T y$. În particular, unghiurile și lungimile euclidiene nu se schimbă. Această observație explică de ce diagonalizarea ortogonală a unei matrice simetrice păstrează fidel geometria problemei.

Laborator integrator 2: spectru și curbe de nivel

Pași obligatorii de lucru. 1. Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. 2. Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. 3. Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. 4. Testați pe un caz cu rezultat cunoscut. 5. Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. 6. Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Generați matrici simetrice 2×2 cu valori proprii prescrise și vectori proprii roțiți. Reprezentați curbele $x^T A x = 1$. Modificați raportul valorilor proprii de la 1 la 10^4 și observați elipsele. Conectați forma geometrică cu raportul dintre valorile proprii extreme.

Probleme pentru seminar

1. Arătați că o matrice SPD este nesară.
2. Demonstrați că suma a două matrici SPD este SPD.
3. Dacă A este SPD și Q ortogonală, arătați că $Q^T A Q$ este SPD și are aceleași valori proprii.
4. Construiți o matrice simetrică indefinită cu determinant pozitiv. De ce determinantul singur nu decide pozitivă definire în dimensiune mai mare?
5. Pentru $A = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$ determinați intervalul lui ρ pentru care A este SPD.
6. Calculați valorile proprii în problema precedentă și studiați ce se întâmplă cu cea mai mică valoare proprie când $|\rho| \rightarrow 1$.
7. Pentru o matrice simetrică diagonalizată ortogonal, explicați cum se citește acțiunea matricei pe fiecare direcție proprie.
8. Explicați de ce o matrice Gram $G = X^T X$ poate fi singulară chiar dacă toate elementele diagonale sunt pozitive.

Mini-proiect: atlas de matrici

Etapele proiectului. **1. Modelare:** definiți toate mărimile și termenii specifici aplicației înainte de a-i folosi și formulați problema matematică. **2. Derivare:** justificați metoda și executați manual un exemplu mic. **3. Algoritm:** prezentați pseudocodul și analizați costul. **4. MATLAB:** scrieți o implementare proprie, modulară. **5. Experimente:** variați cel puțin un parametru și comparați cu o metodă de referință. **6. Validare:** folosiți numai criterii definite până în acel punct al cursului. **7. Explicație:** includeți o concluzie matematică, nu doar grafice sau tabele.

Construiți un script MATLAB cu exemple pentru cel puțin opt clase: diagonală, ortogonală, SPD, semidefinită, indefinită, strict diagonal dominantă, singulară și aproape singulară. Pentru fiecare afișați determinantul, rangul și valorile proprii și verificați proprietățile structurale relevante folosind numai criteriile introduse în acest capitol.

Sarcini de aprofundare și sinteză

Probleme de aprofundare: spectru, forme pătratice și geometrie

1. Pentru $A = \begin{pmatrix} 4 & 1 \\ 1 & 3 \end{pmatrix}$ calculați valorile proprii și o bază ortonormală de vectori proprii. Scrieți diagonalizarea ortogonală $A = Q\Lambda Q^T$.
2. Construiți o matrice simetrică 2×2 cu valori proprii prescrise 1 și 100, ale cărei direcții proprii sunt obținute printr-o rotație de 30° . Scrieți explicit Q , Λ și $A = Q\Lambda Q^T$.
3. Demonstrați că pentru orice matrice B cu rang coloană complet, matricea Gram $B^T B$ este pozitiv definită.
4. Dați o matrice reală 2×2 cu valori proprii complexe. Explicați de ce aceasta nu contrazice teorema spectrală pentru matrici reale simetrice.
5. Demonstrați, folosind teorema spectrală, că o matrice simetrică este pozitiv definită dacă și numai dacă toate valorile proprii sunt strict pozitive.
6. Pentru o matrice simetrică A , arătați că valorile expresiei $x^T A x$ pe vectori unitari sunt cuprinse între cea mai mică și cea mai mare valoare proprie.
7. Verificați criteriul lui Sylvester pentru trei matrici simetrice alese astfel încât una să fie pozitiv definită, una pozitiv semidefinită și una indefinită. Comparați cu semnele valorilor proprii.
8. Construiți o matrice simetrică strict diagonal dominantă cu diagonală pozitivă și verificați direct pozitivă definire. Construiți apoi o matrice pozitiv definită care nu este diagonal dominantă.
9. Considerați matricea

$$L = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}.$$

Calculați valorile proprii și identificați vectorul propriu asociat valorii proprii zero. Explicați algebric de ce suma elementelor de pe fiecare linie fiind zero implică existența acestui vector propriu. Interpretări ale unei asemenea matrice prin grafuri vor fi introduse numai ulterior.

10. Comparați două matrici similare A și SAS^{-1} . Verificați că au aceleași valori proprii, dar vectorii proprii se transformă prin S .

Laborator: matrice simetrice cu spectru prescris

Pași obligatorii de lucru. **1.** Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Obiectiv. Construiți exemple pentru care spectrul este cunoscut dinainte și verificați teorema spectrală.

1. În dimensiunea 2, luați explicit matricea de rotație $Q(\theta)$ și $\Lambda = \text{diag}(1, 10)$. Formați $A = Q\Lambda Q^T$ pentru mai multe valori ale lui θ .
2. Verificați numeric că A este simetrică și că $AQ = Q\Lambda$, comparând element cu element cele două matrice.
3. Reprezentați curbele $x^T A x = 1$ pentru cel puțin trei rapoarte diferite între valorile proprii și explicați schimbarea geometrică.

4. Generați mai mulți vectori unitari x și calculați $x^T Ax$. Verificați experimental intervalul determinat de valorile proprii extreme.
5. Repetați construcția în dimensiunea 3 folosind o matrice ortogonală simplă construită din rotații plane succesive.
6. Formulați o concluzie despre relația dintre valorile proprii, direcțiile proprii și geometria formei pătratice.

Mini-proiect: axele unui nor de puncte

Etapele proiectului. **1. Modelare:** definiți toate mărimile și termenii specifici aplicației înainte de a-i folosi și formulați problema matematică. **2. Derivare:** justificați metoda și executați manual un exemplu mic. **3. Algoritm:** prezentați pseudocodul și analizați costul. **4. MATLAB:** scrieți o implementare proprie, modulară. **5. Experimente:** variați cel puțin un parametru și comparați cu o metodă de referință. **6. Validare:** folosiți numai criterii definite până în acel punct al cursului. **7. Explicație:** includeți o concluzie matematică, nu doar grafice sau tabele.

Nivel: mediu. Considerați punctele

$$(-3, -2), (-2, -1), (-1, 0), (1, 0), (2, 1), (3, 2),$$

al căror centru este originea, și formați matricea $X \in \mathbb{R}^{6 \times 2}$ având aceste puncte pe linii. Definiți $C = X^T X$.

1. Calculați C și verificați că este simetrică și pozitiv semidefinită. Justificați direct relația $z^T C z = \|Xz\|_2^2$.
2. Calculați valorile proprii și o bază ortonormală de vectori proprii ai lui C .
3. Reprezentați punctele în plan și trasați prin origine cele două axe determinate de vectorii proprii. Comparați orientarea lor cu forma norului de puncte.
4. Rotiți toate punctele cu același unghi, folosind o matrice ortogonală Q , și formați noua matrice $\tilde{C}$. Verificați că valorile proprii nu se schimbă, iar axele se rotesc odată cu punctele.
5. Modificați coordonatele astfel încât norul să fie mai alungit și urmăriți raportul dintre cea mai mare și cea mai mică valoare proprie.

Test de capitol

Test – Matrice, valori proprii și pozitiva definire

1. Calcul explicit. Se consideră

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.$$

- a) Determinați valorile proprii și câte un vector propriu pentru fiecare dintre ele.
- b) Construiți o bază ortonormală de vectori proprii și scrieți descompunerea spectrală $A = Q\Lambda Q^T$.
- c) Decideți, justificat în două moduri diferite, dacă A este pozitiv definită.

2. Selectați afirmația/afirmațiile corecte și justificați. Pentru o matrice reală simetrică A :

- A) toate valorile proprii sunt reale;
- B) vectorii proprii corespunzători unor valori proprii distincte pot fi aleși ortogonali;
- C) A este pozitiv definită dacă și numai dacă toate valorile proprii sunt strict pozitive;
- D) orice matrice cu determinant pozitiv este pozitiv definită.

Metode directe pentru sisteme liniare

3.1 Rezolvarea sistemelor triunghiulare

Rezolvarea sistemelor inferior triunghiulare

Pentru exemplificare să considerăm sistemul

$$\begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \quad (3.1)$$

unde $l_{ii} \neq 0$, $i = 1, 2, 3$.

Ultima condiție ne asigură că matricea sistemului este inversabilă, soluția fiind dată de

$$\begin{aligned} x_1 &= \frac{b_1}{l_{11}}, \\ x_2 &= \frac{b_2 - l_{21}x_1}{l_{22}}, \\ x_3 &= \frac{b_3 - l_{31}x_1 - l_{32}x_2}{l_{33}}. \end{aligned}$$

Acest algoritm se numește metoda substituției înainte (*forward substitution*).

În general, pentru $n \geq 2$ avem

$$\begin{aligned} x_1 &= \frac{b_1}{l_{11}}, \\ x_i &= \frac{b_i - \sum_{j=1}^{i-1} l_{ij}x_j}{l_{ii}}, \quad i = 2, \dots, n. \end{aligned} \quad (3.2)$$

Numărarea operațiilor aritmetice dă un cost de ordin n^2 pentru substituția înainte. Pentru comparația dintre metode este suficient, de regulă, termenul dominant. Aplicarea regulii lui Cramer prin dezvoltarea determinantelor are, în schimb, un cost de ordin factorial.

Rezolvarea sistemelor superior triunghiulare

Pentru exemplificare să considerăm sistemul

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \quad (3.3)$$

unde $u_{ii} \neq 0$, $i = 1, 2, 3$.

Ultima condiție ne asigură că matricea sistemului este inversabilă, soluția în cazul general fiind dată de

$$\begin{aligned} x_n &= \frac{b_n}{u_{nn}}, \\ x_i &= \frac{b_i - \sum_{j=i+1}^n u_{ij}x_j}{u_{ii}}, \quad i = n-1, \dots, 1. \end{aligned} \quad (3.4)$$

Acest algoritm se numește metoda substituției înapoi (*backward substitution*).

Avem tot n^2 operații.

Pentru implementare

Pentru implementare ar fi indicat să stocăm doar elementele nenule atunci când avem de rezolvat sistemele triunghiulare.

Completare didactică: substituția înainte și înapoi

Rezolvarea sistemelor triunghiulare este “motorul” ascuns al multor metode directe. Factorizările LU , Cholesky și QR reduc o problemă aparent complicată la una sau mai multe rezolvări triunghiulare. De aceea algoritmul trebuie înțeles și implementat fără calcularea vreunei inverse.

Pentru un sistem inferior triunghiular $Lx = b$, ecuația i conține necunoscutele $x_1, \dots, x_i$, iar primele $i-1$ au fost deja calculate. Rezultă formula scurtă $x_i = (b_i - \sum_{j=1}^{i-1} l_{ij}x_j)/l_{ii}$. În cazul unui sistem superior triunghiular se procedează în ordine inversă.

Exemplu ghidat

Considerăm

$$L = \begin{pmatrix} 2 & 0 & 0 \\ -1 & 1 & 0 \\ 3 & 2 & 4 \end{pmatrix}, \quad b = \begin{pmatrix} 4 \\ 1 \\ 10 \end{pmatrix}.$$

Prima ecuație dă $x_1 = 2$. A doua devine $-2 + x_2 = 1$, deci $x_2 = 3$. Ultima ecuație este $6 + 6 + 4x_3 = 10$, de unde $x_3 = -\frac{1}{2}$. Observați că nu a fost nevoie de nicio operație asupra unei linii deja rezolvate; fiecare pas utilizează doar valorile calculate anterior.

Pentru o matrice triunghiulară densă de ordin n , numărul de termeni utilizați în sume este $1 + 2 + \dots + (n-1) = n(n-1)/2$. Costul este deci de ordin $O(n^2)$, mult mai mic decât costul unei factorizări dense, de ordin $O(n^3)$.

Implementare fără funcții de rezolvare

Pași obligatorii de lucru. 1. Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Scrieți o funcție MATLAB pentru substituția înainte.

```
function x = forward_sub(L,b)
    n = length(b);
    x = zeros(n,1);
    for i = 1:n
        s = L(i,1:i-1)*x(1:i-1);
        x(i) = (b(i)-s)/L(i,i);
    end
end
```

Scrieți apoi funcția analogă `back_sub(U,b)`. Testați ambele funcții pe matrice triunghiulare aleatoare și comparați cu operatorul `\`. Introduceți explicit un test care oprește algoritmul atunci când un element diagonal este zero sau prea mic în raport cu scala problemei.

Exerciții și verificarea înțelegerii

1. Rezolvați manual sistemul cu $U = \begin{pmatrix} 1 & -2 & 1 \\ 0 & 3 & 6 \\ 0 & 0 & 2 \end{pmatrix}$ și $b = (0, 9, 4)^T$.
2. Calculați exact numărul de înmulțiri, adunări/scăderi și împărțiri necesare substituției înainte pentru o matrice densă $n \times n$.
3. Dacă L este triunghiulară inferior și toate elementele diagonale sunt nenule, demonstrați unicitatea soluției fără a invoca formula determinantului.
4. Modificați algoritmul pentru cazul în care L are diagonala identică cu 1. Ce operații dispar?
5. Explicați de ce rezolvarea separată a n sisteme triunghiulare cu termenii liberi $e_1, \dots, e_n$ produce inversa matricei, dar este inutilă atunci când avem nevoie de un singur vector $L^{-1}b$.

Mini-proiect: influența structurii rare

Etapele proiectului. **1. Modelare:** definiți toate mărimile și termenii specifici aplicației înainte de a-i folosi și formulați problema matematică. **2. Derivare:** justificați metoda și executați manual un exemplu mic. **3. Algoritm:** prezentați pseudocodul și analizați costul. **4. MATLAB:** scrieți o implementare proprie, modulară. **5. Experimente:** variați cel puțin un parametru și comparați cu o metodă de referință. **6. Validare:** folosiți numai criterii definite până în acel punct al cursului. **7. Explicație:** includeți o concluzie matematică, nu doar grafice sau tabele.

Construiți matrice triunghiulare dense și rare de dimensiuni până la limita rezonabilă a calculatorului folosit. Comparați memoria și timpul de rezolvare. Pentru matricea rară, folosiți doar câteva diagonale nenule. Explicați de ce o implementare care parcurge toate elementele unei matrice rare poate pierde avantajul structurii rare.

Rezolvare orientativă

Substituție înapoi, fără scurtături

Pentru $U = \begin{pmatrix} 1 & -2 & 1 \\ 0 & 3 & 6 \\ 0 & 0 & 2 \end{pmatrix}$ și $b = (0, 9, 4)^T$, ultima ecuație dă $2x_3 = 4$, deci $x_3 = 2$. A doua devine $3x_2 + 12 = 9$, deci $x_2 = -1$. Prima este $x_1 - 2(-1) + 2 = 0$, de unde $x_1 = -4$. Verificarea este obligatorie: $Ux = (-4 + 2 + 2, -3 + 12, 4)^T = (0, 9, 4)^T$.
 Observați că ordinea este impusă de structură. Dacă am încerca să calculăm x_1 mai întâi, prima ecuație conține încă două necunoscute. Algoritmul nu este doar o formulă, ci o ordine de dependență între calcule.

Situație practică: mulți termeni liberi

Dacă L este fixă și avem matricea $B = (b^{(1)}, \dots, b^{(p)})$, putem rezolva $LX = B$ coloană cu coloană. În MATLAB, operațiile pot fi vectorizate astfel încât la pasul i să actualizăm simultan toate cele p coloane. Costul crește aproximativ proporțional cu p , dar citirea factorului L poate fi reutilizată eficient din memorie. Acest scenariu apare după o factorizare LU când același model este excitat de mai multe forțe.

Exemplu rezolvat suplimentar: substituția înainte fără salturi de calcul

Considerăm sistemul $Lx = b$, cu

$$L = \begin{pmatrix} 2 & 0 & 0 \\ -1 & 3 & 0 \\ 4 & 2 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 4 \\ 5 \\ 3 \end{pmatrix}.$$

Prima ecuație conține o singură necunoscută, deci $2x_1 = 4$ și $x_1 = 2$. A doua ecuație devine $-x_1 + 3x_2 = 5$, de unde $3x_2 = 7$ și $x_2 = 7/3$. Abia după ce primele două componente sunt cunoscute folosim ultima ecuație: $4x_1 + 2x_2 + x_3 = 3$. Rezultă $x_3 = 3 - 8 - 14/3 = -29/3$. Observați că fiecare pas folosește numai componente deja calculate; aceasta este proprietatea structurală care face posibilă substituția înainte.

Verificarea nu trebuie omisă. Pentru un exemplu mic calculăm Lx și verificăm dacă recuperăm vectorul b . În implementările numerice putem compara direct componentele celor doi vectori; instrumentele sistematice pentru măsurarea acestei diferențe vor fi introduse în capitolul despre norme și analiza erorii.

De ce costul este de ordin n^2 ?

La pasul i trebuie evaluată o sumă cu $i - 1$ termeni. Numărul total de înmulțiri din aceste sume este $0 + 1 + \dots + (n - 1) = n(n - 1)/2$. Același ordin apare pentru adunări, iar cele n împărțiri nu schimbă ordinul dominant. De aceea substituția triunghiulară costă $O(n^2)$, mult mai puțin decât factorizarea unei matrice dense, care costă $O(n^3)$.

Sarcină de implementare: forwardSubstitution

Pași obligatorii de lucru. 1. Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric.
2. Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă

este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Scrieți o funcție MATLAB care primește L și b și întoarce x . Funcția trebuie să verifice dacă dimensiunile sunt compatibile și să semnaleze un pivot diagonal nul sau prea mic. Nu folosiți operatorul $\backslash$ sau o rutină de rezolvare existentă. Testați funcția astfel:

1. generați un vector x_* aleator și o matrice triunghiulară L cu diagonală depărtată de zero;
2. construiți $b = Lx_*$;
3. calculați $\hat{x}$ cu funcția proprie;
4. comparați componentă cu componentă $\hat{x}$ cu x_* și $L\hat{x}$ cu b ; raportați cea mai mare diferență în valoare absolută în fiecare comparație;
5. repetați pentru $n = 10, 100, 1000$ și măsurați timpul.

Explicați de ce această metodă de testare este mai informativă decât alegerea arbitrară a unui termen liber pentru care soluția exactă nu este cunoscută.

Exerciții suplimentare

1. Rezolvați prin substituție înapoi sistemul cu $U = \begin{pmatrix} 2 & -1 & 3 \\ 0 & 4 & 2 \\ 0 & 0 & -5 \end{pmatrix}$ și $b = (6, 6, -10)^T$.

Verificați soluția prin înmulțire.

2. Arătați că o matrice triunghiulară este inversabilă dacă și numai dacă toate elementele diagonale sunt nenule. Folosiți argumentul pentru a justifica existența fiecărui pas al substituției.
3. Modificați algoritmul pentru cazul în care L este triunghiulară unitară. Câte împărțiri se economisesc?
4. Pentru o matrice triunghiulară cu bandă de lățime p , arătați că substituția poate fi făcută în $O(np)$ operații, nu în $O(n^2)$.
5. Construiți o matrice triunghiulară cu elemente diagonale de ordine foarte diferită și urmăriți numeric substituția. Ce dificultăți observați când unul dintre elementele diagonale este foarte mic?

3.2 Eliminarea gaussiană și factorizarea LU

Eliminarea gaussiană

Eliminarea gaussiană ne ajută să reducem un sistem

$$Ax = b$$

la un sistem triunghiular (sau la două sisteme triunghiulare) prin transformări succesive ale sistemului în sisteme echivalente

$$A^{(1)}x = b^{(1)} \rightarrow A^{(2)}x = b^{(2)} \rightarrow \dots \rightarrow A^{(k)}x = b^{(k)}.$$

Presupunem că la fiecare pas elementul $a_{kk}^{(k)}$ al matricei $A^{(k)}$ este nenul.

Acest element va fi numit **pivot**.

Presupunem că A este inversabilă, adică sistemul admite soluție unică.

Pasul 1

Plecăm de la sistemul

$$\underbrace{\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & \cdots & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \cdots & \cdots & a_{2n}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & \cdots & \cdots & a_{3n}^{(1)} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \cdots & \cdots & a_{nn}^{(1)} \end{pmatrix}}_{\equiv A} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_n^{(1)} \end{pmatrix}$$

Definim multiplicatorii $m_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}$, $i = 2, 3, \dots, n$, înmulțim prima linie cu m_{i1} , pe rând, și scădem rezultatele din linia $i = 2, 3, \dots, n$, respectiv.

Pasul 1

Se obține astfel un nou sistem, echivalent cu cel inițial

$$\underbrace{\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & \cdots & a_{2n}^{(2)} \\ 0 & a_{32}^{(2)} & \cdots & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & \cdots & a_{nn}^{(2)} \end{pmatrix}}_{:=A^{(2)}} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(2)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{pmatrix}.$$

Pasul k

Repetând procedeul, la pasul k se obține astfel un nou sistem, echivalent cu cel inițial

$$\underbrace{\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & \cdots & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & \cdots & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nk}^{(k)} & \cdots & a_{nn}^{(k)} \end{pmatrix}}_{:=A^{(k)}} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \underbrace{\begin{pmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_k^{(k)} \\ \vdots \\ b_n^{(k)} \end{pmatrix}}_{:=b^{(k)}}.$$

Pasul $n - 1$

După $n - 1$ pași se obține astfel un nou sistem, echivalent cu cel inițial dar **superior triunghiular**

$$\underbrace{\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & \cdots & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & \cdots & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & a_{nn}^{(n)} \end{pmatrix}}_{:=A^{(n)}} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \underbrace{\begin{pmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_k^{(k)} \\ \vdots \\ b_n^{(n)} \end{pmatrix}}_{:=b^{(n)}}.$$

Algoritm

În concluzie, formulele după care se modifică sistemul de la pasul k în sistemul de la pasul $k + 1$ sunt

$$\begin{aligned} m_{ik} &= \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, & i &= k + 1, \dots, n, \\ a_{ij}^{(k+1)} &= a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)}, & i, j &= k + 1, \dots, n, \\ b_i^{(k+1)} &= b_i^{(k)} - m_{ik} b_k^{(k)}, & i &= k + 1, \dots, n. \end{aligned} \quad (3.5)$$

Numărul de operații

- Triangularizarea prin eliminare Gauss are costul dominant

$$\#op(\text{triangularizare}) = \frac{2}{3}n^3 + O(n^2).$$

- Substituția înapoi necesită $n^2 + O(n)$ operații.
- Prin urmare, costul total rămâne

$$\#op(\text{GEM}) = \frac{2}{3}n^3 + O(n^2).$$

Pentru compararea algoritmilor ne interesează în primul rând termenul dominant; formulele exacte depind de convenția folosită pentru numărarea împărțirilor, înmulțirilor și adunărilor.

GEM funcționează dacă $a_{kk}^{(k)} \neq 0$, $k = 1, 2, \dots, n - 1$.

Din păcate, faptul că diagonală inițială este nenulă nu garantează că la un pas ulterior k vom avea $a_{kk}^{(k)} \neq 0$.

De exemplu, considerând matricea

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 7 & 8 & 9 \end{pmatrix}$$

după primul pas găsim

$$A^{(2)} = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0 & -1 \\ 0 & -6 & -12 \end{pmatrix}.$$

Ce e de făcut?

Nu abandonăm metoda, ci identificăm cauza eșecului și construim o strategie care acoperă toate cazurile: pivotarea.

Regândim teoretic problema fără a recurge la “peticii” de moment.

Anticipăm

Poate putem ști de la bun început dacă pentru o matrice este potrivit sau nu să folosim GEM?

Într-adevăr sunt criterii care ne asigură că putem folosi GEM, de exemplu

- Matrice strict diagonal dominante pe linii sau pe coloane.
- Matrice simetrice pozitiv definite.

Însă toate aceste afirmații trebuie verificate și argumentate; simplele bănueli nu constituie justificări matematice.

În continuare urmărim să rescriem matricea $A \in \mathbb{R}^{n \times n}$ sub forma $A = LU$, unde L este triunghiulară inferioară, iar U este triunghiulară superioară.

Facem acest lucru deoarece după ce vom reuși, sistemul inițial va putea fi rescris sub forma a două sisteme triunghiulare, adică

$$Ax = b \quad \leftrightarrow \quad LUx = b \quad \leftrightarrow \quad \begin{cases} Ly = b \\ Ux = y \end{cases} \quad (3.6)$$

ce se vor rezolva pe rând.

GEM prin multiplicare de matrice

Operațiile de eliminare din prima coloană se pot scrie prin înmulțire la stânga cu

$$M_1 = I_n - m_1 e_1^T = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ -m_{21} & 1 & 0 & \cdots & 0 \\ -m_{31} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -m_{n1} & 0 & 0 & \cdots & 1 \end{pmatrix}, \quad m_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}.$$

Astfel $M_1 A^{(1)} = A^{(2)}$. La pasul următor folosim analog

$$M_2 = I_n - m_2 e_2^T, \quad M_2 A^{(2)} = A^{(3)},$$

și, repetând procedeul,

$$M_{n-1} \cdots M_2 M_1 A = A^{(n)} =: U,$$

unde U este triunghiulară superioară. Această scriere conduce direct la factorizarea LU .

Eliminarea gaussiană

Considerăm o matrice $A \in \mathbb{R}^{n \times n}$. Scopul este de a construi o secvență $A^{(k)} = (a_{ij}^{(k)})$ de matrici prin efectuarea de transformări liniare, astfel încât să ajungem la o matrice triunghiulară superioară $U = (u_{ij})$ după un număr finit de pași.

În ultima săptămână am văzut că dacă presupunem că pivoții $a_{11} \neq 0$, $a_{kk}^{(k)} = (M_{k-1} \cdots M_1 A)_{kk} \neq 0$ la orice pas $k = 2, \dots, n-1$, atunci

$$M_{n-1} M_{n-2} \cdots M_1 A = U, \quad (3.7)$$

cu U o matrice triunghiulară superioară, unde

$$M_k = \begin{pmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & -m_{k+1,k} & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -m_{n,k} & 0 & \cdots & 1 \end{pmatrix} = I_n - m_k e_k^T, \quad (3.8)$$

și

$$m_k = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ m_{k+1,k} \\ \vdots \\ m_{n,k} \end{pmatrix} \in \mathbb{R}^n, \quad e_k = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{R}^n, \quad m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, i = k+1, \dots, n. \quad (3.9)$$

Mai mult, deoarece $M_k^{-1} = I_n + m_k e_k^T$, putem reconstrui matricea inițială din matricea triunghiulară U . Într-adevăr, $A = M_1^{-1} M_2^{-1} \dots M_{n-1}^{-1} U$. Pentru $i < j$ avem $e_i^T m_j = 0$, deoarece primele j componente ale lui m_j sunt nule. Ca urmare, toți termenii încrucișați din produsul matricelor $I_n + m_i e_i^T$ dispar și obținem

$$A = \left(I_n + \sum_{i=1}^{n-1} m_i e_i^T \right) U = \underbrace{\begin{pmatrix} 1 & 0 & \dots & 0 & \dots & 0 \\ m_{21} & 1 & 0 & \dots & \dots & \vdots \\ m_{31} & m_{32} & 1 & 0 & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & 0 \\ m_{n1} & m_{n2} & \dots & m_{n,n-1} & & 1 \end{pmatrix}}_{:=L} U \quad (3.10)$$

Deci dacă nu întâlnim pivoți nuli ($a_{kk}^{(k)}$) atunci putem construi factorizarea LU a acelei matrice.

Dar cum știm dacă vom întâlni pivoți nuli fără a începe procesul de construcție al factorizării?

Existența și unicitatea factorizării LU

Teoremă 3.2.1. *Fie $A \in \mathbb{R}^{n \times n}$. Există factorizarea LU a matricei A cu $l_{ii} = 1$ pentru orice $i = 1, \dots, n$ și este unică dacă și numai dacă submatricele principale $A_i = A(1:i, 1:i)$ ale lui A de orice ordin $i = 1, \dots, n-1$ sunt nesingulare.*

Demonstrație. Vom demonstra mai întâi implicația “ $\Leftarrow$ ”. Presupunem că submatricele principale $A_i = A(1:i, 1:i)$ sunt nesingulare pentru $i = 1, \dots, n-1$. Construcția factorilor se face prin inducție și, în același timp, arată unicitatea lor.

Pentru $i = 1$ avem $A_1 = (1)(a_{11})$. Deoarece A_1 este nesingulară atunci când $n > 1$, rezultă $a_{11} \neq 0$. Presupunem acum că $A_{i-1} = L_{i-1} U_{i-1}$, unde L_{i-1} este triunghiulară inferioară unitară, iar U_{i-1} este triunghiulară superioară. Scriem bloc matricea principală următoare sub forma

$$A_i = \begin{pmatrix} A_{i-1} & c \\ d^T & a_{ii} \end{pmatrix} = \begin{pmatrix} L_{i-1} & 0 \\ \ell^T & 1 \end{pmatrix} \begin{pmatrix} U_{i-1} & u \\ 0^T & u_{ii} \end{pmatrix}.$$

Prin identificarea blocurilor trebuie să avem $L_{i-1} u = c$, $U_{i-1}^T \ell = d$ și $u_{ii} = a_{ii} - \ell^T u$. Deoarece A_{i-1} este nesingulară și $\det L_{i-1} = 1$, rezultă $\det U_{i-1} = \det A_{i-1} \neq 0$. Așadar cele două sisteme triunghiulare au soluții unice, $u = L_{i-1}^{-1} c$ și $\ell = U_{i-1}^{-T} d$, după care u_{ii} este determinat în mod unic. Pentru pasul final $i = n$ este suficientă nesingularitatea lui A_{n-1} ; matricea A însăși poate fi singulară, caz în care este posibil ca $u_{nn} = 0$.

Pentru implicația “ $\Rightarrow$ ”, presupunem că $A = LU$ este o factorizare cu L inferior triunghiulară unitară și că această factorizare este unică. Pentru fiecare i , blocul principal verifică $A_i = L_i U_i$, de unde $\det A_i = \prod_{j=1}^i u_{jj}$. Rămâne să arătăm că $u_{kk} \neq 0$ pentru orice $k < n$.

Presupunem, prin reducere la absurd, că $u_{kk} = 0$ pentru un anumit $k < n$. Pentru $t \in \mathbb{R}$ definim

$$S_t = I + te_{k+1}e_k^T, \quad S_t^{-1} = I - te_{k+1}e_k^T,$$

unde am folosit $(e_{k+1}e_k^T)^2 = 0$. Atunci $A = (LS_t)(S_t^{-1}U)$. Matricea LS_t rămâne inferior triunghiulară unitară. În același timp, $S_t^{-1}U$ se obține scăzând din linia $k+1$ a lui U de t ori linia k . Singura intrare care ar putea apărea sub diagonală în coloana k este $-tu_{kk} = 0$, deci $S_t^{-1}U$ rămâne superior triunghiulară. Pentru $t \neq 0$ obținem în general factori diferiți, ceea ce contrazice unicitatea. Prin urmare $u_{kk} \neq 0$ pentru $k = 1, \dots, n-1$, iar $\det A_i = \prod_{j=1}^i u_{jj} \neq 0$ pentru $i = 1, \dots, n-1$. $\square$

De ce nu se cere ca A să fie nesingulară?

Condiția privește numai minorii principali de ordine $1, \dots, n-1$. Ultimul pivot u_{nn} poate fi zero fără a mai exista un rând inferior pe care să-l putem modifica pentru a produce o altă factorizare. De aceea o matrice singulară poate avea totuși o factorizare LU unică; primul exemplu de mai jos ilustrează exact această situație.

Exemplu 3.2.2. Considerăm matricele

$$B = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}, \quad C = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad D = \begin{pmatrix} 0 & 1 \\ 0 & 2 \end{pmatrix}. \quad (3.11)$$

- B admite o unică factorizare LU .
- matricea neinvertibilă C nu admite factorizare LU .
- matricea neinvertibilă D admite o infinitate de factorizări de forma $D = L_\beta U_\beta$, cu

$$L_\beta = \begin{pmatrix} 1 & 0 \\ \beta & 1 \end{pmatrix}, \quad U_\beta = \begin{pmatrix} 0 & 1 \\ 0 & 2 - \beta \end{pmatrix} \quad \forall \beta \in \mathbb{R}.$$

Există un alt rezultat:

Teoremă 3.2.3. Dacă A este o matrice strict diagonal dominantă (pe linii sau pe coloane), atunci factorizarea LU fără pivotare există și este unică. În particular, dacă A este diagonal dominantă pe coloane, atunci $|l_{ij}| \leq 1 \forall i, j$.

Demonstrație. Demonstrăm mai întâi că o matrice strict diagonal dominantă pe linii este nesingulară. Dacă $Ax = 0$ și $x \neq 0$, alegem i astfel încât $|x_i| = \max_j |x_j| > 0$. Din ecuația i rezultă

$$|a_{ii}| |x_i| = \left| \sum_{j \neq i} a_{ij} x_j \right| \leq \sum_{j \neq i} |a_{ij}| |x_j| \leq \left(\sum_{j \neq i} |a_{ij}| \right) |x_i|,$$

contradicție cu diagonal-dominanța strictă. Deci matricea este nesingulară. Fiecare submatrice principală a unei matrice strict diagonal dominante pe linii este la rândul ei strict diagonal dominantă pe linii; toți minorii principali conducători de ordine $1, \dots, n-1$ sunt deci nenuli. Teorema de existență și unicitate a factorizării LU se aplică.

Dacă A este strict diagonal dominantă pe coloane, atunci A^T este strict diagonal dominantă pe linii. Submatricele principale ale lui A^T au aceiași determinanți ca cele ale lui A , deci concluzia privind existența și unicitatea urmează din același criteriu.

Rămâne să justificăm estimarea multiplicatorilor. Vom arăta prin inducție că submatricea activă după fiecare pas al eliminării păstrează dominanța strictă pe coloane. Presupunem că, la pasul

k , matricea activă $A^{(k)} = (a_{ij}^{(k)})_{i,j=k}^n$ este strict diagonal dominantă pe coloane și definim, pentru $i > k$, multiplicatorii $m_i = a_{ik}^{(k)} / a_{kk}^{(k)}$. Pentru $i, j > k$ avem

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_i a_{kj}^{(k)}.$$

Fixăm $j > k$. Din dominanța coloanei j rezultă

$$|a_{jj}^{(k)}| > |a_{kj}^{(k)}| + \sum_{\substack{i>k \\ i \neq j}} |a_{ij}^{(k)}|,$$

iar din dominanța coloanei k ,

$$|a_{kk}^{(k)}| > |a_{jk}^{(k)}| + \sum_{\substack{i>k \\ i \neq j}} |a_{ik}^{(k)}|.$$

Prin urmare

$$\begin{aligned} |a_{jj}^{(k+1)}| &= |a_{jj}^{(k)} - m_j a_{kj}^{(k)}| \\ &\geq |a_{jj}^{(k)}| - \frac{|a_{jk}^{(k)}|}{|a_{kk}^{(k)}|} |a_{kj}^{(k)}| \\ &> \sum_{\substack{i>k \\ i \neq j}} |a_{ij}^{(k)}| + \frac{|a_{kj}^{(k)}|}{|a_{kk}^{(k)}|} \sum_{\substack{i>k \\ i \neq j}} |a_{ik}^{(k)}| \\ &\geq \sum_{\substack{i>k \\ i \neq j}} \left| a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)} \right| = \sum_{\substack{i>k \\ i \neq j}} |a_{ij}^{(k+1)}|. \end{aligned}$$

Așadar complementul Schur este din nou strict diagonal dominant pe coloane. Inducția pornește de la $A^{(1)} = A$.

În particular, în coloana pivotului avem $|a_{ik}^{(k)}| < |a_{kk}^{(k)}|$ pentru orice $i > k$. Cum $l_{ik} = m_i$, rezultă

$$|l_{ik}| = \frac{|a_{ik}^{(k)}|}{|a_{kk}^{(k)}|} < 1,$$

și deci, în forma mai slabă enunțată, $|l_{ij}| \leq 1$ pentru toate elementele subdiagonale ale lui L . $\square$

Completare didactică: eliminarea gaussiană ca factorizare

Eliminarea gaussiană nu trebuie privită ca o succesiune de trucuri asupra liniilor. Fiecare pas este o multiplicare la stânga cu o matrice elementară, iar toți multiplicatorii pot fi stocați într-o matrice triunghiulară inferior L . La final obținem, în absența pivotării, factorizarea $A = LU$. Această interpretare separă două faze: o fază costisitoare, în care factorizăm A , și o fază ieftină, în care rezolvăm $Ly = b$ și apoi $Ux = y$.

Exemplu complet

Fie

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{pmatrix}.$$

Primul pivot este 2. Multiplicatorii sunt $m_{21} = 2$ și $m_{31} = -1$, iar după eliminarea primei coloane rămân liniile $(0, -8, -2)$ și $(0, 8, 3)$. Al doilea multiplicator este $m_{32} = -1$, ceea ce conduce la

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{pmatrix}, \quad A = LU.$$

Dacă alegem $x_* = (1, 2, 3)^T$, atunci $b = Ax_* = (7, -8, 18)^T$. Rezolvând succesiv $Ly = b$ și $Ux = y$ recuperăm x_* .

De ce păstrăm factorizarea?

Dacă avem aceeași matrice A și mai mulți termeni liberi $b^{(1)}, \dots, b^{(p)}$, factorizarea se calculează o singură dată. Pentru fiecare termen liber nou efectuăm doar două substituții de cost $O(n^2)$. Acest scenariu apare în simulări parametrice, metode Newton, probleme de control și discretizări dependente de timp.

Costul dominant al eliminării gaussiene pentru o matrice densă este aproximativ $\frac{2}{3}n^3$ operații aritmetice de tip înmulțire/adunare, în timp ce cele două rezolvări triunghiulare au cost de ordin n^2 . Diferența devine enormă pentru multe coloane de termeni liberi.

Laborator: o implementare LU elementară

Pași obligatorii de lucru. 1. Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Implementați eliminarea gaussiană fără pivotare și salvați multiplicatorii sub diagonală, în aceeași matrice în care se formează U . Nu construiți L la fiecare pas. La final extrageți L și U numai pentru verificare. Testați relația $\text{norm}(A-L*U)$ și rezolvarea unui sistem cu soluție cunoscută.

Apoi modificați programul astfel încât să accepte o matrice B cu mai mulți termeni liberi, rezolvând $AX = B$. Cronometrați separat timpul de factorizare și timpul rezolvărilor triunghiulare.

Exerciții și verificarea înțelegerii

1. Calculați manual factorizarea LU pentru $A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 5 & 3 \\ 1 & 4 & 6 \end{pmatrix}$, dacă aceasta poate fi obținută fără pivotare.
2. Pornind de la matricile elementare de eliminare $E_1, \dots, E_{n-1}$, explicați de ce $U = E_{n-1} \cdots E_1 A$ și $L = (E_{n-1} \cdots E_1)^{-1}$.
3. Arătați că dacă toți minorii principali de ordin $1, \dots, n$ sunt nenule, eliminarea gaussiană poate fi efectuată fără schimbări de linii.
4. Pentru o matrice A deja factorizată, estimați costul rezolvării a p sisteme $Ax = b^{(j)}$. Comparați cu refactorizarea lui A pentru fiecare j .
5. Explicați de ce formarea explicită a lui A^{-1} prin LU și apoi calculul $A^{-1}b$ efectuează mai multă muncă decât rezolvarea directă a celor două sisteme triunghiulare.

Studiu practic: discretizarea ecuației Poisson

Etapele proiectului. 1. Modelare: definiți toate mărimile și termenii specifici aplicației înainte de a-i folosi și formulați problema matematică. **2. Derivare:** justificați metoda și executați manual un exemplu mic. **3. Algoritm:** prezentați pseudocodul și analizați costul. **4. MATLAB:** scrieți o implementare proprie, modulară. **5. Experimente:** variați cel puțin un parametru și comparați cu o metodă de referință. **6. Validare:** folosiți numai criterii definite până în acel punct al cursului. **7. Explicație:** includeți o concluzie matematică, nu doar grafice sau tabele.

Porniți de la discretizarea unidimensională a problemei $-u'' = f$ pe $(0, 1)$ cu condiții Dirichlet. Obțineți sistemul tridiagonal corespunzător pentru mai multe valori ale pasului h . Rezolvați sistemul mai întâi folosind o rutină LU densă scrisă de voi, apoi folosind structura tridiagonală. Comparați timpul, memoria și eroarea față de o soluție exactă aleasă în prealabil.

Rezolvare orientativă**Factorizare LU completă**

Pentru $A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 5 & 3 \\ 1 & 4 & 6 \end{pmatrix}$, primul pivot este 1. Eliminăm cu multiplicatorii $l_{21} = 2$ și $l_{31} = 1$, obținând liniile $(0, 1, 1)$ și $(0, 2, 5)$. Al doilea pivot este 1, iar $l_{32} = 2$; ultima linie devine $(0, 0, 3)$. Astfel

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 2 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 3 \end{pmatrix}.$$

Înmulțirea directă confirmă $LU = A$. Dacă termenul liber este $b = A(1, 1, 1)^T = (4, 10, 11)^T$, rezolvăm mai întâi $Ly = b$: $y_1 = 4$, $y_2 = 2$, $y_3 = 3$. Apoi $Ux = y$ dă $x_3 = 1$, $x_2 = 1$, $x_1 = 1$.

Situație practică: aceeași matrice, mai mulți termeni liberi

Un motiv pentru a păstra o factorizare este rezolvarea mai multor sisteme $Ax = b^{(j)}$ cu aceeași matrice A și termeni liberi diferiți. De exemplu, într-un model mecanic liniar, necunoscutele pot fi deplasări, iar ecuațiile de echilibru pot avea forma $Kx = f$; matricea K , care transformă deplasările în forțe interne, este numită în mecanică *matrice de rigiditate*. Nu vom folosi aici teoria mecanică, ci doar faptul că K rămâne aceeași când se schimbă forța exterioară f . Astfel, matricea poate rămâne aceeași, iar vectorul termenilor liberi se schimbă. Dacă factorizarea costă aproximativ $O(n^3)$ și fiecare rezolvare triunghiulară $O(n^2)$, pentru 1000 de termeni liberi costul total este $O(n^3 + 1000n^2)$, nu $1000O(n^3)$. Reutilizarea factorizării este deseori diferența dintre un calcul fezabil și unul inutil de costisitor.

Exemplu complet: factorizare, două sisteme triunghiulare și verificare

Fie

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 5 \\ -2 \\ 9 \end{pmatrix}.$$

Eliminarea fără pivotare produce multiplicatorii $m_{21} = 2$ și $m_{31} = -1$. După primul pas, liniile a doua și a treia devin $(0, -8, -2)$, respectiv $(0, 8, 3)$. La al doilea pas, $m_{32} = -1$, iar ultima linie devine $(0, 0, 1)$. Prin urmare

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{pmatrix},$$

și se verifică direct $A = LU$.

Pentru $Ax = b$ nu calculăm A^{-1} . Rezolvăm mai întâi $Ly = b$. Din prima ecuație $y_1 = 5$, din a doua $2y_1 + y_2 = -2$, deci $y_2 = -12$, iar din a treia $-y_1 - y_2 + y_3 = 9$, deci $y_3 = 2$. Apoi rezolvăm $Ux = y$: ultima ecuație dă $x_3 = 2$, a doua dă $-8x_2 - 4 = -12$, deci $x_2 = 1$, iar prima dă $2x_1 + 1 + 2 = 5$, deci $x_1 = 1$. Soluția este $x = (1, 1, 2)^T$.

Acest calcul trebuie citit în trei niveluri: eliminarea construiește factorii, factorii permit rezolvarea rapidă, iar rezolvarea pentru mai mulți termeni liberi refolosește aceeași factorizare. Dacă avem $AX = B$ cu s coloane în B , costul mare $O(n^3)$ al factorizării este plătit o singură dată, iar fiecare coloană suplimentară necesită doar două substituții de ordin $O(n^2)$.

Problematizare: de ce nu calculăm inversa?

Formula $x = A^{-1}b$ este matematic impecabilă, dar nu descrie algoritmul recomandat. Construirea explicită a inversei înseamnă, în esență, rezolvarea a n sisteme cu termeni liberi e_i , urmată de o înmulțire matrice-vector. Pentru un singur sau câțiva termeni liberi, această muncă este inutilă și poate introduce erori suplimentare. În calcul numeric, „rezolvă sistemul” și „calculează inversa” sunt operații distincte.

Sarcină de programare: propria factorizare LU

Pași obligatorii de lucru. 1. Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. 2. Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. 3. Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. 4. Testați pe un caz cu rezultat cunoscut. 5. Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. 6. Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Implementați eliminarea gaussiană fără pivotare astfel încât funcția să întoarcă explicit L și U . Adăugați un contor pentru operațiile aritmetice și o verificare $\text{norm}(A-L*U)$. Apoi implementați o funcție separată $\text{luSolve}(L,U,b)$. Testați pe matrici pentru care soluția exactă este cunoscută. Într-un al doilea experiment, fixați A și rezolvați sistemul pentru 100 de vectori b_j ; comparați timpul dintre refolosirea factorilor și refactorizarea lui A la fiecare termen liber.

Exerciții suplimentare

1. Refaceți exemplul de mai sus cu aritmetică exactă și verificați fiecare relație $a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik}a_{kj}^{(k)}$.
2. Arătați că $\det A = \prod_{i=1}^n u_{ii}$ atunci când L este unitară. Folosiți formula pentru a calcula determinantul matricii din exemplu.
3. Explicați de ce factorizarea fără pivotare poate eșua pentru o matrice inversabilă. Construiți un exemplu 2×2 .
4. Presupuneți că A este fixă și $b = b(t)$ se schimbă la fiecare pas de timp. Descrieți ce parte a calculului se face o singură dată și ce parte se repetă.

5. Comparați experimental rezolvarea prin LU cu $\text{inv}(\mathbf{A}) * \mathbf{b}$ pentru matrici de dimensiuni crescătoare. Folosiți sisteme cu soluție fabricată și raportați timpul și cea mai mare diferență absolută dintre componentele soluției calculate și cele exacte.

3.3 Forma compactă a factorizării LU

Forma compactă a factorizării LU

O variantă a factorizării LU este factorizarea **Doolittle**. Aceasta este cunoscută și ca formă compactă a eliminării gaussiene.

Această denumire se datorează faptului că aceste abordări necesită mai puține rezultate intermediare decât metoda GEM standard pentru a genera factorizarea lui A .

Calcularea factorizării LU a lui A este echivalentă din punct de vedere formal cu rezolvarea următorului sistem neliniar de ecuații n^2

$$a_{ij} = \sum_{r=1}^{\min(i,j)} l_{ir}u_{rj}, \quad i, j = 1, \dots, n, \quad (3.12)$$

cele două matrice triunghiulare având împreună $n(n+1)$ intrări potențial nenule. Deoarece $A = LU$ nu fixează separat scalarea fiecărei perechi coloană-rând, impunem n condiții de normalizare, de exemplu $l_{ii} = 1$.

Dacă stabilim în mod arbitrar n coeficienți (l_{ii}) la 1, ajungem la metoda Doolittle care oferă o cale eficientă sistemului neliniar.

De fapt, presupunând că primele $k-1$ coloane din L și primele rânduri din U sunt disponibile și stabilind $l_{kk} = 1$ (metoda Doolittle), se obțin următoarele ecuații din

$$a_{kj} = \sum_{r=1}^{k-1} l_{kr}u_{rj} + \textcolor{red}{u}_{kj}, \quad j = k, \dots, n, \quad (3.13)$$

$$a_{ik} = \sum_{r=1}^{k-1} l_{ir}u_{rk} + \textcolor{red}{l}_{ik}u_{kk}, \quad i = k+1, \dots, n. \quad (3.14)$$

Rețineți că aceste ecuații pot fi rezolvate într-un mod secvențial în ceea ce privește variabilele roșii u_{kj} și l_{ik} .

Din metoda compactă Doolittle obținem astfel mai întâi al k -lea rând al lui U și apoi a k -a coloană a lui L , după cum urmează: pentru $k = 1, \dots, n$

$$\textcolor{red}{u}_{kj} = a_{kj} - \sum_{r=1}^{k-1} l_{kr}u_{rj}, \quad j = k, \dots, n, \quad (3.15)$$

$$\textcolor{red}{l}_{ik} = \frac{1}{u_{kk}} \left(a_{ik} - \sum_{r=1}^{k-1} l_{ir}u_{rk} \right), \quad i = k+1, \dots, n. \quad (3.16)$$

Completare didactică: forma compactă și calculul in-place

Într-o implementare matură nu este necesar să păstrăm separat toate matricile intermediare ale eliminării. Elementele lui U pot ocupa diagonala și partea superioară a memoriei alocate lui

A , iar multiplicatorii lui L pot fi stocați sub diagonală. Diagonala lui L este 1 și nu trebuie memorată. Această idee conduce la o factorizare *in-place*.

În varianta Doolittle, L are diagonala unitară. Presupunând cunoscute primele $k - 1$ coloane relevante ale lui L și primele $k - 1$ linii ale lui U , elementele liniei k din U se obțin din $u_{kj} = a_{kj} - \sum_{s=1}^{k-1} l_{ks}u_{sj}$ pentru $j \geq k$. Apoi, pentru $i > k$, calculăm $l_{ik} = (a_{ik} - \sum_{s=1}^{k-1} l_{is}u_{sk})/u_{kk}$. Observați ordinea: mai întâi pivotul u_{kk} și linia lui U , apoi coloana lui L .

Exemplu scurt

Pentru matricea din exemplul precedent, memoria finală poate conține

$$\begin{pmatrix} 2 & 1 & 1 \\ 2 & -8 & -2 \\ -1 & -1 & 1 \end{pmatrix}.$$

Citită cu convenția compactă, partea strict inferioară reprezintă multiplicatorii lui L , diagonala și partea superioară reprezintă U , iar diagonala lui L este implicit egală cu 1.

Algoritm in-place

Pași obligatorii de lucru. **1.** Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Scrieți o funcție `LUc = lu_compact(A)` care suprascrie o copie a lui A . Interziceți în cod construirea unei matrici elementare de eliminare. Scrieți separat două funcții care citesc factorii compactați și rezolvă $Ly = b$, respectiv $Ux = y$, fără a extrage explicit L și U .

Exerciții și verificarea înțelegerii

1. Derivați formulele Doolittle direct din egalitatea $a_{ij} = \sum_s l_{is}u_{sj}$.
2. Precizați care elemente trebuie deja cunoscute atunci când se calculează u_{kj} și l_{ik} .
3. Ce problemă apare dacă $u_{kk} = 0$? Dar dacă $|u_{kk}|$ este foarte mic față de elementele coloanei? Legați răspunsul de secțiunea despre pivotare.
4. Estimați economia de memorie dintre păstrarea separată a lui A , L , U și stocarea compactă.

Mini-proiect: profilarea accesului la memorie

Etapele proiectului. **1. Modelare:** definiți toate mărimile și termenii specifici aplicației înainte de a-i folosi și formulați problema matematică. **2. Derivare:** justificați metoda și executați manual un exemplu mic. **3. Algoritm:** prezentați pseudocodul și analizați costul. **4. MATLAB:** scrieți o implementare proprie, modulară. **5. Experimente:** variați cel puțin un parametru și comparați cu o metodă de referință. **6. Validare:** folosiți numai criterii definite până în acel punct al cursului. **7. Explicație:** includeți o concluzie matematică, nu doar grafice sau tabele.

Comparați două implementări ale factorizării: una care actualizează matricea prin bucle scalar-cu-scalar și una care folosește operații MATLAB pe blocuri/vectori. Pentru dimensiuni moderate, măsurați timpul și discutați de ce numărul aritmetic de operații nu este

singurul factor care determină performanța pe hardware modern.

Rezolvare orientativă

Citirea corectă a stocării compacte

Matricea compactă $C = \begin{pmatrix} 2 & 1 & 1 \\ 2 & -8 & -2 \\ -1 & -1 & 1 \end{pmatrix}$ nu reprezintă un nou operator. Convenția este:

U este partea triunghiulară superior a lui C , inclusiv diagonală, iar elementele strict sub diagonală sunt cele ale lui L ; diagonală lui L este implicit 1. Rezultă exact factorii $L =$

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix} \text{ și } U = \begin{pmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{pmatrix}.$$

Avantajul este că nu dublăm memoria. Dezavantajul este că programatorul trebuie să respecte riguros convenția: aceeași poziție de memorie are semnificații diferite în funcție de poziția față de diagonală.

Situație practică: rezolvare fără extragerea factorilor

La substituția înainte, când avem nevoie de l_{ij} pentru $j < i$, citim direct C_{ij} ; pentru diagonală folosim valoarea 1, nu C_{ii} . La substituția înapoi, citim $u_{ij} = C_{ij}$ pentru $j \geq i$. Astfel, factorii nu sunt niciodată copiați. În aplicații mari, eliminarea copiilor temporare poate economisi memorie și trafic de date, chiar dacă numărul de operații aritmetice rămâne același.

Exemplu și observații despre stocarea in-place

În algoritmul LU cu L unitară, diagonală lui L nu trebuie stocată: ea este cunoscută a priori și este formată numai din unități. În plus, multiplicatorii l_{ij} se află strict sub diagonală, în timp ce elementele lui U se află pe diagonală și deasupra ei. Cele două mulțimi de poziții nu se suprapun. De aceea aceeași matrice de n^2 numere poate conține simultan informația ambilor factori.

Pentru matricea exemplului anterior, forma compactă este

$$\hat{A} = \begin{pmatrix} 2 & 1 & 1 \\ 2 & -8 & -2 \\ -1 & -1 & 1 \end{pmatrix}.$$

Partea strict inferior triunghiulară reprezintă multiplicatorii din L , iar partea superior triunghiulară, inclusiv diagonală, este U . Citirea corectă este deci $L = I + \text{tril}(\hat{A}, -1)$ și $U = \text{triu}(\hat{A})$.

De ce contează stocarea?

Pentru o matrice densă de ordin $n = 20000$, $n^2 = 4 \cdot 10^8$ numere. În double precision aceasta înseamnă aproximativ 3.2 GB numai pentru o singură matrice. Dublarea inutilă a memoriei poate transforma un algoritm corect într-unul imposibil de rulat. Complexitatea de memorie este o parte a analizei algoritmului, nu doar un detaliu de programare.

Sarcină practică: factorizare in-place

Pași obligatorii de lucru. **1.** Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Modificați implementarea LU astfel încât să nu creeze matrici separate L și U în timpul eliminării. Suprascrieți elementele eliminate cu multiplicatorii corespunzători. Scrieți apoi două funcții auxiliare care extrag factorii numai pentru verificare. Măsurați memoria folosită de versiunea cu trei matrici (A, L, U) și de versiunea in-place pentru dimensiuni la care diferența devine vizibilă.

Întrebări și exerciții

1. Pentru forma compactă de mai sus, reconstruiți L și U fără a efectua din nou eliminarea.
2. Explicați de ce diagonală lui L nu poate fi folosită pentru a stoca altă informație dacă programul trebuie să permită și factorizări cu o convenție diferită de normalizare.
3. Cum se modifică interpretarea matricei compacte atunci când folosim pivotare parțială? Ce informație suplimentară trebuie stocată?
4. Estimați memoria, în MB, pentru o matrice 10000×10000 în precizie double. Comparați stocarea unei singure matrice cu stocarea separată a lui A, L, U .
5. Scrieți pseudocodul unei funcții care rezolvă $Ax = b$ folosind direct matricea compactă, fără a construi explicit L și U .

3.4 Factorizarea LDM^T

Factorizarea LDM^T

Este posibil să se conceapă și alte tipuri de factorizări ale lui A .

Mai exact, vom aborda unele variante în care factorizarea lui A este de forma

$$A = L D M^T, \quad (3.17)$$

unde L , M^T și D sunt matrici inferior triunghiulare, superior triunghiulare și, respectiv, diagonale.

După construirea acestei factorizări, rezolvarea sistemului se poate realiza rezolvând mai întâi sistemul inferior triunghiular $Ly = b$, apoi cel diagonal $Dz = y$ și în final sistemul superior triunghiular $M^T x = z$, cu un cost de aproximativ $2n^2$ operații (mai precis, $2n^2 - n$ în convenția în care diagonalele lui L și M sunt unitare).

În cazul simetric, obținem $M = L$, iar factorizarea LDL^T poate fi calculată cu jumătate din cost, după cum vom vedea când vom vorbi despre cazul matricelor simetrice. Factorizarea LDM^T se bucură de o proprietate analogă cu cea pentru factorizarea LU . În particular, se aplică următorul rezultat.

Teoremă 3.4.1. *Dacă toți minorii principali ai unei matrice $A \in \mathbb{R}^{n \times n}$ sunt nenuli, atunci*

există o matrice diagonală unică D , o matrice triunghiulară inferioară unitară unică¹ L și o matrice triunghiulară superioară unitară unică M^T , astfel încât $A = LDM^T$.

Demonstrație. Ipoteza spune în particular că toți minorii principali conducători de ordine $1, \dots, n$ sunt nenuli. Teorema anterioară furnizează factorizarea unică $A = LU$, cu L inferior triunghiulară unitară și U superior triunghiulară. Deoarece $\det A \neq 0$, toate elementele diagonale u_{ii} sunt nenule. Definim

$$D = \text{diag}(u_{11}, \dots, u_{nn}), \quad M^T = D^{-1}U.$$

Atunci M^T este triunghiulară superioară unitară și $A = LDM^T$, ceea ce dovedește existența.

Pentru unicitate, presupunem $A = L_1 D_1 M_1^T = L_2 D_2 M_2^T$. Produsele $D_i M_i^T$ sunt superior triunghiulare, deci am obținut două factorizări LU cu factor inferior unitar. Unicitatea factorizării LU dă $L_1 = L_2$ și $D_1 M_1^T = D_2 M_2^T$. Comparând diagonalele rezultă $D_1 = D_2$; cum diagonalele sunt nesingulare, urmează $M_1^T = M_2^T$. $\square$

Completare didactică: separarea scalării de transformările triunghiulare

Factorizarea $A = LDM^T$, cu L și M triunghiulare inferior cu diagonală unitară și D diagonală, separă în mod explicit pivoturile de multiplicatori. Din punct de vedere conceptual, L și M descriu eliminările, iar D păstrează scalările asociate pivoturilor. Pentru matrice simetrice, atunci când factorizarea se poate efectua fără pivotare, simetria conduce la $M = L$, deci la forma $A = LDL^T$.

Exemplu 2×2

Pentru $A = \begin{pmatrix} 2 & 4 \\ 6 & 13 \end{pmatrix}$ avem

$$L = \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix}, \quad D = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}, \quad M^T = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}.$$

Înmulțind factorii obținem $LDM^T = A$. În general, pentru $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ cu $a \neq 0$, primul pivot este $d_1 = a$, multiplicatorii sunt $l_{21} = c/a$ și $m_{21} = b/a$, iar al doilea element diagonal este $d_2 = d - cb/a$.

Legătura cu complementul Schur

Cantitatea $d - cb/a$ este complementul Schur al blocului scalar a . Aceeași structură apare în eliminarea pe blocuri, în sisteme saddle-point, optimizare constrânsă și metode de domeniu. Factorizările elementare studiate aici sunt astfel prototipuri pentru algoritmi mult mai mari.

¹Noi numim matrice triunghiulară unitară o matrice triunghiulară care are intrările diagonale egale cu 1.

Laborator / algoritm

Pași obligatorii de lucru. **1.** Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Implementați o factorizare LDL^T pentru matrice simetrice pentru care pivotarea nu este necesară. Verificați separat norma lui $A - LDL^T$ și comparați rezultatul cu factorizarea Cholesky atunci când A este pozitiv definită. Observați că LDL^T nu necesită extragerea rădăcinilor pătrate.

Exerciții și verificarea înțelegerii

1. Derivați factorizarea LDL^T a matricei $\begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}$.
2. Arătați că dacă $A = LDL^T$ cu L inversabilă, inerția formei pătratice este determinată de semnele elementelor lui D .
3. Explicați de ce pentru o matrice simetrică pozitiv definită elementele diagonale din D sunt pozitive.
4. Comparați pașii necesari pentru rezolvarea lui $Ax = b$ folosind LDL^T cu cei necesari pentru LU .

Mini-proiect: matrice simetrică, dar nu pozitiv definită

Etapele proiectului. **1. Modelare:** definiți toate mărimile și termenii specifici aplicației înainte de a-i folosi și formulați problema matematică. **2. Derivare:** justificați metoda și executați manual un exemplu mic. **3. Algoritm:** prezentați pseudocodul și analizați costul. **4. MATLAB:** scrieți o implementare proprie, modulară. **5. Experimente:** variați cel puțin un parametru și comparați cu o metodă de referință. **6. Validare:** folosiți numai criterii definite până în acel punct al cursului. **7. Explicație:** includeți o concluzie matematică, nu doar grafice sau tabele.

Construiți o familie de matrice simetrice cu valori proprii atât pozitive cât și negative. Investigați când o implementare LDL^T fără pivotare eșuează și comparați cu rutina MATLAB destinată matricelor generale. Scopul proiectului nu este reproducerea unei strategii sofisticate de pivotare simetrică, ci identificarea motivului pentru care simetria singură nu garantează siguranța algoritmului.

Rezolvare orientativă**Factorizarea LDL^T a unei matrice SPD**

Pentru $A = \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}$ căutăm $L = \begin{pmatrix} 1 & 0 \\ \ell & 1 \end{pmatrix}$ și $D = \text{diag}(d_1, d_2)$. Din elementul $(1, 1)$ obținem $d_1 = 4$. Din elementul $(2, 1)$ rezultă $\ell d_1 = 2$, deci $\ell = 1/2$. În final, elementul $(2, 2)$ dă $\ell^2 d_1 + d_2 = 3$, adică $1 + d_2 = 3$, deci $d_2 = 2$. Prin urmare $A = \begin{pmatrix} 1 & 0 \\ 1/2 & 1 \end{pmatrix} \text{diag}(4, 2) \begin{pmatrix} 1 & 1/2 \\ 0 & 1 \end{pmatrix}$.

Situație practică: detectarea semnului forme pătratică

Dacă o factorizare simetrică produce D cu elemente de semne diferite, forma pătratică nu poate fi pozitiv definită. Într-o problemă de optimizare, aceasta poate semnala că Hessiana are direcții de curbă negativă și punctul curent nu este minim strict. Astfel, factorul diagonal nu este doar un obiect de calcul, ci și un instrument de diagnostic structural.

Exemplu rezolvat: factorizarea LDM^T pentru o matrice nesimetrică

Considerăm

$$A = \begin{pmatrix} 2 & 1 \\ 3 & 4 \end{pmatrix}.$$

Căutăm $L = \begin{pmatrix} 1 & 0 \\ \ell & 1 \end{pmatrix}$, $D = \text{diag}(d_1, d_2)$ și $M^T = \begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix}$. Din produsul

$$LDM^T = \begin{pmatrix} d_1 & d_1 m \\ \ell d_1 & \ell d_1 m + d_2 \end{pmatrix}$$

identificăm succesiv $d_1 = 2$, $m = 1/2$, $\ell = 3/2$ și $d_2 = 4 - (3/2) \cdot 2 \cdot (1/2) = 5/2$. Așadar

$$A = \begin{pmatrix} 1 & 0 \\ 3/2 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 5/2 \end{pmatrix} \begin{pmatrix} 1 & 1/2 \\ 0 & 1 \end{pmatrix}.$$

Factorizarea separă transformările triunghiulare de scalările diagonale. În cazul simetric, factorii din stânga și dreapta se leagă între ei și ajungem la forma LDL^T .

Sarcină de implementare

Pași obligatorii de lucru. **1.** Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Pornind de la o implementare LU , scrieți o rutină care normalizează diagonală factorului superior și construiește LDM^T . Verificați numeric identitatea $A = LDM^T$ comparând elementele celor două matrice și raportând cea mai mare diferență în valoare absolută. Pentru matrici simetrice pentru care factorizarea există fără pivotare, verificați experimental că factorii triunghiulari unitari din stânga și dreapta coincid până la erorile de rotunjire.

Exerciții suplimentare

1. Calculați LDM^T pentru $A = \begin{pmatrix} 4 & 2 \\ 1 & 3 \end{pmatrix}$.
2. Arătați că $\det A = \prod_i d_i$ atunci când L și M sunt unitare.
3. Pentru o matrice simetrică, deduceți din unicitatea factorizării că $M = L$ în condițiile corespunzătoare de existență.
4. Explicați ce problemă apare dacă un element d_i este zero înainte de ultimul pas.
5. Comparați conceptual LDL^T cu Cholesky: când este preferabil să nu extragem

rădăcini pătrate?

3.5 Pivotare

Pivotare

După cum s-a subliniat anterior, procesul GEM se întrerupe imediat ce se calculează o intrare pivotală zero. Într-un astfel de caz, trebuie să se apeleze la așa-numita tehnică de pivotare, care constă în schimbarea rândurilor (sau a coloanelor) din sistem astfel încât să se obțină pivoti nenuli.

Strategia de pivotare adoptată până în prezent poate fi generalizată prin căutarea, la fiecare pas k al procedurii de eliminare, a unei intrări pivotante care nu este nulă, căutând în interiorul intrărilor din subcoloana $A^{(k)}(k : n, k)$. Din acest motiv, se numește pivotare parțială (pe rânduri).

Se poate observa că o valoare mare a lui $m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$, $i = k + 1, \dots, n$ (generată, de exemplu, de o valoare mică a pivotului $a_{kk}^{(k)}$) poate amplifica erorile de rotunjire care afectează intrările $a_{kj}^{(k)}$. Prin urmare, pentru a asigura o mai bună stabilitate, la pasul k alegem $r \in \{k, \dots, n\}$ astfel încât

$$|a_{rk}^{(k)}| = \max_{j=k, \dots, n} |a_{jk}^{(k)}|,$$

iar $a_{rk}^{(k)}$ devine pivotul după schimbarea liniilor k și r . În general, pivotarea parțială se efectuează la fiecare etapă, chiar dacă pivotul curent este deja nenul.

Alternativ, procesul de căutare ar fi putut fi extins la întreaga submatrice $A^{(k)}(k : n, k : n)$, finalizându-se cu o pivotare completă.

Observați, totuși, că pivotarea parțială necesită un cost suplimentar de ordin n^2 comparații, în timp ce pivotarea completă necesită aproximativ $n^3/3$ comparații. Acest cost este de același ordin cu cel aritmetic al eliminării, motiv pentru care în practică se preferă aproape întotdeauna pivotarea parțială.

Matrice de permutare

Schimbul dintre a i -a și j -a linie a unei matrice; acest lucru se poate face prin înmulțirea prealabilă a lui A cu matricea $P^{(i,j)}$ de elemente

$$p_{rs}^{(i,j)} = \begin{cases} 1 & \text{dacă } r = s = 1, \dots, i-1, i+1, \dots, j-1, j+1, \dots, n, \\ 1 & \text{dacă } r = j, s = i \text{ or } r = i, s = j, \\ 0, & \text{în caz contrar.} \end{cases} \quad (3.18)$$

Matricele de tipul $P^{(i,j)}$ se numesc matrice de permutare elementară.

Produsul matricelor de permutare elementară se numește matrice de permutare și efectuează schimburile de rânduri asociate fiecărei matrice de permutare elementară.

În practică, o matrice de permutare este o reordonare pe rânduri a matricei identitate.

Să analizăm modul în care pivotarea parțială afectează factorizarea LU indusă de GEM.

La prima etapă a GEM cu pivotare parțială, după ce se află intrarea a_{r1} de modul maxim din prima coloană, se construiește matricea elementară de permutare P_1 care schimbă prima linie cu a r -a linie (dacă $r = 1$, P_1 este matricea identitate).

În continuare, se generează prima matrice de transformare gaussiană M_1 și se stabilește

$$A^{(2)} = M_1 P_1 A^{(1)}. \quad (3.19)$$

O abordare similară se face acum pentru $A^{(2)}$, căutând o nouă matrice de permutare P_2 și o nouă matrice M_2 astfel încât

$$A^{(3)} = M_2 P_2 A^{(2)} = M_2 P_2 M_1 P_1 A^{(1)}. \quad (3.20)$$

Executând toate etapele de eliminare, matricea superior triunghiulară U rezultată este acum dată de

$$U = A(n) = \underbrace{M_{n-1} P_{n-1} \cdots M_2 P_2 M_1 P_1}_{:=M} A^{(1)}. \quad (3.21)$$

Rețineți că $m_{ik}^{(k)}$ utilizat în construcția lui M_k este acum $m_{ik}^{(k)} = \frac{b_{ik}^{(k)}}{b_{kk}^{(k)}}$, unde $b_{ik}^{(k)}$ sunt intrările matricei $P_k A^{(k)}$.

Obținem că $U = M A$ și, astfel, $U = (M P^{-1}) P A$, unde $P = P_{n-1} \cdots P_1$. Afirmăm că $L = P M^{-1}$ este triunghiulară inferioară unitară.

Nu trebuie să fim îngrijorați de prezența inversului lui M , deoarece $M^{-1} = P_1^{-1} M_1^{-1} \cdots P_{n-1}^{-1} M_{n-1}^{-1}$ și $P_i^{-1} = P_i^T$, în timp ce $M_i^{-1} = 2I_n - M_i = I_n + m_i e_i^T$.

Prin urmare, avem

$$\begin{aligned} L &= P_{n-1} \cdots P_2 P_1 P_1^{-1} (I_n + m_1 e_1^T) P_2^{-1} (I_n + m_2 e_2^T) \cdots P_{n-1}^{-1} (I_n + m_{n-1} e_{n-1}^T) \\ &= P_{n-1} \cdots P_2 (I_n + m_1 e_1^T) P_2^{-1} (I_n + m_2 e_2^T) \cdots P_{n-1}^{-1} (I_n + m_{n-1} e_{n-1}^T). \end{aligned}$$

Să discutăm acum

$$\begin{aligned} &P_{n-1} \cdots P_2 (I_n + m_1 e_1^T) \\ &= (P_{n-1} \cdots P_2 + P_{n-1} \cdots P_2 m_1 e_1^T P_2^T \cdots P_{n-1}^T P_{n-1} \cdots P_2) \\ &= (P_{n-1} \cdots P_2 + P_{n-1} \cdots P_2 m_1 e_1^T (P_{n-1} \cdots P_2)^T P_{n-1} \cdots P_2) \\ &= (P_{n-1} \cdots P_2 + P_{n-1} \cdots P_2 m_1 (P_{n-1} \cdots P_2 e_1)^T P_{n-1} \cdots P_2) \\ &= [I_n + P_{n-1} \cdots P_2 m_1 (P_{n-1} \cdots P_2 e_1)^T] P_{n-1} \cdots P_2. \end{aligned} \quad (3.22)$$

Vom avea

$$\begin{aligned} &P_{n-1} \cdots P_2 (I_n + m_1 e_1^T) \\ &= [I_n + \underbrace{P_{n-1} \cdots P_2 m_1 (P_{n-1} \cdots P_2 e_1)^T}_{:=\tilde{m}_1}] P_{n-1} \cdots P_2. \end{aligned} \quad (3.23)$$

Dar permutarea $P_{n-1} \cdots P_2$ permută doar intrările de la 2 la n dintr-un vector; intrările 1 rămân neatinsse. Aceasta înseamnă că prima intrare a lui $\tilde{m}_1$ este încă zero și e_1 rămâne neschimbat sub această permutare, adică $P_{n-1} \cdots P_2 e_1 = e_1$.

Astfel, avem de fapt

$$\begin{aligned} &P_{n-1} \cdots P_2 (I_n + m_1 e_1^T) \\ &= \underbrace{[I_n + \tilde{m}_1 e_1^T]}_{\text{inferior triunghiulară}} P_{n-1} \cdots P_2 \end{aligned} \quad (3.24)$$

și

$$L = \underbrace{[I_n + \widetilde{m}_1 e_1^T]}_{\text{inferior triunghiulară}} P_{n-1} \cdots P_2 P_2^{-1} (I_n + m_2 e_2^T) \cdots P_{n-1}^{-1} (I_n + m_{n-1} e_{n-1}^T).$$

Repetând argumentul avem că L este triunghiulară inferioară.

Deoarece $L = PM^{-1}$ este triunghiulară inferioară unitară, factorizarea LU se citește

$$PA = LU. \quad (3.25)$$

Odată ce L , U și P sunt disponibile, rezolvarea sistemului liniar inițial se rezumă la rezolvarea sistemelor triunghiulare $Ly = Pb$ și $Ux = y$.

Teoremă 3.5.1. *Fie $A \in \mathbb{R}^{n \times n}$ o matrice nesingulară. Atunci există o matrice de permutare P astfel încât $PA = LU$, unde L și U sunt, respectiv, o matrice triunghiulară inferioară și una triunghiulară superioară, obținute prin eliminarea gaussiană.*

Demonstrație. Demonstrația este deja făcută, cu excepția faptului că la orice pas avem

$$\max \text{abs}(A^{(k)}(k : n, k)) \neq 0. \quad (3.26)$$

Dacă ar fi posibil să avem

$$\max \text{abs}(A^{(k)}(k : n, k)) = 0, \quad (3.27)$$

atunci $\det A = 0$ ceea ce este evitat de ipoteză. $\square$

Dacă se realizează pivotarea completă, la primul pas al procesului, odată găsit elementul a_{qr} al celui mai mare modul din submatricea $A(1 : n, 1 : n)$, trebuie să schimbăm prima linie și prima coloană cu a q -a linie și a r -a coloană. Se generează astfel matricea $P_1 A^{(1)} Q_1$, unde P_1 și Q_1 sunt matrice de permutare pe rânduri și, respectiv, pe coloane.

În consecință, acțiunea matricei M_1 este acum astfel încât $A^{(2)} = M_1 P_1 A^{(1)} Q_1$. Repetând procesul, la ultima etapă, obținem

$$U = A^{(n)} = M_{n-1} P_{n-1} \cdots M_1 P_1 A^{(1)} Q_1 \cdots Q_{n-1}. \quad (3.28)$$

În cazul pivotării complete, factorizarea LU devine

$$PAQ = LU, \quad (3.29)$$

unde $Q = Q_1 \cdots Q_{n-1}$ este o matrice de permutare care ține cont de toate permutările care au fost operate. Prin construcție, matricea L este tot inferior triunghiulară, cu intrări de modul mai mici sau egale cu 1.

Calculul inversei unei matrice

Calculul explicit al inversei unei matrice poate fi efectuat folosind factorizarea LU după cum urmează.

Notând cu X inversa unei matrice nesingulare în $\mathbb{R}^{n \times n}$, vectorii coloană ai lui X sunt soluțiile sistemelor liniare $Ax_i = e_i$, pentru $i = 1, \dots, n$.

Presupunând că $PA = LU$, unde P este matricea de permutare cu pivotare parțială, trebuie să rezolvăm $2n$ sisteme triunghiulare de forma $Ly_i = Pe_i$, $Ux_i = y_i$, $i = 1, \dots, n$, adică o succesiune de sisteme liniare care au aceeași matrice de coeficienți, dar părți drepte diferite.

Aplicarea factorizării LU într-o problemă de deformare elastică 1D

Se consideră o bară elastică de lungime $L = 1$ m, cu capetele menținute fixe

$$u(0) = u(1) = 0.$$

Ecuția care descrie deformarea barei este de tip Poisson:

$$-k u''(x) = q(x), \quad 0 < x < 1,$$

unde u este deplasarea, k este un coeficient de elasticitate, iar $q(x)$ este forța care acționează asupra barei.

Aproximarea numerică a derivatei a doua

Pentru fiecare nod interior x_i , a doua derivată $u''(x_i)$ se poate aproxima prin diferențe finite centrale.

Pornim de la seriile Taylor în jurul nodului x_i :

$$u(x_i + h) = u(x_i) + hu'(x_i) + \frac{h^2}{2}u''(x_i) + \frac{h^3}{6}u'''(x_i) + O(h^4),$$

$$u(x_i - h) = u(x_i) - hu'(x_i) + \frac{h^2}{2}u''(x_i) - \frac{h^3}{6}u'''(x_i) + O(h^4).$$

Adunând cele două ecuații, derivata întâi se elimină:

$$u(x_i + h) + u(x_i - h) = 2u(x_i) + h^2u''(x_i) + O(h^4),$$

de unde rezultă formula de diferențe finite centrale:

$$u''(x_i) \approx \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1}))}{h^2}.$$

Această formulă stă la baza discretizării ecuației Poisson pentru fiecare nod interior.

Discretizare numerică a ecuației Poisson

Alegem $N = 4$ noduri interioare, deci $N + 1 = 5$ subintervale egale de lungime $h = L/(N + 1) = 0.2$ m, și notăm deplasările în nodurile $x_i = ih$ prin u_i , $i = 1, \dots, 4$. Nodurile de frontieră sunt $x_0 = 0$ și $x_5 = L$, cu $u_0 = u_5 = 0$.

Înlocuind aproximația derivatei a doua în ecuația Poisson:

$$-\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} = \frac{q(x_i)}{k}, \quad i = 1, \dots, 4,$$

sau echivalent:

$$2u_i - u_{i-1} - u_{i+1} = h^2 \frac{q(x_i)}{k}.$$

Astfel se generează un sistem liniar tridiagonal:

$$A \cdot \mathbf{u} = \mathbf{b},$$

unde

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} h^2 q(x_1)/k \\ h^2 q(x_2)/k \\ h^2 q(x_3)/k \\ h^2 q(x_4)/k \end{bmatrix}.$$

Factorizarea LU

Observație: valoarea deplasării în fiecare nod interior depinde doar de nodurile vecine, ceea ce face sistemul tridiagonal și foarte potrivit pentru factorizarea LU. Pentru rezolvarea eficientă, factorăm matricea A ca:

$$A = L \cdot U,$$

unde L este triunghiulară inferioară, iar U este triunghiulară superioară:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 & 0 \\ 0 & -\frac{2}{3} & 1 & 0 \\ 0 & 0 & -\frac{3}{4} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & \frac{3}{2} & -1 & 0 \\ 0 & 0 & \frac{4}{3} & -1 \\ 0 & 0 & 0 & \frac{5}{4} \end{bmatrix}.$$

Matrice cu structură de bandă

Metodele de discretizare pentru problemele cu valori la frontieră conduc adesea la rezolvarea sistemelor liniare cu matrici care au forme de bandă, bloc sau rare. Exploatarea structurii matricei permite o reducere drastică a costurilor de calcul ale factorizării și ale algoritmilor de substituție.

Vom aborda variante speciale ale factorizării GEM sau LU care sunt concepute în mod corespunzător pentru a trata matrici de acest tip.

Completare didactică: pivotarea ca mecanism de stabilizare

Existența unei factorizări algebrice nu este suficientă pentru un algoritm numeric robust. Dacă pivotul folosit la un pas este foarte mic, multiplicatorii de eliminare pot deveni foarte mari, iar erorile de rotunjire se pot amplifica. Pivotarea parțială alege, în coloana curentă, un element de modul maxim și schimbă liniile astfel încât acel element să devină pivot.

Exemplul clasic al unui pivot periculos

Considerăm

$$A = \begin{pmatrix} 10^{-16} & 1 \\ 1 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 1 + 10^{-16} \\ 2 \end{pmatrix},$$

care are soluția exactă $x = (1, 1)^T$. Fără pivotare, primul multiplicator este 10^{16} . În aritmetică finită, operația de eliminare combină numere de scăări foarte diferite și poate pierde cifre semnificative. Dacă schimbăm cele două linii, primul pivot devine 1, multiplicatorul are modul 10^{-16} , iar calculul este mult mai benign.

Cu pivotare parțială, factorizarea se scrie $PA = LU$, unde P este o matrice de permutare. Pentru rezolvarea sistemului trebuie deci folosit termenul liber permutat: mai întâi rezolvăm $Ly = Pb$, apoi $Ux = y$.

De ce pivotăm?

În aritmetica de calculator, împărțirea la un pivot foarte mic poate produce multiplicatori foarte mari și poate amplifica efectul rotunjirilor. Pivotarea încearcă să evite asemenea alegeri nefavorabile. Analiza cantitativă a sensibilității problemei și a erorilor algoritmului va fi făcută abia în capitolul următor.

Laborator: cu și fără pivotare

Pași obligatorii de lucru. **1.** Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Implementați două versiuni ale eliminării gaussiene: una fără pivotare și una cu pivotare parțială. Testați familia $A_\varepsilon = \begin{pmatrix} \varepsilon & 1 \\ 1 & 1 \end{pmatrix}$ pentru $\varepsilon = 10^{-2}, 10^{-4}, \dots, 10^{-16}$. Folosiți o soluție exactă prestabilită și reprezentați eroarea relativă în funcție de ε . Raportați și mărimea maximă a multiplicatorilor.

Exerciții și verificarea înțelegerii

1. Scrieți matricea de permutare care schimbă liniile 1 și 3 ale unei matrice 4×4 și verificați că $P^{-1} = P^T$.
2. Efectuați un pas de pivotare parțială pentru matricea $\begin{pmatrix} 0.01 & 2 & 1 \\ 3 & 1 & -1 \\ 2 & 4 & 1 \end{pmatrix}$.
3. Explicați de ce pivotarea parțială garantează multiplicatori de modul cel mult 1 la pasul curent.
4. Dați un exemplu în care pivotarea este necesară algebric deoarece pivotul curent este exact zero, deși matricea este inversabilă.
5. De ce o matrice de permutare nu modifică norma euclidiană a unui vector?

Mini-proiect: factor de creștere

Etapale proiectului. **1. Modelare:** definiți toate mărimile și termenii specifici aplicației înainte de a-i folosi și formulați problema matematică. **2. Derivare:** justificați metoda și executați manual un exemplu mic. **3. Algoritm:** prezentați pseudocodul și analizați costul. **4. MATLAB:** scrieți o implementare proprie, modulară. **5. Experimente:** variați cel puțin un parametru și comparați cu o metodă de referință. **6. Validare:** folosiți numai criterii definite până în acel punct al cursului. **7. Explicație:** includeți o concluzie matematică, nu doar grafice sau tabele.

Pentru matrice aleatoare și pentru câteva matrice construite deliberat, calculați raportul dintre cel mai mare element în modul apărut în factorul U și cel mai mare element în modul al matricei inițiale. Comparați eliminarea fără pivotare cu pivotarea parțială și urmăriți cum se modifică multiplicatorii și elementele intermediare.

Rezolvare orientativă

Permutarea unui pivot nul

Pentru $A = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}$ eliminarea fără pivotare nu poate începe, deși $\det A = -2 \neq 0$. Schimbăm liniile cu $P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ și obținem $PA = \begin{pmatrix} 2 & 3 \\ 0 & 1 \end{pmatrix}$. În acest caz $L = I$ și $U = PA$. Pentru sistemul $Ax = b$, termenul liber trebuie schimbat în aceeași ordine: $PAx = Pb$. Acest exemplu arată motivul algebric minimal al pivotării. Exemplul cu 10^{-16} arată motivul numeric: uneori pivotul nu este zero, dar este atât de mic încât folosirea lui este riscantă.

Situație practică: alegerea pivotului și unitățile de măsură

Dacă liniile provin din ecuații exprimate în unități foarte diferite, elementul cu modul maxim poate reflecta mai ales scala ecuației. O scalare preliminară poate face comparația pivoturilor mai semnificativă. Totuși, după scalare trebuie urmărită cu atenție problema transformată; efectul scalării asupra sensibilității va fi analizat sistematic în capitolul următor.

Exemplu rezolvat: pivotarea și efectul scării ecuațiilor

Considerăm matricea

$$A = \begin{pmatrix} 10^3 & 10^6 \\ 1 & 1 \end{pmatrix}.$$

Pivotarea parțială simplă compară $|10^3|$ și $|1|$ în prima coloană și păstrează prima linie. Totuși, raportat la cel mai mare element al fiecărei linii, primul candidat are mărimea relativă $10^3/10^6 = 10^{-3}$, iar al doilea are $1/1 = 1$. Acest exemplu motivează pivotarea parțială scalată: înainte de alegerea pivotului, comparăm $|a_{ik}|/s_i$, unde $s_i = \max_j |a_{ij}|$.

Nu trebuie însă confundate două idei. Pivotarea scalată poate evita o alegere nefavorabilă produsă numai de unitățile de măsură. Dacă două coloane sunt aproape dependente liniar, apar în continuare dificultăți numerice; acestea vor fi cuantificate în capitolul despre analiza erorii.

Ce trebuie stocat în practică?

Într-o implementare eficientă nu construim de obicei matricea de permutare P . Stocăm un vector de indici p care descrie schimburile de linii. Aplicarea lui P asupra unui vector b înseamnă doar reordonarea componentelor lui b conform lui p . Aceasta economisește memorie și evită operații cu o matrice care conține în principal zerouri.

Sarcină de implementare: vector de pivotare

Pași obligatorii de lucru. **1.** Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Extindeți factorizarea in-place astfel încât, la pasul k , să găsească indicele $p = \operatorname{argmax}_{i \geq k} |a_{ik}|$, să schimbe liniile și să actualizeze un vector de permutare. Funcția trebuie să întoarcă matricea compactă și vectorul p . Scrieți apoi `luPivotSolve`, care

aplică permutarea lui b și efectuează cele două substituții. Verificați relația $PA \approx LU$ fără a construi neapărat P : comparați $A(p, :)$ cu $L * U$.

Exerciții suplimentare

1. Pentru $A_\varepsilon = \begin{pmatrix} \varepsilon & 1 \\ 1 & 1 \end{pmatrix}$ și $\varepsilon = 10^{-12}$, efectuați primul pas cu și fără pivotare. Comparați multiplicatorii.
2. Arătați că o schimbare de linii este o transformare ortogonală: $P^T P = I$. Ce consecință are pentru norma euclidiană?
3. Explicați diferența dintre pivotare parțială, pivotare totală și pivotare parțială scalată.
4. De ce pivotarea totală necesită și memorarea permutărilor de coloane? Cum se recuperează ordinea necunoscutelor?
5. Construiți un experiment în care rescalarea unei singure ecuații schimbă pivotul ales de pivotarea parțială, dar nu schimbă soluția exactă a sistemului.

3.6 Factorizarea Cholesky

Matrice simetrice pozitiv definite: Factorizarea Cholesky

În continuare folosim abrevierea **SPD** (din engleză, *symmetric positive definite*) pentru o matrice **simetrică pozitiv definită**. Pentru factorizarea Cholesky vom folosi în text convenția $A = H^T H$, cu H superior triunghiular. Forma echivalentă $A = LL^T$, cu $L = H^T$ inferior triunghiular, este folosită în unele exemple și exerciții; cele două convenții descriu aceeași factorizare.

După cum s-a arătat deja, factorizarea LDM^T se simplifică considerabil atunci când A este simetrică, deoarece într-un astfel de caz $M = L$, obținându-se așa-numita factorizare LDL^T . Costul de calcul se înjumătățește, față de factorizarea LU, la aproximativ ($n^3/3$) flop-uri.

Factorizarea Cholesky

Teoremă 3.6.1 (Factorizarea Cholesky). *Fie $A \in \mathbb{R}^{n \times n}$ simetrică și pozitiv definită. Atunci există o unică matrice triunghiulară superioară H , cu elemente diagonale strict pozitive, astfel încât*

$$A = H^T H.$$

Demonstrație. Demonstrăm existența prin inducție după n . Pentru $n = 1$, avem $A = (a_{11})$ cu $a_{11} > 0$, iar $H = (\sqrt{a_{11}})$.

Presupunem rezultatul adevărat pentru matrici de ordin $n - 1$ și scriem

$$A = \begin{pmatrix} B & c \\ c^T & a \end{pmatrix},$$

unde $B \in \mathbb{R}^{(n-1) \times (n-1)}$. Orice submatrice principală a unei matrice SPD este SPD, deci B este SPD. Prin ipoteza de inducție există o unică matrice triunghiulară superioară R , cu diagonală pozitivă, pentru care $B = R^T R$.

Căutăm H sub forma

$$H = \begin{pmatrix} R & h \\ 0^T & \beta \end{pmatrix}, \quad \beta > 0.$$

Atunci

$$H^T H = \begin{pmatrix} R^T R & R^T h \\ h^T R & h^T h + \beta^2 \end{pmatrix}.$$

Identificarea cu A impune $R^T h = c$, deci $h = R^{-T} c$, și $\beta^2 = a - h^T h$. Trebuie să arătăm că această ultimă cantitate este strict pozitivă. Deoarece $B = R^T R$, avem $h^T h = c^T B^{-1} c$. Pentru vectorul nenul $z = (-B^{-1} c, 1)^T$ rezultă

$$0 < z^T A z = a - c^T B^{-1} c = a - h^T h.$$

Așadar $\beta = \sqrt{a - h^T h}$ este real și strict pozitiv, iar existența este demonstrată.

Pentru unicitate, presupunem $A = H_1^T H_1 = H_2^T H_2$, unde H_1, H_2 sunt superior triunghiulare cu diagonală pozitivă. Atunci

$$Q := H_1 H_2^{-1}$$

este triunghiulară superioară și verifică $Q^T Q = I$, deci este ortogonală. O matrice ortogonală triunghiulară superioară este diagonală, cu elemente diagonale egale cu ± 1 . Deoarece diagonalele lui H_1 și H_2 sunt pozitive, diagonală lui Q este pozitivă; prin urmare $Q = I$ și $H_1 = H_2$. $\square$

De ce apar radicali reali în algoritm?

La fiecare pas, expresia de sub radical este un complement Schur al unei submatrice principale SPD. Pozitiva definire nu este doar o ipoteză abstractă: ea garantează exact că pivotul Cholesky este strict pozitiv și că algoritmul poate continua fără pivotare.

Calculul elementelor factorului Cholesky

Teoremă 3.6.2 (Formulele recursive Cholesky). *Fie $A = (a_{ij})$ SPD și $A = H^T H$, unde $H = (h_{ij})$ este triunghiulară superioară cu $h_{jj} > 0$. Atunci, pentru $j = 1, \dots, n$,*

$$h_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} h_{kj}^2},$$

iar, după calculul lui h_{jj} , pentru $i = j + 1, \dots, n$,

$$h_{ji} = \frac{1}{h_{jj}} \left(a_{ji} - \sum_{k=1}^{j-1} h_{kj} h_{ki} \right).$$

În particular, $h_{11} = \sqrt{a_{11}}$.

Demonstrație. Din $A = H^T H$, pentru $j \leq i$ avem

$$a_{ji} = (H^T H)_{ji} = \sum_{k=1}^j h_{kj} h_{ki},$$

deoarece $h_{kj} = 0$ pentru $k > j$. Dacă $i = j$, obținem

$$a_{jj} = \sum_{k=1}^{j-1} h_{kj}^2 + h_{jj}^2.$$

Cum $h_{jj} > 0$, rezultă formula pentru h_{jj} . Dacă $i > j$, separăm ultimul termen al sumei,

$$a_{ji} = \sum_{k=1}^{j-1} h_{kj}h_{ki} + h_{jj}h_{ji},$$

și, deoarece $h_{jj} > 0$, izolarea lui h_{ji} dă formula a doua. Ordinea de calcul este astfel determinată: coloanele (sau, echivalent, liniile factorului superior) se obțin succesiv, iar fiecare formulă folosește numai elemente deja calculate. $\square$

Un exemplu Cholesky făcut cu pixul pe hârtie

Pentru

$$A = \begin{pmatrix} 4 & 2 & 2 \\ 2 & 5 & 1 \\ 2 & 1 & 3 \end{pmatrix}$$

avem $h_{11} = 2$, $h_{12} = 1$, $h_{13} = 1$. Apoi $h_{22} = \sqrt{5-1} = 2$ și $h_{23} = (1 - 1 \cdot 1)/2 = 0$. În final, $h_{33} = \sqrt{3 - 1^2 - 0^2} = \sqrt{2}$. Prin urmare

$$H = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 2 & 0 \\ 0 & 0 & \sqrt{2} \end{pmatrix}, \quad H^T H = A.$$

Studentul trebuie să poată explica, pentru fiecare dintre cele șase numere calculate, din ce egalitate $a_{ij} = (H^T H)_{ij}$ provine formula folosită.

Când folosim Cholesky?

Factorizarea Cholesky este construită pentru matrici simetrice pozitiv definite. Dacă matricea unui sistem are deja această structură, metoda o exploatează direct și necesită mai puține operații și mai puțină memorie decât o factorizare generală.

Principiu de alegere a metodei

Nu transformăm artificial orice sistem într-unul potrivit pentru Cholesky. În acest capitol alegem metoda după structura deja disponibilă a matricei: LU cu pivotare pentru cazul general, Cholesky pentru cazul SPD și algoritmi specializați pentru matrici de bandă. Motivele numerice mai fine pentru această alegere vor putea fi formulate după introducerea normelor și a analizei erorii.

Factorizarea Cholesky va fi folosită și ulterior, în probleme care conduc la matrici SPD.

Dicționar între convențiile Cholesky

În corpul capitolului am folosit convenția $A = H^T H$, cu H superior triunghiular. În multe implementări, inclusiv în exercițiile de laborator, este mai comodă convenția $A = LL^T$, cu $L = H^T$ inferior triunghiular. Formulele sunt echivalente prin $\ell_{ij} = h_{ji}$ pentru $i \geq j$:

$$\ell_{jj} = \sqrt{a_{jj} - \sum_{k<j} \ell_{jk}^2}, \quad \ell_{ij} = \frac{1}{\ell_{jj}} \left(a_{ij} - \sum_{k<j} \ell_{ik} \ell_{jk} \right), \quad i > j.$$

În laborator vom folosi consecvent forma inferioară $A = LL^T$.

Completare didactică: de ce Cholesky este metoda naturală pentru SPD

Dacă A este simetrică pozitiv definită, informația structurală trebuie folosită. Factorizarea Cholesky scrie $A = LL^T$, cu L triunghiulară inferior și diagonală strict pozitivă. Față de o factorizare LU generală, se stochează practic un singur factor și costul aritmetic se reduce aproximativ la jumătate: termenul dominant este de ordin $\frac{1}{3}n^3$.

Pozitiva definire garantează că expresiile aflate sub radical în calculul elementelor diagonale sunt pozitive. Astfel, proprietatea matematică a matricei este exact ceea ce permite existența algoritmului fără pivotare.

Exemplu construit din factor

Considerăm

$$L = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

Atunci

$$A = LL^T = \begin{pmatrix} 4 & 2 & 2 \\ 2 & 5 & 1 \\ 2 & 1 & 2 \end{pmatrix}$$

este automat simetrică pozitiv definită. Dacă $x_* = (1, 2, 3)^T$, atunci $b = Ax_* = (14, 15, 10)^T$. Pentru rezolvare efectuăm $Ly = b$ și apoi $L^T x = y$. Acest exemplu este util și pentru testarea unui cod: fiindcă A a fost construită dintr-un factor cunoscut, avem o verificare independentă.

Implementarea factorizării Cholesky

Pași obligatorii de lucru. **1.** Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Implementați formulele element-cu-element, fără a apela `chol`. Testați mai întâi pe matrice de forma $A = RR^T + \alpha I$, cu $\alpha > 0$, unde R este aleatoare. Comparați `norm(A-L*L')` cu precizia mașinii. Apoi modificați A astfel încât să devină indefinită și observați locul în care algoritmul încearcă să extragă rădăcina pătrată dintr-o valoare nepozitivă.

Aplicație: ecuații normale

Dacă o matrice are forma $A^T A$ și A are rang coloană complet, atunci $A^T A$ este simetrică pozitiv definită și poate fi factorizată Cholesky. Aplicațiile acestei observații la probleme dreptunghiulare vor fi discutate mai târziu, după introducerea instrumentelor necesare pentru analiza erorii.

Exerciții și verificarea înțelegerii

1. Calculați factorizarea Cholesky a matricei $\begin{pmatrix} 4 & 2 \\ 2 & 5 \end{pmatrix}$.

2. Demonstrați direct că dacă $A = LL^T$ și L este inversabilă, atunci A este pozitiv definită.
3. Arătați că o matrice SPD este inversabilă și că elementele sale diagonale sunt pozitive.
4. Comparați numărul de elemente care trebuie stocate pentru o matrice simetrică densă cu cel necesar pentru o matrice generală.
5. Explicați de ce pivotarea parțială standard ar distruge, în general, simetria dacă ar fi aplicată naiv într-un algoritm Cholesky.

Mini-proiect: matrice Gram

Etapele proiectului. **1. Modelare:** definiți toate mărimile și termenii specifici aplicației înainte de a-i folosi și formulați problema matematică. **2. Derivare:** justificați metoda și executați manual un exemplu mic. **3. Algoritm:** prezentați pseudocodul și analizați costul. **4. MATLAB:** scrieți o implementare proprie, modulară. **5. Experimente:** variați cel puțin un parametru și comparați cu o metodă de referință. **6. Validare:** folosiți numai criterii definite până în acel punct al cursului. **7. Explicație:** includeți o concluzie matematică, nu doar grafice sau tabele.

Generați n vectori $v_1, \dots, v_n \in \mathbb{R}^m$ liniar independenți. Matricea $G = (v_i^T v_j)_{ij} = V^T V$, ale cărei elemente sunt produsele scalare ale vectorilor, se numește matrice Gram. Formați această matrice. Verificați pozitivitatea definită și calculați Cholesky. Apoi apropiați două coloane una de alta și urmăriți determinantul lui G și elementele diagonale ale factorului Cholesky.

Rezolvare orientativă

Cholesky 2×2

Pentru $A = \begin{pmatrix} 4 & 2 \\ 2 & 5 \end{pmatrix}$, scriem $L = \begin{pmatrix} \ell_{11} & 0 \\ \ell_{21} & \ell_{22} \end{pmatrix}$. Din $LL^T = A$ rezultă $\ell_{11}^2 = 4$, deci alegem diagonala pozitivă $\ell_{11} = 2$. Apoi $\ell_{21}\ell_{11} = 2$, deci $\ell_{21} = 1$. Ultima relație este $\ell_{21}^2 + \ell_{22}^2 = 5$, de unde $\ell_{22} = 2$. Astfel $L = \begin{pmatrix} 2 & 0 \\ 1 & 2 \end{pmatrix}$.

Dacă $b = (6, 7)^T$, rezolvarea $Ly = b$ dă $y = (3, 2)^T$, iar $L^T x = y$ dă $x = (1, 1)^T$. Nu apare niciun calcul al inversei.

Situație practică: matrice Gram aproape singulară

Dacă două coloane ale unei matrici V sunt aproape egale, matricea Gram $G = V^T V$ rămâne teoretic SPD cât timp coloanele sunt independente, dar cea mai mică valoare proprie devine foarte mică. Cholesky poate calcula în continuare factorii, însă problema este sensibilă. Acesta este un exemplu bun de separare între existența factorizării și calitatea numerică a soluției.

Exemplu complet: Cholesky și rezolvarea unui sistem SPD

Fie

$$A = \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}, \quad b = \begin{pmatrix} 6 \\ 5 \end{pmatrix}.$$

Matricea este simetrică, iar minorii principali sunt $4 > 0$ și $\det A = 8 > 0$, deci este pozitiv definită. Căutăm $A = LL^T$ cu $L = \begin{pmatrix} \ell_{11} & 0 \\ \ell_{21} & \ell_{22} \end{pmatrix}$. Din identificarea elementelor rezultă $\ell_{11} = 2$, $\ell_{21} = 1$ și $\ell_{22} = \sqrt{3-1} = \sqrt{2}$. Prin urmare

$$L = \begin{pmatrix} 2 & 0 \\ 1 & \sqrt{2} \end{pmatrix}.$$

Rezolvăm $Ly = b$: $y_1 = 3$, iar $y_2 = (5-3)/\sqrt{2} = \sqrt{2}$. Apoi $L^T x = y$: din a doua ecuație $\sqrt{2}x_2 = \sqrt{2}$, deci $x_2 = 1$, iar din prima $2x_1 + x_2 = 3$, deci $x_1 = 1$. Soluția este $x = (1, 1)^T$.

De ce este Cholesky aproximativ de două ori mai ieftină decât LU ?

Pentru o matrice simetrică nu avem nevoie să calculăm și să stocăm separat informația de deasupra și de sub diagonală: una este transpusul celeilalte. Numărul dominant de operații pentru Cholesky este aproximativ $n^3/3$, față de aproximativ $2n^3/3$ pentru eliminarea gaussiană densă. Avantajul este simultan algebric, numeric și de memorie.

Sarcină practică: detectarea pierderii pozitivității

Pași obligatorii de lucru. 1. Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Implementați Cholesky folosind formula diagonală $\ell_{jj} = \sqrt{a_{jj} - \sum_{k < j} \ell_{jk}^2}$. Dacă expresia de sub radical devine nepozitivă, opriți algoritmul și raportați indicele. Testați pe matrici $A = B^T B + \alpha I$ pentru $\alpha > 0$, apoi micșorați α . Comparați cu o matrice simetrică indefinită. Explicați de ce simpla simetrie nu este suficientă.

Exerciții suplimentare

1. Calculați factorul Cholesky pentru $A = \begin{pmatrix} 9 & 3 \\ 3 & 5 \end{pmatrix}$ și verificați produsul.
2. Demonstrați că dacă $A = LL^T$ cu L nesingulară, atunci $x^T A x = (L^T x)^T (L^T x) > 0$ pentru orice $x \neq 0$.
3. Arătați că matricea Gram $A^T A$ este pozitiv semidefinită și precizați când este pozitiv definită.
4. Pentru A cu rang coloană complet, demonstrați direct că $A^T A$ este SPD și explicați de ce Cholesky este aplicabilă acestei matrice.
5. Formulați o verificare element cu element a identității $A = LL^T$ pentru o factorizare calculată.

3.7 Matrice cu structură de bandă

Matrice cu structură de bandă

Definiție 3.7.1. Spunem că o matrice $A \in \mathbb{R}^{m \times n}$ are bandă inferioară p dacă $a_{ij} = 0$ când $i > j + p$ și banda superioară q dacă $a_{ij} = 0$ când $j > i + q$.

Matricele diagonale sunt matrici cu benzi pentru care $p = q = 0$, în timp ce matricele bidiagonale au $p = 1, q = 0$ sau $p = 0, q = 1$ iar dacă $p = 1$ și $q = 1$ atunci spunem că avem o matrice tridiagonală.

Rezultatul principal pentru matrici cu benzi este următorul.

Teoremă 3.7.2. Fie $A \in \mathbb{R}^{n \times n}$. Să presupunem că există o factorizare LU a lui A . Dacă A are lățimea de bandă superioară q și lățimea de bandă inferioară p , atunci L are lățimea de bandă inferioară p și U are lățimea de bandă superioară q .

Demonstrație. Folosim formulele lui Doolittle și inducția după pasul k . Presupunem că primele $k - 1$ rânduri din U și primele $k - 1$ coloane din L respectă benzile. Dacă $j > k + q$, atunci $a_{kj} = 0$, iar

$$u_{kj} = a_{kj} - \sum_{r=1}^{k-1} l_{kr} u_{rj} = 0,$$

pentru că $j > k + q > r + q$ implică $u_{rj} = 0$ pentru orice $r < k$. Analog, dacă $i > k + p$, atunci $a_{ik} = 0$ și

$$l_{ik} = \frac{1}{u_{kk}} \left(a_{ik} - \sum_{r=1}^{k-1} l_{ir} u_{rk} \right) = 0,$$

deoarece $i > k + p > r + p$ implică $l_{ir} = 0$. Inducția pornește la $k = 1$ și arată că eliminarea nu creează elemente nenule în afara benzilor prescrise. $\square$

În special, observați că aceeași zonă de memorie utilizată pentru A este suficientă pentru a stoca și factorizarea sa LU .

Să considerăm, într-adevăr, că o matrice A având lățimea de bandă superioară q și lățimea de bandă inferioară p este de obicei stocată într-o matrice $(p + q + 1) \times n$, pe care o vom nota cu B , presupunând că

$$b_{i-j+q+1,j} = a_{ij} \quad (3.30)$$

pentru toți indicii i, j care se încadrează în banda matricei, în rest fiind zero.

De exemplu, în cazul matricei tridiagonale

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix} \quad (3.31)$$

stocarea compactă se citește

$$B = \begin{pmatrix} 0 & -1 & -1 & -1 & -1 \\ 2 & 2 & 2 & 2 & 2 \\ -1 & -1 & -1 & -1 & 0 \end{pmatrix} \quad (3.32)$$

Același format poate fi utilizat pentru stocarea factorizării LU a lui A .

Este clar că acest format de stocare poate fi destul de incomod în cazul în care doar câteva benzi ale matricei sunt mari.

La limită, dacă doar o coloană și un rând ar fi pline, am avea $p = q = n$ și astfel B ar fi o matrice plină cu multe intrări zero.

În cele din urmă, observăm că inversa unei matrice cu benzi este în general plină (așa cum se întâmplă pentru matricea A considerată mai sus).

Matrice tridiagonale

Considerăm cazul particular al unui sistem liniar cu matrice tridiagonală nesară A dată de

$$\begin{pmatrix} a_1 & c_1 & & 0 \\ b_2 & a_2 & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ 0 & & b_n & a_n \end{pmatrix}. \quad (3.33)$$

În acest caz, matricile L și U din factorizarea LU a lui A sunt matrici bidiagonale de forma

$$\begin{pmatrix} 1 & 0 & & 0 \\ \beta_2 & 1 & \ddots & \\ & \ddots & \ddots & 0 \\ 0 & & \beta_n & 1 \end{pmatrix}, \quad \begin{pmatrix} \alpha_1 & c_1 & & \\ 0 & \alpha_2 & \ddots & 0 \\ & \ddots & \ddots & c_{n-1} \\ 0 & & 0 & \alpha_n \end{pmatrix} \quad (3.34)$$

Algoritmul Thomas

Coeficienții α_i, β_i pot fi calculați cu ușurință prin următoarele relații

$$\alpha_1 = a_1, \quad \beta_i = \frac{b_i}{\alpha_{i-1}}, \quad \alpha_i = a_i - \beta_i c_{i-1}, \quad i = 2, \dots, n. \quad (3.35)$$

Algoritmul Thomas poate fi extins și pentru a rezolva întregul sistem tridiagonal $Ax = f$. Acest lucru înseamnă rezolvarea a două sisteme bidiagonale $Ly = f$ și $Ux = y$, pentru care se aplică următoarele formule:

$$(Ly = f) : \quad y_1 = f_1, y_i = f_i - \beta_i y_{i-1}, \quad i = 2, \dots, n, \quad (3.36)$$

$$(Ux = y) : \quad x_n = \frac{y_n}{\alpha_n}, \quad x_i = \frac{y_i - c_i x_{i+1}}{\alpha_i}, \quad i = n-1, \dots, 1. \quad (3.37)$$

Sisteme pe blocuri

În această secțiune ne ocupăm de factorizarea LU a matricelor partiționate în blocuri, în care fiecare bloc poate avea o dimensiune diferită.

Obiectivul nostru este dublu: optimizarea ocupării spațiului de stocare prin exploatarea adecvată a structurii matricei și reducerea costului de calcul al soluției sistemului.

Factorizarea LU

Fie $A = \mathbb{R}^{n \times n}$ următoarea matrice partiționată în blocuri

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad (3.38)$$

unde $A_{11} \in \mathbb{R}^{r \times r}$ este o matrice nesară a cărei factorizare $L_{11}D_1R_{11}$ este cunoscută, în timp ce $A_{22} \in \mathbb{R}^{(n-r) \times (n-r)}$.

În acest caz este posibilă factorizarea A folosind doar factorizarea LU a blocului A_{11} . Într-adevăr, este adevărat că

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & I_{n-r} \end{pmatrix} \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ 0 & I_{n-r} \end{pmatrix}, \quad (3.39)$$

unde

$$\begin{aligned} L_{21} &= A_{21}R_{11}^{-1}D_1^{-1}, \\ R_{12} &= D_1^{-1}L_{11}^{-1}A_{12}, \\ D_2 &= A_{22} - L_{21}D_1R_{12}. \end{aligned} \quad (3.40)$$

Dacă este necesar, procedura de reducere poate fi repetată pe matricea D_2 , obținându-se astfel o versiune în bloc a factorizării LU .

Dacă A_{11} ar fi un scalar, abordarea de mai sus ar reduce cu unu dimensiunea factorizării unei matrice date.

Prin aplicarea iterativă a acestei metode se obține un mod alternativ de efectuare a eliminării Gauss.

Completare didactică: exploatarea structurii de bandă

O matrice de bandă are majoritatea elementelor exact nule, iar aceste zerouri nu trebuie nici stocate, nici procesate. Dacă semi-lățimile benzii sunt fixe când n crește, costul unei factorizări scade de la ordin cubic la ordin liniar în n . Acest exemplu ilustrează principiul general: *algoritmul trebuie ales în funcție de structură, nu doar în funcție de dimensiune*.

Matricele tridiagonale apar la discretizarea derivatelor de ordinul al doilea. Pentru problema $-u''(x) = f(x)$, formula diferențelor finite centrate produce, după înmulțirea cu h^2 , o matrice cu 2 pe diagonală și -1 pe diagonalele imediat vecine.

Exemplu: sistem tridiagonal

Pentru

$$T = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix},$$

se verifică imediat că $x = (1, 1, 1, 1)^T$ este soluție. Algoritmul Thomas este eliminarea gaussiană specializată pentru această structură; el modifică doar cele trei diagonale și are cost $O(n)$.

Dacă notăm cu a_i subdiagonala, d_i diagonală și c_i supradiagonala, la pasul i putem elimina a_i folosind multiplicatorul $m_i = a_i/\tilde{d}_{i-1}$, apoi actualizăm numai $\tilde{d}_i = d_i - m_i c_{i-1}$ și termenul liber $\tilde{b}_i = b_i - m_i \tilde{b}_{i-1}$. Substituția înapoi folosește aceeași structură locală.

Laborator: Thomas versus matrice densă

Pași obligatorii de lucru. 1. Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al

cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil. Scrieți un algoritm Thomas care primește doar cei trei vectori ai diagonalelor și vectorul b . Nu construiți matricea $n \times n$. Comparați timpul și memoria cu o versiune densă pentru $n = 10^3, 10^4, 10^5$ atunci când este posibil. Comparați apoi cu reprezentarea rară MATLAB, de tip `sparse`.

Exerciții și verificarea înțelegerii

1. Aplicați manual algoritmul Thomas sistemului de ordin 4 de mai sus.
2. Determinați numărul de valori scalare necesare pentru stocarea unei matrice tridiagonale de ordin n .
3. Arătați că matricea tridiagonală cu 2 pe diagonală și -1 pe diagonalele vecine este SPD.
4. Ce condiție trebuie verificată pentru ca pasul $m_i = a_i/\tilde{d}_{i-1}$ să fie definit? Discutați ce garantează pozitivă definire în exemplul Poisson.
5. Generalizați ideea de stocare la o matrice cu p subdiagonale și q supradiagonale nenule.

Studiu practic: temperatură într-o bară

Etapele proiectului. **1. Modelare:** definiți toate mărimile și termenii specifici aplicației înainte de a-i folosi și formulați problema matematică. **2. Derivare:** justificați metoda și executați manual un exemplu mic. **3. Algoritm:** prezentați pseudocodul și analizați costul. **4. MATLAB:** scrieți o implementare proprie, modulară. **5. Experimente:** variați cel puțin un parametru și comparați cu o metodă de referință. **6. Validare:** folosiți numai criterii definite până în acel punct al cursului. **7. Explicație:** includeți o concluzie matematică, nu doar grafice sau tabele.

Considerați o bară unidimensională cu temperaturi prescrise la capete și cu o sursă termică $f(x)$. Discretizați problema staționară pentru mai multe valori ale lui n , rezolvați sistemul prin Thomas și comparați cu o soluție analitică pentru un f simplu. Studiați separat eroarea de discretizare și eroarea algoritmului liniar.

Rezolvare orientativă

Thomas pentru sistemul de ordin 4

Pentru diagonalele $a_i = -1$, $d_i = 2$, $c_i = -1$ și $b = (1, 0, 0, 1)^T$, eliminăm succesiv. Primul multiplicator este $m_2 = -1/2$, deci noul pivot este $\tilde{d}_2 = 2 - (-1/2)(-1) = 3/2$, iar $\tilde{b}_2 = 1/2$. Apoi $m_3 = -2/3$, $\tilde{d}_3 = 4/3$, $b_3 = 1/3$. În final $m_4 = -3/4$, $d_4 = 5/4$, $b_4 = 5/4$. Substituția înapoi dă $x_4 = 1$, apoi $x_3 = 1$, $x_2 = 1$, $x_1 = 1$.

Am folosit numai diagonalele, nu matricea completă. Pentru n mare, aceasta este esența economiei.

Situație practică: discretizare versus rezolvare

Dacă soluția numerică a unei EDP are eroare de ordin h^2 , nu are sens să rezolvăm sistemul liniar până la o toleranță de ordin 10^{-14} pentru un h relativ mare. Toleranța algoritmului liniar trebuie pusă în contextul erorii de modelare/discretizare. Într-un calcul științific complet, "precizia" nu este determinată de un singur pas al lanțului de calcul.

Exemplu rezolvat: un sistem tridiagonal provenit din diferențe finite

Considerăm matricea

$$T = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Vectorul $x = (1, 1, 1, 1)^T$ este soluție, deoarece pe liniile interioare apare combinația $-1+2-1=0$, iar pe liniile de margine obținem 1. O astfel de matrice apare la discretizarea operatorului $-u''$ cu condiții la frontieră.

Dacă ignorăm structura și folosim o factorizare densă, alocăm $O(n^2)$ memorie și efectuăm $O(n^3)$ operații. Algoritmul Thomas pentru o matrice tridiagonală păstrează numai cele trei diagonale și efectuează un număr de operații proporțional cu n . Pentru probleme obținute din discretizări cu milioane de necunoscute, diferența dintre cele două perspective este decisivă.

Sarcină de implementare: algoritmul Thomas

Pași obligatorii de lucru. **1.** Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Reprezentați matricea numai prin trei vectori: subdiagonala a , diagonală d și supradiagonala c . Implementați eliminarea înainte direct pe acești vectori și apoi substituția înapoi. Testați familia tridiagonală cu diagonală 2 și diagonalele vecine -1 pentru $n = 10^2, 10^3, 10^4, 10^5$. Comparați timpul și memoria cu o reprezentare densă numai pentru dimensiunile la care aceasta este încă fezabilă.

Exerciții suplimentare

1. Executați manual algoritmul Thomas pentru sistemul 4×4 de mai sus.
2. Arătați că matricea tridiagonală cu 2 pe diagonală și -1 pe diagonalele vecine este SPD.
3. Dacă lățimea totală de bandă este fixă și independentă de n , estimați costul factorizării în funcție de n .
4. Explicați de ce stocarea zerourilor din afara benzii este o risipă și propuneți o structură de date.
5. Investigați numeric ce se întâmplă când unul dintre pivoții algoritmului Thomas devine foarte mic. Când ar fi necesară pivotarea?

Fișă de seminar și laborator: metode directe în ansamblu

Problema rezolvată 1: alegerea algoritmului după structură

Presupunem că trebuie rezolvate patru sisteme de aceeași dimensiune: (a) o matrice densă generală; (b) o matrice densă SPD; (c) o matrice tridiagonală SPD; (d) aceeași matrice densă generală pentru 500 de termeni liberi. În cazul (a), alegerea standard este LU cu pivotare.

În (b), Cholesky exploatează simetria și pozitivă definită și reduce costul și stocarea. În (c), folosirea unei factorizări dense ar ignora aproape toate zerourile; un algoritm tridiagonal are cost liniar. În (d), cheia este refolosirea factorilor: factorizăm o dată și efectuăm câte două substituții pentru fiecare termen liber.

Această problemă nu cere un calcul complicat; urmărește recunoașterea structurii înainte de aplicarea unui algoritm generic.

Problema rezolvată 2: determinantul fără dezvoltare Laplace

Dacă $PA = LU$, atunci $\det(P)\det(A) = \det(L)\det(U) = \prod_i u_{ii}$, deoarece L este unitară. Cum $\det(P) = \pm 1$, determinantul lui A se obține aproape gratuit după factorizare. Numărul de schimburi de linii determină semnul lui $\det(P)$. Astfel, o cantitate care prin dezvoltare Laplace ar fi prohibitivă poate fi calculată cu aceeași factorizare folosită pentru rezolvarea sistemului.

Laborator integrator 3: o rutină de rezolvare adaptivă

Pași obligatorii de lucru. 1. Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. 2. Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. 3. Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. 4. Testați pe un caz cu rezultat cunoscut. 5. Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. 6. Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Scrieți o funcție de nivel înalt care inspectează matricea și alege o strategie: dacă este triunghiulară, substituție; dacă este simetrică și Cholesky reușește, Cholesky; dacă este tridiagonală, Thomas; altfel, LU cu pivotare. Raportați metoda aleasă. Nu urmăriți să reproduceți toate optimizările unei biblioteci profesionale, ci să demonstrați logica deciziei și să testați fiecare ramură pe exemple controlate.

Set de probleme

1. Pentru o matrice densă $n \times n$, comparați ordinele de cost ale factorizării și substituției. Pentru câți termeni liberi începe să domine costul substituțiilor?
2. Arătați că dacă A este SPD, pivotii Cholesky sunt reali și pozitivi.
3. Construiți o matrice pentru care LU fără pivotare eșuează la primul pas, dar $PA = LU$ există.
4. Dați un exemplu singular cu factorizare LU unică și explicați de ce ultimul pivot poate fi zero.
5. Demonstrați formula $S_t^{-1} = I - te_{k+1}e_k^T$ folosită în demonstrația unicității.
6. Pentru o matrice pentadiagonală, estimați stocarea necesară prin vectori de diagonale.
7. Comparați determinantul obținut din LU cu $\det(A)$ pentru matrici apropiate de singularitate și observați cât de repede poate deveni determinantul foarte mic. Nu folosiți această observație ca măsură cantitativă a sensibilității; aceasta va fi definită în capitolul următor.
8. Într-o aplicație cu matrice fixă și termeni liberi care sosesc în timp real, proiectați o structură care păstrează factorii și rezolvă noile sisteme.

Proiect de capitol: bibliotecă minimală de metode directe

Etapele proiectului. **1. Modelare:** definiți toate mărimile și termenii specifici aplicației înainte de a-i folosi și formulați problema matematică. **2. Derivare:** justificați metoda și executați manual un exemplu mic. **3. Algoritm:** prezentați pseudocodul și analizați costul. **4. MATLAB:** scrieți o implementare proprie, modulară. **5. Experimente:** variați cel puțin un parametru și comparați cu o metodă de referință. **6. Validare:** folosiți numai criterii definite până în acel punct al cursului. **7. Explicație:** includeți o concluzie matematică, nu doar grafice sau tabele.

Realizați un pachet cu funcții pentru substituție înainte/înapoi, LU cu pivotare, Cholesky și sistem tridiagonal. Fiecare funcție trebuie să aibă teste unitare pe soluții fabricate și mesaje clare în caz de eșec. Comparați componentele soluției calculate cu soluția fabricată și cu rezultatul unei rutine MATLAB standard. Adăugați un benchmark pentru dimensiuni crescătoare și discutați timpul, memoria și ipotezele fiecărei metode.

Sarcini de aprofundare și sinteză a metodelor directe**Probleme de aprofundare: factorizare, pivotare și structură**

1. Pentru $A(c) = \begin{pmatrix} 1 & c & 0 \\ 2 & 4 & 1 \\ 3 & 5 & 1 \end{pmatrix}$ determinați valorile lui c pentru care eliminarea gaussiană fără pivotare produce un pivot nul la pasul al doilea sau al treilea. Decideți separat dacă matricea este singulară sau numai eliminarea fără schimbări de linii eșuează.
2. Demonstrați legătura dintre existența factorizării $A = LU$ cu L unitară și nenulitatea minorilor principali conducători care apar în procesul de eliminare.
3. Construiți o matrice inversabilă 3×3 pentru care eliminarea fără pivotare eșuează la primul pas, dar o singură permutare de linii permite continuarea. Scrieți explicit matricea de permutare.
4. Pentru $A = LU$, cu L unitară, demonstrați $\det A = \prod_i u_{ii}$. Cum se modifică formula pentru $PA = LU$?
5. Arătați că dacă A este simetrică pozitiv definită, atunci factorizarea Cholesky poate fi construită pas cu pas și elementele diagonale ale factorului sunt strict pozitive.
6. Pentru o matrice tridiagonală deduceți recurențele algoritmului Thomas din eliminarea gaussiană și numărați operațiile necesare la fiecare pas. Explicați de ce numărul total crește liniar cu n .
7. Presupuneți că aceeași matrice A apare în sistemele $Ax = b_j$, $j = 1, \dots, m$. Comparați numărul de operații necesare dacă factorizați A la fiecare rezolvare cu situația în care factorii sunt calculați o singură dată.
8. Pentru o matrice de bandă cu semilățime p , numărați aproximativ elementele care trebuie stocate și comparați cu cele n^2 elemente ale unei matrice dense.
9. Explicați diferența algoritmică dintre un pivot exact nul și un pivot nenul, dar foarte mic. Construiți un exemplu 2×2 pentru fiecare situație și urmăriți eliminarea numerică.
10. Comparați LU și Cholesky pentru o matrice simetrică pozitiv definită: ce informație suplimentară folosește Cholesky și ce operații sunt evitate?

Laborator integrator: un solver care exploatează structura

Pași obligatorii de lucru. **1.** Refaceți mai întâi pe hârtie relațiile și cel puțin un pas numeric. **2.** Scrieți algoritmul în pseudocod și precizați datele de intrare, ieșire și criteriul de oprire, dacă este cazul. **3.** Implementați în MATLAB fără a apela direct rutina care rezolvă exact sarcina studiată, exceptând etapa de verificare. **4.** Testați pe un caz cu rezultat cunoscut. **5.** Comparați cu o rutină MATLAB de referință folosind numai mărimi și criterii definite până în acel punct al cursului. **6.** Formulați în câteva propoziții ce a făcut algoritmul și de ce rezultatul este credibil.

Obiectiv. Implementați metodele din capitol și comparați-le fără a folosi încă instrumentele de analiză introduse în capitolul următor.

1. Scrieți funcții separate pentru substituție înainte, substituție înapoi, LU cu pivotare parțială, Cholesky și Thomas. Nu folosiți `inv`.
2. Testați-le pe: o matrice densă generală, una simetrică pozitiv definită și una tridiagonală.
3. Pentru fiecare familie alegeți $x_* = \mathbf{1}$, formați $b = Ax_*$ și comparați componentele soluției calculate cu cele ale lui x_* .
4. Pentru dimensiuni crescătoare măsurați timpul de factorizare și timpul necesar rezolvării după ce factorii sunt deja disponibili.
5. Pentru problema cu mai mulți termeni liberi comparați refactorizarea repetată cu reutilizarea factorilor.
6. Pentru matricea tridiagonală comparați memoria necesară stocării dense cu memoria necesară celor trei diagonale.
7. Comparați rezultatele cu `A\b` numai ca verificare externă și raportați cea mai mare diferență absolută dintre componentele celor două soluții.

Proiect de capitol: discretizarea unei probleme la limită 1D

Etapele proiectului. **1. Modelare:** definiți toate mărimile și termenii specifici aplicației înainte de a-i folosi și formulați problema matematică. **2. Derivare:** justificați metoda și executați manual un exemplu mic. **3. Algoritm:** prezentați pseudocodul și analizați costul. **4. MATLAB:** scrieți o implementare proprie, modulară. **5. Experimente:** variați cel puțin un parametru și comparați cu o metodă de referință. **6. Validare:** folosiți numai criterii definite până în acel punct al cursului. **7. Explicație:** includeți o concluzie matematică, nu doar grafice sau tabele.

Nivel: mediu. Considerați problema $-u''(x) = f(x)$ pe $(0, 1)$, cu valorile soluției prescrise la capete, $u(0) = u(1) = 0$. Acest tip de condiție la frontieră se numește *condiție Dirichlet*. Formula diferențelor finite centrale și construcția sistemului tridiagonal au fost prezentate în exemplul de discretizare din acest capitol.

1. **Modelare.** Folosiți diferențe finite centrale și deduceți sistemul tridiagonal pentru necunoscutele interioare.
2. **Calcul manual.** Pentru trei necunoscute interioare scrieți complet sistemul și efectuați eliminarea de tip Thomas.
3. **Structură.** Arătați că matricea este simetrică și pozitiv definită.
4. **Algoritm.** Implementați rezolvarea prin Cholesky și Thomas.
5. **Experiment.** Folosiți o soluție exactă simplă pentru a genera termenul drept și comparați valorile calculate în nodurile grilei cu valorile exacte.
6. **Complexitate.** Măriți numărul de puncte și comparați timpul celor două implementări.

7. **Interpretare.** Explicați de ce structura tridiagonală trebuie folosită în alegerea algoritmului și nu tratată doar ca un detaliu de programare.

Test de capitol

Test – Metode directe pentru sisteme liniare

1. Calcul explicit și verificare critică. Pentru matricea

$$A = \begin{pmatrix} 4 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 2 \end{pmatrix},$$

un instrument AI afirmă că o factorizare Cholesky este

$$L = \begin{pmatrix} 2 & 0 & 0 \\ 1 & \sqrt{2} & 0 \\ 0 & 1/\sqrt{2} & 1 \end{pmatrix}.$$

Este afirmația corectă? Verificați prin calculul lui LL^T . Dacă este necesar, construiți factorizarea corectă pas cu pas. Înainte de calcul, explicați de ce factorizarea Cholesky este sau nu este permisă pentru această matrice.

2. Cod MATLAB. Se dorește rezolvarea unui sistem $Ax = b$ după o factorizare $PA = LU$ deja calculată:

```
y = L \ (P*b);  
x = U \ y;
```

Selectați afirmația/afirmațiile corecte și justificați:

- A) prima instrucțiune reprezintă o substituție înainte;
- B) a doua instrucțiune reprezintă o substituție înapoi;
- C) vectorul b nu trebuie permutat niciodată;
- D) aceeași factorizare poate fi reutilizată pentru mai mulți termeni liberi.