

Metode Numerice @ CS-UPB

Echipa MN CS - 2015:

Profesorii:

- Pantelimon George Popescu - Seria CA;
- Florin Pop - Seria CB, CC;

Asistenti:

PG Popescu, Clementin Cercel, Sorin Ciolofan, Bogdan Tiganoaia, Diana Udrea, Alexandru Tifrea, David Iancu, Bogdan Cristian Marchis, Radu Poenaru, Madalina Grosu, Andrei Vlad Postoaca, Radu Constantinescu, Roxana Istrate, Madalina Hristache, Adelina Vidovici;



Regulament MN 2015

- NotaParcurs = NotaPartial + NotaLab + NotaTeme
- NotaPartial (maxim 3 puncte) este nota obținuta la examenul parțial. Nu se repeta.
- NotaLab (maxim 1 punct).
- NotaTeme (maxim 2 puncte) este nota obținută din cele 4 teme (0.75 pe tema) de casă



Exemplu Problema

Bungee-jumping



$$F = ma$$

$$F/m = a$$

$$F = F_{\text{gravitationala}} + F_{\text{rezistenta_aer}}$$

$$F_{\text{gravitationala}} = mg$$

$$F_{\text{rezistenta_aer}} = -cv^2$$

$$\frac{dv}{dt} = g - \frac{c_d}{m} v^2$$



Regulament MN 2015

Reguli de notare

- NotaMN = NotaExamen + NotaParcurs + Bonus
- NotaExam (maxim 4 puncte) este nota obținuta la examenul final.
- NotaParcurs (maxim 6 puncte) este nota obținuta pentru activitatea din timpul semestrului
- Bonus (maxim 1 punct) este nota obținuta din bonusuri curs + laborator.



Regulament MN 2015

Reguli de promovare

- NotaParcurs >= 3
- NotaExam >= 2



Exemplu Problema

Bungee-jumping



Consideram urmatoarele date intitiale

$$m = 65,1 \text{ kg};$$

$$g = 9,81 \text{ m/s}^2;$$

$$c = 0,25 \text{ kg/m};$$

Ne propunem sa aflam viteza la un moment dat de timp



Exemplu Problema

Bungee-jumping Solutie NUMERICA



Acum cunoscand viteza la 2 secunde, putem afla viteza la 4 secunde,

$$v = 19.62 + \left[9.81 - \frac{0.25}{68.1} (19.62)^2 \right] \times 2 = 36.4137 \text{ m/s}$$

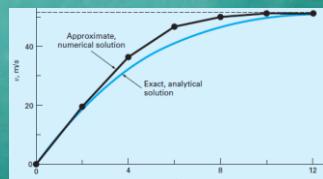


Exemplu Problema

Bungee-jumping Solutie NUMERICA



Reprezentand grafic, numeric, $v(t)$



The Best of the 20th Century: Editors Name Top 10 Algorithms

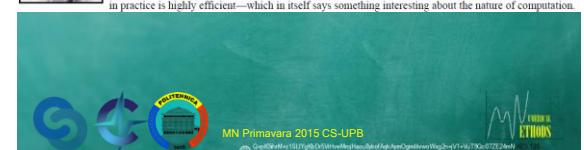
1946: John von Neumann, Stan Ulam, and Nick Metropolis, all at the Los Alamos Scientific Laboratory, cook up the Metropolis algorithm, also known as the **Monte Carlo method**.

1946: John von Neumann, Stan Ulam, and Nick Metropolis, all at the Los Alamos Scientific Laboratory, cook up the Metropolis algorithm, also known as the **Monte Carlo method**.

The Metropolis algorithm aims to obtain approximate solutions to numerical problems with unmanageably many degrees of freedom and to combinatorial problems of factorial size, by mimicking a random process. Given the digital computer's reputation for deterministic calculation, it's fitting that one of its earliest applications was the generation of random numbers.



1947: George Dantzig, at the RAND Corporation, creates the **simplex method for linear programming**. In terms of widespread application, Dantzig's algorithm is one of the most successful of all time. Linear programming dominates the world of industry, where economic survival depends on the ability to optimize within budgetary and other constraints. (Of course, the "real" problems of industry are often nonlinear; the use of linear programming is sometimes dictated by the computational budget.) The simplex method is an elegant way of arriving at optimal answers. Although theoretically susceptible to exponential delays, the algorithm in practice is highly efficient—which in itself says something interesting about the nature of computation.



The Best of the 20th Century: Editors Name Top 10 Algorithms

1951: Alston Householder of Oak Ridge National Laboratory formalizes the **decompositional approach to matrix computations**.

The decomposition of matrices into triangular, diagonal, orthogonal, and other special forms has turned out to be extremely useful. The decompositional approach has enabled software developers to produce flexible and efficient matrix packages. It also facilitates the analysis of rounding errors. One of the big bugbears of numerical linear algebra (In 1961, James Wilkinson of the National Physical Laboratory in London published a seminal paper in the *Journal of the ACM*, titled "Error Analysis of Direct Methods of Matrix Inversion," based on the LU decomposition of a matrix as a product of lower and upper triangular factors.)



1957: John Backus leads a team at IBM in developing the **Fortran optimizing compiler**.

The creation of Fortran may rank as the single most important event in the history of computer programming. Finally, scientists (and others) could tell the computer what they wanted it to do, without having to descend into the netherworld of machine code. Although modest by modern compiler standards—Fortran consisted of a mere 23,500 assembly-language instructions—the early compiler was nonetheless capable of surprisingly sophisticated computations. As Backus himself recalls in a recent history of Fortran I, II, and III, published in 1998 in the *IEEE Annals of the History of Computing*, the compiler "produced code of such efficiency that its output would startle the programmers who studied it."



Exemplu Problema

Bungee-jumping Solutie NUMERICA



samăt, obținând astfel următoarele date

t, s	$v, \text{m/s}$
0	0
2	19.6200
4	36.4137
6	46.2983
8	50.1802
10	51.3123
12	51.6008
∞	51.6938



The Best of the 20th Century: Editors Name Top 10 Algorithms

1959-61: J.G.F. Francis of Ferranti Ltd., London, finds a stable method for computing eigenvalues, known as the QR algorithm.
Eigenvalues are arguably the most important numbers associated with matrices—and they can be the trickiest to compute. It's relatively easy to transform a square matrix into a matrix that's "almost" upper triangular, meaning one with a single extra set of nonzero entries just below the main diagonal. But chopping away those final nonzeros, without launching an avalanche of zeros, is nontrivial. The QR algorithm is just the ticket. Based on the QR decomposition, which writes A as the product of an orthogonal matrix Q and an upper triangular matrix R , this approach iteratively changes $A_i = QR$ into $A_{i+1} = RQ$, with a few bells and whistles for accelerating convergence to upper triangular form. By the mid-1960s, the QR algorithm had turned once-formidable eigenvalue problems into routine calculations.

1962: Tony Hoare of Elliott Brothers, Ltd., London, presents Quicksort.

Putting N things in numerical or alphabetical order is mind-numbingly mundane. The intellectual challenge lies in devising ways of doing so quickly. Hoare's algorithm uses the age-old recursive strategy of divide and conquer to solve the problem: Pick one element as a "pivot," separate the rest into piles of "big" and "small" elements (as compared with the pivot), and then repeat this procedure on each pile. Although it's possible to get stuck doing all $N(N-1)/2$ comparisons (especially if you use as your pivot the first item on a list that's already sorted!), Quicksort runs on average with $O(N \log N)$ efficiency. Its elegant simplicity has made Quicksort the poster-child of computational complexity.



The Best of the 20th Century: Editors Name Top 10 Algorithms

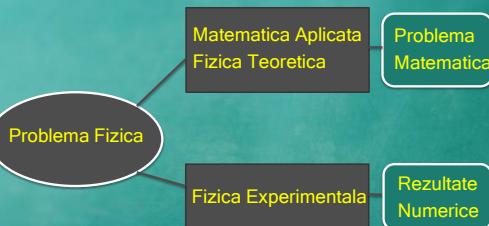
1987: Leslie Greengard and Vladimir Rokhlin of Yale University invent the fast multipole algorithm.

This algorithm overcomes one of the biggest headaches of N-body simulations: the fact that accurate calculations of the motions of N particles interacting via gravitational or electrostatic forces (think stars in a galaxy, or atoms in a protein) would seem to require $O(N^2)$ computations for each pair of particles. The fast multipole algorithm gets by with $O(N)$ computations. It does so by using multipole expansions (per charge or mass, dipole moment, quadrupole, and so forth) to approximate the effects of a distant group of particles on a local group. A hierarchical decomposition of space is used to define ever-larger groups as distances increase. One of the distinct advantages of the fast multipole algorithm is that it comes equipped with rigorous error estimates, a feature that many methods lack.

*January/February 2000, Computing in Science & Engineering, a joint publication of the American Institute of Physics and the IEEE Computer Society. Guest editors Jack Dongarra of the University of Tennessee and Oak Ridge National Laboratory and Francis Sullivan of the Center for Computing Sciences at the Institute for Defense Analyses put together a list they call the "Top Ten Algorithms of the Century."



Problematica



The Best of the 20th Century: Editors Name Top 10 Algorithms

1965: James Cooley of the IBM T.J. Watson Research Center and John Tukey of Princeton University, plus AT&T Bell Laboratories, invent the fast Fourier transform.

Early on, the most-reaching algorithm in applied mathematics—the FFT revolutionized signal processing. The underlying idea goes back to Gauss (who needed to calculate orbits of asteroids); but it was the Cooley-Tukey paper that made it clear how easily Fourier transforms can be computed. Like Quicksort, the FFT relies on a divide-and-conquer strategy to reduce an ostensibly $O(N^2)$ chore to an $O(N \log N)$ frolic. But unlike Quicksort, the implementation is (at first sight) more numberless and less straightforward. This in itself gave computer science an impetus to investigate the inherent complexity of computational problems and algorithms.

1977: Helmut Ferguson and Rodney Forcade of Brigham Young University advance an integer relation detection algorithm.
This problem is an old one: Given a bunch of real numbers, say y_1, y_2, \dots, y_n , are there integers a_1, a_2, \dots, a_n (not all 0) for which $a_1y_1 + a_2y_2 + \dots + a_ny_n = 0$? For $n = 2$, the venerable Euclidean algorithm does the job, computing terms in the continued-fraction expansion of y_1/y_2 . If y_1/y_2 is rational, the expansion terminates and, with proper unraveling, gives the "smallest" integers a_1 and a_2 . If the Euclidean algorithm doesn't terminate—or if you simply get tired of computing it—then the unraveling procedure at least provides lower bounds on the size of the smallest integer relation. Ferguson and Forcade's generalization, although much more difficult to implement (and to understand), is also more powerful. Their detection algorithm, for example, has been used to find the precise coefficients of the polynomials satisfied by the third and fourth bifurcation points, $B_3 = 3.544090$ and $B_4 = 3.564407$, of the logistic map. (The latter polynomial is of degree 120; its largest coefficient is 257^{10} .) It has also proved useful in simplifying calculations with Feynman diagrams in quantum field theory.

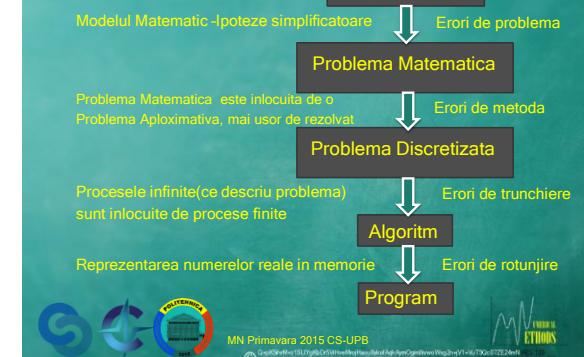


Continut curs MN

- Introducere.
- Despre matrici.
- Transformari.
- Sisteme de ecuatii lineare.
- Ecuatii neliniare.
- Valori proprii și vectori proprii.
- Interpolare polinomiala.
- Teoria Aproximarii.
- Integrale si derivare numerica.
- Integrarea ecuatiilor diferențiale ordinare. (ODE)



Problematica



Virgula mobila IEEE754

Virgula mobila

$$f1(x) = \pm 0.a_1a_2\dots a_t \times \beta^e$$

reprezentare normalizată

$$1 \leq a_i < \beta \quad \text{și} \quad 0 \leq a_i < \beta, \quad i=2:t$$

$$L \leq e \leq U$$

Sistemul de reprezentare în virgula mobilă $F(\beta, t, L, U)$ cuprinde:

- baza β
- precizia reprezentării t
- limitele (superioară și inferioară ale) exponentului L și U
- reprezentarea lui zero

Exemplu: $F(10, 1, 0, 1) = (\pm 0.a_1 \times 10^e) \cup \{0\}$ cu $a_1 \in \{1:9\}$ și $e \in \{0,1\}$, în total 37 de numere.



MN Primavara 2015 CS-UPB

(c) Facultatea de Inginerie Electrică și Calculatoare, Universitatea Politehnica București, 2015, www.cs.upb.ro

Virgula mobila IEEE754

Surse erori

Un număr real $x \in F$ se reprezintă exact, dacă suma se termină înainte de t termeni și exponentul este cuprins între limite. Altfel, numărul real x se *aproximează* prin-o valoare $f1(x) \in F$.

Aproximarea numărului real se face prin *rotunjire* sau prin *truncare*.

• Aproximare prin truncare ignoră cifrele numărului real din dreapta poziției t .

$$f1(x) = (0.a_1a_2\dots a_t)\beta^e = (a_1\beta^{-1} + a_2\beta^{-2} + \dots + a_t\beta^{-t})\beta^e$$

• Aproximarea prin rotunjire consideră:

$$f1(x) = (0.a_1a_2\dots a_t+1)\beta^e \quad \text{dacă } a_{t+1} \geq \beta/2$$

$$f1(x) = (0.a_1a_2\dots a_t)\beta^e \quad \text{dacă } a_{t+1} < \beta/2$$

O *dejunsire superioară* apare dacă $e \in U$.

Ea declanșază o eroare la execuție, care conduce la interruperea calculelor.

O *dejunsire inferioară* apare dacă $e \in L$.

ea duce la înlocuirea numărului prin zero.



MN Primavara 2015 CS-UPB

(c) Facultatea de Inginerie Electrică și Calculatoare, Universitatea Politehnica București, 2015, www.cs.upb.ro

Virgula mobila IEEE754

Surse erori

• *Eroarea absolută* la rotunjirea prin truncare:

$$\begin{aligned} e_x &= x - f1(x) = (a_1/\beta^1 + a_2/\beta^2 + \dots + a_t/\beta^t + a_{t+1}/\beta^{t+1} + \dots) \beta^e \\ &\quad - (a_1/\beta^1 + a_2/\beta^2 + \dots + a_t/\beta^t) \beta^e \\ e_x &= (a_{t+1}/\beta^1 + a_{t+2}/\beta^2 + \dots) \beta^{e-t} \end{aligned}$$

$$\begin{aligned} |e_x| &\leq (\beta-1)/\beta^1 + (\beta-1)/\beta^2 + \dots + |\beta^{e-t}| \\ &= (\beta-1)\beta^{e-t} / (1/\beta^1 + 1/\beta^2 + \dots) \leq (\beta-1)\beta^{e-t} / (\beta-1) = \beta^{e-t} \end{aligned}$$

$$|e_x| \leq \beta^{e-t}$$

Dacă se folosește **rotunjirea** atunci eroarea absolută este și mai mică:

$$|e_x| \leq 1/2 \cdot \beta^{e-t}$$



MN Primavara 2015 CS-UPB

(c) Facultatea de Inginerie Electrică și Calculatoare, Universitatea Politehnica București, 2015, www.cs.upb.ro

Virgula mobila IEEE754

Virgula mobila

$$2(\beta-1)\beta^{t-1}(U-L+1) \quad \text{valori distincte}$$

• a_i poate lua $\beta-1$ valori distincte,

• restul de $t-1$ cifre poate lua fiecare β valori diferite, deci β^{t-1} ,

• exponentul în $U-L+1$, și

• semnul două).

Cel mai mare număr reprezentabil Ω , (**realmax**) are forma:

$$\Omega = 0 \cdot (\beta-1) (\beta-1) \dots (\beta-1) \times \beta^U =$$

$$= [(\beta-1)/\beta^1 \cdot (\beta-1)/\beta^2 \dots + (\beta-1)/\beta^t] \times \beta^U$$

$$= (\beta-1)/\beta \cdot (1-\beta^{-1}) \cdot (1-\beta^{-2}) \times \beta^U = \beta^U \cdot (1-\beta^{-1})$$

$$\Omega = \beta^U \cdot (1-\beta^{-1})$$

• Cel mai mic număr pozitiv reprezentabil ω numit și **realmmin** este:

$$\omega = 0 \cdot 10 \dots 0 \times \beta^{-t} = \beta^{-t}/\beta = \beta^{t-1}$$

$$\omega = \beta^{t-1}$$



MN Primavara 2015 CS-UPB

(c) Facultatea de Inginerie Electrică și Calculatoare, Universitatea Politehnica București, 2015, www.cs.upb.ro

Virgula mobila IEEE754

Surse erori

• *Epsilon mașină* (notat **eps** în Matlab sau μ) reprezintă cel mai mic număr pozitiv cu proprietatea că:

$$f1(1+\mu) > 1$$

De exemplu în $F(10, 4, -3, 3)$ cu *rotunjire prin tăiere* (*truncare*):

$$f1(1+0.0009) = f1(1.0009) = 1$$

$$f1(1+0.0010) = f1(1.0010) = 1.001 > 1$$

ășadar $\mu_{\text{tr}} = 0.001 = 10^{-3} > \omega = 10^{-4}$.

• Dacă se folosește rotunjire, atunci:

$$f1(1+0.0004) = f1(1.0004) = 1$$

$$f1(1+0.0005) = f1(1.0005) = 1.001 > 1$$

$$\omega_{\text{rot}} = 0.0005 = 1/2 \cdot 10^{-3} = 1/2 \cdot \mu_{\text{tr}}$$



MN Primavara 2015 CS-UPB

(c) Facultatea de Inginerie Electrică și Calculatoare, Universitatea Politehnica București, 2015, www.cs.upb.ro

Virgula mobila IEEE754

Surse erori

• *Eroarea relativă* este:

$$e_x = |e_x| / |x| = |x - f1(x)| / |x| \leq \beta^{e-t} / (0 \cdot a_1 \dots a_e \dots \beta^e)$$

$$e_x \leq \beta^{e-t} / (0.10 \dots 0 \beta^e) = \beta^{1-t}$$

la truncare

$$e_x \leq 1/2 \cdot \beta^{1-t} \quad \text{la rotunjire}$$

In general:

$$|x - f1(x)| / |x| \leq \mu$$

de unde deducem:

$$f1(x) = x(1+\epsilon), \quad |\epsilon| \leq \mu = K \beta^{-t}$$

• De exemplu $F(10, 4, -20, 20), \Omega = 10^{20} (1-10^{-4}) = 9.999 \times 10^{19}$,

$$\omega = 10^{-20-1} = 1.0 \times 10^{-21}, \quad \mu = 1/2 \cdot 10^{-4+1} = 5 \times 10^{-4}$$



MN Primavara 2015 CS-UPB

(c) Facultatea de Inginerie Electrică și Calculatoare, Universitatea Politehnica București, 2015, www.cs.upb.ro

Virgula mobila IEEE754

Propagare erori

Numere aproximative - operații exacte

Rezultatul exact al adunării a două numere x și y , dacă operațiile se execută exact este $x+y$. În realitate, se lucrează cu *valorile inexacte* \underline{x} și \underline{y} , în care:

$$\begin{aligned} \underline{x} &= |x| - e_x \text{ și } e_x = |\underline{x} - x| \\ \underline{x} + \underline{y} &= x + y \pm e_{x+y} = x \pm e_x + y \pm e_y = x + y \pm (e_x + e_y) \\ e_{x+y} &= e_{x+y} / |x+y| = (e_x + e_y) / |x+y| = ((|x| e_x + |y| e_y) / |x+y|) \\ e_{x+y} &= |x| / |x+y| e_x + |y| / |x+y| e_y = k_x e_x + k_y e_y \end{aligned}$$


Virgula mobila IEEE754

Propagare erori

Operații aproximative - date exacte

Dacă operațiile se reprezintă *aproximativ*, iar numerele sunt reprezentate *exact*, adunarea a două numere $x = e_x \cdot b^{m_x}$ și $y = e_y \cdot b^{m_y}$ presupune aducerea celui mai mic (în acest caz y) la exponentul celui mai mare, producându-se o *denormalizare*:

$$\begin{aligned} f1(x+y) &= f1(e_x \cdot b^{m_x} + e_y \cdot b^{-(m_x-m_y)}) = f1((e_x + e_y) \cdot b^{(m_x-m_y)}) \\ f1(x+y) &= f1((e_x + e_y) \cdot (1+\mu)) \cdot b^m = f1(x \cdot (1+\mu) \cdot y) \end{aligned}$$

Rezultatul operației este *normalizat*:

$$f1(x+y) = [x \cdot (1+\mu) \cdot y] \cdot (1+\mu)$$

Denormalizarea unuia dintre termenii poate fi evitată dacă se păstrează rezultatul intermediar într-un acumulator cu lungimea $2t$ (*accumulator dublu*). În acest caz numai rezultatul final va fi afectat de trunchiere la t cifre semantice și normalizare, deci:

$$f1_2(x+y) = (x+y) \cdot (1+\mu)$$



Virgula mobila IEEE754

Reprezentarea numerelor reale - Standardul IEEE754 Precizie simplă

- reprezentare pe 32 biți
- baza $\beta = 2$
- precizie $t = 24$ biți (numerele normalizate păstrează numai 23 biți)
- Numărul real este păstrat prin 3 componente:
 - semnul: 1 bit
 - exponent: 8 biți
 - mantissa: 23 biți (logic24)
- Cei 8 biți permisi: $2^8 = 256$ valori diferite. Domeniul $[0, 255]$ este transformat în $[-127, 128]$
- La exponentul (pozitiv sau negativ) se adaugă o valoare constantă care ducă la un *exponent deplasat* sau *caracteristică* pozitivă. Factorul de deplasare pentru precizie simplă este 127.



Virgula mobila IEEE754

Propagare erori

Numere aproximative - operații exacte

Pentru scădere:

$$\underline{x} - \underline{y} = x - y \pm e_{x-y} = x \pm e_x - (y \pm e_y) = x - y \pm (e_x - e_y)$$

dе unde:

$$\begin{aligned} e_{x-y} &= e_x - e_y \\ e_{x-y} &= |x| / |x-y| e_x + |y| / |x-y| e_y = k_x e_x + k_y e_y \end{aligned}$$

În acest caz coeficienții de ponderare:

$$k_x = |x| / |x-y| \text{ și } k_y = |y| / |x-y|$$

pot lua valori foarte mari dacă $x \approx y$, deci în cazul scăderii numerelor apropiate ca ordin de mărime se pot comite erori foarte mari

În cazuri limită:

$$\underline{x} - \underline{y} = x - y = (x \pm e_x) - (y \pm e_y) = xy \pm xe_x \pm ye_y + e_x e_y = xy \pm (e_x + e_y)$$

$$e_x = x - \underline{x}$$

$$e_y = y - \underline{y}$$



Virgula mobila IEEE754

Reprezentarea numerelor reale - Standardul IEEE754

Permite *reprezentarea realilor*:

1) precizie simplă $F(2, 24, -126, 127)$, folosind 32 biți

2) precizie dublă $F(2, 53, -1022, 1023)$, se folosesc 64 biți;

3) precizie extinsă $F(2, 65, -16382, 16382)$, se folosesc 80 biți;

Întrucât $a_1 = 1$, acesta nu se mai reprezintă (este ascuns), căștișându-se astfel precizie suplimentară. Biul ascuns este evidențiat în reprezentarea:

$$f1(x) = (-1)^s \cdot 1 \cdot . \cdot e \cdot 2^{\text{exp}}$$

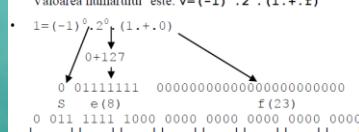


Virgula mobila IEEE754

Reprezentarea numerelor reale - Standardul IEEE754 Precizie simplă

- Domeniul deplasat $[0-255]$ reprezintă exponentii din domeniul $[-127, 128]$
- exponent_deplasat = exponent + 127

Valoarea numărului este: $V = (-1)^s \cdot 1 \cdot . \cdot e \cdot 2^{\text{exp}}$



Conditionarea problemelor

- Conditionarea unei probleme caracterizează **sensibilitatea** soluției în raport cu erorile din datele de intrare.
- O problemă este **bine conditionată** dacă erori mici în date produc de asemenea erori mici în rezultate.
- Conditionarea este o **proprietate a problemei, independentă de soluția aleasă**.
- O problemă rău conditionată este „aproape nerezolvabilă” în practică (chiar dacă problema este rezolvată exact, soluția poate fi lipsită de semnificație).
- De exemplu, la evaluarea funcției $y = f(x)$, o perturbație a datelor $x + \Delta x$ produce o perturbație a soluției $y + \Delta y = f(x + \Delta x)$, în care:
- eroarea absolută $|\Delta y| \approx |f'(x)| \cdot |\Delta x|$
- eroarea relativă $\frac{|\Delta y|}{|y|} \approx \frac{|f'(x)|}{|f(x)|} \cdot |x| \cdot \frac{|\Delta x|}{|x|}$
- Problema este **rău conditionată** dacă factorul Lipschitz $L = \frac{|f'(x)|}{|f(x)|} \cdot |x|$ este mare.



Stabilitatea numerica a algoritmilor

Dacă $f: X \rightarrow Y$ este o **problemă** și $\tilde{f}: X \rightarrow Y$ este un **algoritm**, atunci acesta este **numeric stabil** dacă pentru $\forall x \in X, \exists x^* \in X$, astfel încât:

$$\|\tilde{f}(x) - f(x)\| = O(\epsilon_n) \text{ și } \|x - x^*\| = O(\epsilon_n)$$

Algoritmul \tilde{f} destinație rezolvării problemei f este numeric stabil, dacă este îndeplinită una din condițiile:

- $\tilde{f}(x) \equiv f(x)$, adică soluția calculată aproximarea bine soluția exactă
- există x^* apropiat de x astfel încât $\tilde{f}(x) = f(x) -$ soluția calculată de algoritru cu date de intrare exacte este soluția exactă cu date ușor perturbate.

Exemple de algoritmi instabili:

- inversarea de matrice folosind determinanți
- rezolvarea sistemelor liniare prin factorizare LU fără pivotare
- utilizarea factorizării Cholesky în metoda celor mai mici pătrate (rezultate mult mai bune furnizează factorizarea QR).
- calculul valorilor proprii ca rădăcini ale polinomului caracteristic



Stabilitatea numerică a algoritmilor

- Stabilitatea numerică caracterizează erorile numerice introduse de algoritru, în ipoteza unor date de intrare exacte. Se referă la precizia algoritmului.
- Un algoritm este **instabil** dacă erorile de rotunjire produc erori mari în rezultate.
- Un algoritm numeric stabil nu introduce o sensibilitate suplimentară la perturbații.
- Un algoritm stabil dă rezultate apropiate de soluția exactă pentru o problemă bine condionată.
- Un algoritm stabil nu poate rezolva o problemă rău condionată, dar un algoritm instabil poate da soluții slabe chiar pentru o problemă bine condionată.



Bibliografie

- J. Douglas Faires, Richard L. Burden, **Numerical Methods**, 4th ed., 2012;
- Richard L. Burden, J. Douglas Faires, **Numerical Analysis**, 9th ed., 2010;

Curs01:

- V.Iorga, B.Jora, **Metode Numerică**, Ed. Albastră, 2004;
- Steven G. Krantz, **The Proof is in the Pudding**, Springer, 2007;
- Chapra, Steven C., **Applied numerical methods with MATLAB for engineers and scientists**, 3rd ed., McGraw-Hill, 2012;



METODE NUMERICE: Laborator #1

Introducere în Matlab/Octave. Funcții și instrucțiuni.

Fisiere .m. Funcții de citire scriere de tipul C

Noțiuni teoretice

Elemente Introductive

Modelarea problemelor matematice în sistemele informatic se poate face cu ajutorul *programelor specializate pentru calcule matematice* (Mathematica¹, MathCAD²) sau cu ajutorul *mediilor de programare* (MATLAB³, Maple⁴, Octave⁵, Scilab⁶). Vă recomandăm și portalul WolframAlpha⁷.

MATLAB (MATrix LABoratory) este un set de pachete de programe de înaltă performanță, dedicat calculului numeric și reprezentărilor grafice în domeniul științei și ingineriei. Aceasta integrează analiza numerică, calculul matriceal, procesarea semnalului și reprezentările grafice. Limbajul MATLAB a fost creat de profesorul Cleve B. Moler de la Universitatea din New Mexico pentru a permite un acces ușor la bibliotecile de calcul matriceal realizate în Fortran. Octave este implementarea open source pentru o parte din bibliotecile de funcții MATLAB. Vom prezenta în continuare câteva elemente de bază, ca o introducere în MATLAB/Octave.

Elementul de bază cu care MATLAB/Octave operează este matricea. În MATLAB/Octave nu se declară variabile (ca în C), și nici nu se folosesc tipuri de date predefinite, deoarece orice set de date manipulat de utilizator este văzut ca o matrice (o zonă continuă de memorie). Toate programele scrise în MATLAB/Octave sunt interpretate (nu compilate) și sunt executate linie cu linie.

Se presupune că ati instalat deja Octave. Deschideți terminalul Octave dând dublu-click pe iconița Octave sau scriind octave în consolă. Acest mediu de programare pune la dispoziția utilizatorului o consolă pentru lansarea în execuție a comenziilor. Pentru editarea codului sursă aveți nevoie de un editor text, obținut folosind comanda edit.

Rețineți că fisierile ulterioare pe care le veți crea trebuie salvate în directorul curent de lucru (comanda pwd, similară cu cea din Unix). Semnul > reprezintă promptul Octave (poate fi precedat de diverse afișări, de exemplul octave:>, cifra x indicând numărul comenzi din sesiunea curentă de lucru).

```
> pwd  
/home/student/Desktop
```

¹<http://www.wolfram.com/mathematica/>

²<http://www.ptc.com/product/mathcad>

³<http://www.mathworks.com/products/matlab/>

⁴<http://www.maplesoft.com/products/maple/>

⁵<https://www.gnu.org/software/octave/>

⁶<http://www.scilab.org>

⁷<http://www.wolframalpha.com>

În Octave se pot executa majoritatea comenziilor de consolă. Încercați `dir`, `ls`, etc.

Funcția `help` permite obținerea unor informații cu caracter general despre comenzi interne și externe Octave. Ea poate fi apelată în mai multe forme:

```
> help      % oferă informații despre elementele limbajului
            % Matlab/Octave și a fișierelor .m din
            % directorul curent;
> help <functie>    % oferă informații despre o funcție.
```

Alte funcții folosite pentru controlul general al mediului de lucru sunt:

```
> what      % listează fișiere de tipul .m, .mat etc. din
            % directorul de lucru;
> type      % listează fișierul .m menționat;
> lookfor   % returnează numele fișierelor care au în prima
            % linie a help-ului (linia H1) cuvintele
            % precizate ca argument;
> which     % calea în care este localizat un fișier sau o
            % funcție Octave;
> path      % returnează căile cu care lucrează Octave;
> who       % listează variabilele curente din memorie;
> whos      % listează variabilele curente, dimensiunile
            % și tipul acestora;
> format    % setează formatul de afișare a datelor
            % (short, long, short e, long e, hex,
            % plus, bank, rat).
```

Pentru a construi o matrice de tip coloană, folosiți comanda:

```
> v = [0; 1; 2]
```

Pentru a construi o matrice de tip linie, folosiți comanda:

```
> l = [0 1+5i 2]
```

Observați modul de reprezentare a numerelor complexe. Este unică situație în care se poate exclude semnul de înmulțire `*`, subînțelegându-se coeficientul numărului imaginär i .

Pentru a construi o matrice de 2 linii și 3 coloane, folosiți comanda:

```
> m = [0 1 2; 3 4 5]
```

Se observă că atunci când este întâlnit simbolul `;`, programul consideră că începe o linie nouă în matrice. Cazul pentru o matrice de m linii și n coloane se generalizează similar.

Sunt situații când dorim să construim vectori cu elemente multe, cu termeni în progresie aritmetică. Introducerea lor manuală ar lua mult timp, dar există următoarea comandă care simplifică lucrurile:

```
> v = [initial : pas : final]
```

Dați diverse valori pasului (chiar și pas negativ), valorii inițiale și celei finale. Pentru pas=1, comanda poate fi dată doar ca $v = [initial : final]$ sau chiar $v = initial : final$. Urmăriți rezultatul. După ce ați construit diverse matrici de tip coloană, linie sau matrice generală, introduceți comenziile:

```
> length(v)
> size(m)
```

În timp ce `length` returnează lungimea unui vector sau a unei linii, `size` returnează un vector $[n_l, n_c]$, adică numărul de linii și coloane ale matricei m . Pentru mai multe detalii, tasteți `help length`.

La fel ca în C/C++, putem accesa un element al unui vector. În acest scop, folosim comanda $v(i)$, unde i este indicele elementului accesat. Analog pentru o matrice m , folosim $m(i, j)$ pentru a accesa elementul de pe linia i , coloana j .

ATENTIE! Indicii încep de la 1, nu de la 0 ca în C/C++!!!

Pentru a extrage o submatrice dintr-o matrice m , cu liniile 11, 12, 13... și coloanele c1, c2, c3..., construim doi vectori l și c (fie linie, fie coloană, nu are relevanță) și rulați $m(l, c)$:

```
> m = [0 1 2; 3 4 5; -3 -1 10]
> m([1 2 3], [2,3])
> m([1:3], [2,3])
> v = [1 2 3];
> m(v, [2 3])
```

Am arătat mai multe metode de extragere a unei submatrici.

Folosiți operatorul ; dacă dorîți ca o comandă să nu afișeze rezultatele.

Pentru transpunerea unei matrice, se folosește operatorul '. De exemplu, comanda m' va returna transpusa (hermitica) matricei m . Putem defini mai multe tipuri de operații aritmetice pe matrice, de exemplu:

```
> m + 3      % se adună 3 la toate componente matricei m;
> 3 * m      % se înmulțesc toate componente matricei m cu 3;
> m * n      % se înmulțesc două matrici, cu dimensiunile
              % compatibile;
> m + n      % se adună două matrici, cu dimensiunile
              % compatibile;
> a .* b      % noua matrice are componente a(i,j)*b(i,j).
```

Operatorii cu . se numesc operatori Hadamard (notația vine de la produsul Hadamard, dar s-a extins și altor operatori). Acești operatori se aplică elementelor matricilor, element cu element. Considerăm următoarele exemple pentru operatorii Hadamard:

```
> a = [1 1; 2 3]
> a^2
ans =
    3     4
    8    11
> a.^2
ans =
    1     1
    4     9
```

Instrucțiuni și funcții Octave.

Funcții și constante

Rulați următoarele comenzi și observați efectul lor:

```
> cos(pi/3)
> sin(pi/4)
> ans
> inf
> eps
> realmax
> realmin
```

Instrucțiuni

Instrucțiunea de decizie `if` are sintaxa generală:

```
if condiție
    ...
endif
```

Bucla `for` are sintaxa generală:

```
for variabila = vector
    ...
endfor
```

Exemplu de program ce calculează media elementelor unui vector. Programul va fi salvat în fișierul cu numele `medie.m` și se va lansa în execuție folosind comanda `> medie` (fără extensia `.m`).

```
1 x = [2 3 4 1 -20];
2 suma = 0;
3 for var = x
4     suma = suma + var;
5 endfor
6 disp('Media este')
7 disp(suma / length(x))
```

Listing 1: Exemplu de program ce calculează media elementelor unui vector.

Bucla `while`. Sintaxa generală se poate observa în următorul exemplu:

```
1 x = 1.0;
2 while x < 1000
3     x = x*2;
4     disp(x);
5 endwhile
```

Listing 2: Exemplu de utilizare a buclei `while`.

Funcții în Octave

O funcție în Octave are același comportament ca în C/C++: primește parametri, execută instrucțiuni și întoarce un rezultat. Fiecare funcție trebuie definită într-un fișier separat, iar numele funcției trebuie să coincidă cu numele fișierului (exceptând extensia .m). Exemplu de funcție care calculează suma a două numere/vectori/matrice:

```

1 function [s] = suma(a,b)
2     s = a + b;
3 endfunction

```

Listing 3: Exemplu de funcție în Octave.

Această funcție trebuie salvată într-un fișier cu numele suma.m. Un exemplu de apel al funcției este:
> suma(3, 2) (testați și apelul suma(2:5, 3:6)).

Putem avea și funcții void, de tipul `function funcție(parametri)`. Se pot returna și mai multe rezultate (un vector de rezultate), în felul următor:

```
function [x y z] = functie(parametri).
```

Funcții de citire/scriere de tipul C

Deschiderea fișierelor

Înaintea citirii și scrierii dintr-un fișier (text sau binar), acesta trebuie deschis folosind comanda `fopen` ce are una din formele:

```
fid = fopen('numefis','mod')
[fid, mesaj] = fopen('numefis', 'mod')
```

Modul (sau permisiunea) poate fi una din alternativele:

```
r, w, a    % numai pentru citire, scriere, respectiv adăugare;
r+          % atât pentru citire cât și pentru scriere.
```

Dacă operația de deschidere fișier reușește, `fopen` întoarce un întreg nenegativ, numit identificator de fișier (`fid`). Valoarea aceasta este transmisă ca argument altor funcții de I/E care accesează fișierul deschis. Dacă deschiderea fișierului eșuează, întrucât fișierul nu există, `fid` primește valoarea -1. Fișierele standard nu trebuie deschise. Fișierul standard de ieșire are identificatorul `fid=1`, iar fișierul standard de eroare `fid=2`. Deschideți pentru citire un fișier, al căruia nume îl introduceți de la tastatură. Afisați mesajul care specifică dacă operația de deschidere a reușit sau nu.

```

1 fid = 0;
2 while fid < 1
3     numefis = input('Deschide fisier: ', 's') ;
4     [fid, mesaj] = fopen(numefis, 'r');
5     if fid == -1
6         disp(mesaj);
7     endif
8 endwhile

```

Listing 4: Exemplu de program care deschide un fișier.

Funcția `input` permite introducerea de date de la tastatură. Sirul de caractere dat ca prim parametru va fi afișat. Al doilea parametru să arată că datele introduse sunt caractere.

Funcția `disp` afișează un sir de caractere la consolă.

Scrierea datelor formatate în fișiere text

```
contor = fprintf(fid, format, A, ...)
```

Funcția întoarce numărul de octeți transferați. Descriptorii de format sunt aceiași din C. Există descriptorii specifici MATLAB/Octave:

```
%bx % afișare valoare double în hexazecimal;
%tx % afișare valoare float în hexazecimal.
```

Descriptorii pot fi precedați de caracterele: `'-'`, `'+'`, `'.'`, `'0'`, cu semnificațiile:

```
'-' % aliniere stânga;
'+' % afișează întotdeauna cu semn;
' ' % inserează un spațiu înaintea valorii afișate;
'0' % pune zerouri în locul spațiilor.
```

Citirea datelor formatate din fișiere text

```
A = fscanf(fid, format);
[A, contor] = fscanf(fid, format, dimensiune);
```

Prima formă citește date până la sfârșitul fișierului. Cea de-a doua formă citește date de o dimensiune dată ca parametru de intrare. Pentru a specifica dimensiunea datelor se utilizează una din variantele:

```
n      % cel mult n numere, caractere sau siruri;
inf    % până la sfârșitul fișierului;
[m,n]  % cel mult m*n valori, care completează o matrice
       % pe coloane.
```

Funcția `fscanf` este vectorizată și întoarce un argument matrice. Funcția acceptă și valorile `-inf`, `+inf`, `NaN`, pe care le convertește în reprezentările numerice corespunzătoare.

O linie din fișier se citește cu funcția `linie = fgetl(fid, LEN)`. Dacă se întâlnește `eof`, funcția `fgetl` întoarce `-1`. Parametrul `LEN` indică numărul de caractere de citit. Dacă acest parametru este omis, se va citi până la întâlnirea terminatorului de linie.

```
1 fid = fopen('printfile.m');
2 while 1
3     linie = fgetl(fid);
4     if ~ischar(linie), break, end
```

```

5 disp(linie);
6 endwhile
7 fclose(fid);
```

Listing 5: Exemplu de citire a unui fișier text linie cu linie și afișarea lui pe ecran.

Controlul poziției în fișier

Funcția `fseek` ne permite să ne poziționăm oriunde în fișier:

```
stare = fseek(fid, deplasare, origine)
```

Parametrul `origine` poate lua una din valorile:

```
'bof' % față de începutul fișierului
'cof' % față de poziția curentă
'eof' % față de sfârșitul fișierului
```

Parametrul `deplasare` este o valoare pozitivă sau negativă exprimată în octeți și raportată la `origine`. Funcția `ftell` determină poziția curentă în fișier, față de începutul fișierului:

```
pozitie = ftell(fid)
```

Exportul și importul datelor

Pentru memorarea variabilelor cu care se lucrează, la încheierea unei sesiuni de lucru, se poate utiliza comanda `save file`. Această comandă va salva toate variabilele curente, generate de către utilizator, într-un fișier dat ca parametru prin `file`. De exemplu:

```
> save date A B x y
```

realizează memorarea variabilelor `A`, `B`, `x`, `y` în fișierul `date.mat`. Pentru obținerea variabilelor păstrate într-un fișier `.mat` se folosește comanda `load`.

Vectorizări

Operațiile cu vectori și matrice sunt executate de MATLAB mult mai repede decât operațiile de interpretare a instrucțiunilor și executare a lor. Obținem astfel o îmbunătățire a timpului de execuție pentru programele scrise. *Vectorizarea* constă în transformarea ciclurilor `for` și `while`, acolo unde este posibil, în operații pe vectori sau matrice. De exemplu, soluția alternativă pentru secvența:

```
for n = 1:10
    x(n) = sin(n*pi/5)
end
```

este o soluție vectorizată, mult mai rapidă, atribuind memorie pentru vectorul `x` o singură dată. Mai întâi, se inițializează vectorul, apoi se folosește funcția `sin` care a fost implementată optimizat pentru calcule vectorizate.

```
n = 1:10;
x = sin(n*pi/5);
```

Probleme rezolvate

Vom prezenta, în continuare, alte exemple de vectorizări în care am folosit puterea operatorilor logici și a funcțiilor Octave.

Exemplul 1

```
1 x = -2 : 0.5 : 2;
2 for i = 1 : length(x)
3   if x(i)>=0
4     s(i) = sqrt(x(i));
5   else
6     s(i) = 0;
7   endif
8 endfor
```

Listing 6: Exemplu de utilizare a buclei for.

```
1 x = -2 : 0.5 : 2;
2 s = sqrt(x);
3 s(x<0) = 0;
```

Listing 7: Exemplu de cod vectorizat.

În acest exemplu, ne-am bazat pe faptul că funcția `sqrt` primește ca parametri și numere negative, având ca rezultat un număr complex. Am folosit operatorul `<` care pentru vectori are ca rezultat un alt vector cu 1 pe pozițiile ce satisfac condiția. Mai mult, am utilizat indexarea unui vector prin intermediul altui vector.

Exemplul 2

```
1 M = magic(3);
2 for i = 1 : 3,
3   for j = 1 : 3,
4     if (M(i,j) > 4),
5       M(i,j) = -M(i,j);
6     endif
7   endfor
8 endfor
```

Listing 8: Exemplu de utilizare a buclei for.

În acest exemplu, secvența care folosește bucla `for` se execută în 23.4956 unități de timp iar secvența vectorizată, prin intermediul funcției `find`, se execută în 2.1153 unități de timp, deci de 11 ori mai rapid.

```
1 ind = find(M > 4);
2 M(ind) = -M(ind);
```

Listing 9: Exemplu de cod vectorizat.

Exemplul 3

```

1 V = 'Sunt      multe      spatii      albe      in acest      text.';
2 len = length(V);
3 i = 1;
4 while (i<len)
5     if (V(i) == ' ' & V(i+1) == ' ')
6         for j = i:len-1
7             V(j) = V(j+1);
8         endfor
9         V(len) = 0; len = len-1;
10    else
11        i = i+1;
12    endif
13 endwhile
14 V = char(V)

```

Listing 10: Exemplu de utilizare a buclei for.

Un alt exemplu de vectorizare care folosește funcții specifice operațiilor cu vectori precum `filter` și `find` (`findstr`).

```

1 V = 'Sunt      multe      spatii      albe      in acest      text.';
2 ind = find(filter([1 1], 2, V==' ')==1);
3 V(ind) = []

```

Listing 11: Exemplu de cod vectorizat.

sau, o altă metodă care elimină toate spațiile:

```

1 V = 'Sunt      multe      spatii      albe      in acest      text.';
2 ind = findstr(V, ' ');
3 V(ind) = []

```

Listing 12: Exemplu de cod vectorizat.

Exemplul 4

Folosirea operatorilor Hadamard poate conduce la eliminarea buclelor și vectorizarea unei secvențe de program. De exemplu, secvența:

```

1 for i = 1 : n
2     for j = 1 : n
3         M(i,j) = A(i,j)/(B(i,j)*C(i,j));
4     end
5 end

```

Listing 13: Exemplu de funcție în Octave.

se poate transforma în urma folosirii operatorilor $./$ și $.*$ în:

```
1 M = A ./ (B .* C);
```

Listing 14: Exemplu de utilizare al operatorilor Hadamard.

Este recomandat ca ori de câte ori folosiți o funcție Octave, în interiorul unei bucle, să verificați (consultând help-ul Octave) dacă aceasta poate fi de folos la vectorizarea calculului. De asemenea, țineți cont de faptul că operațiile logice din instrucțiunile de ramificare pot ajuta la vectorizare.

Probleme propuse

Problema 1

Scrieți fișierul `valori.txt` cu valorile funcției $f(x) = 2x + 1$ pe intervalul $[0, 1]$ și pasul 0.1.

Problema 2

Scrieți o funcție care să calculeze suma numerelor impare mai mici ca n utilizând bucla `for`. Aceeași problemă utilizând bucla `while`. Citirea unui numar de la tastatură se face utilizând comanda:

```
var = input('Introduceți variabila:');
```

Problema 3

Scrieți o funcție care citește o matrice pătratică din fișier și verifică dacă matricea are proprietățile unui pătrat magic (suma elementelor pe linii, coloane și diagonale este aceeași).

Problema 4

Scrieți o funcție pentru determinarea dimensiunii unui fișier. Funcția are ca parametru numele fișierului.

Problema 5

Scrieți o funcție care citește un fișier text, linie cu linie și întoarce numărul total de apariții ale unui anumit sir de caractere în fișier. Funcția va afișa fiecare linie din fișier, precedată de numărul de apariții ale sirului în linie. La final, se va afisa numărul total de apariții. Funcția are semnatura:

```
function y = NumarAparitii(numefis, sir)
```

Metode Numerice @ CS-UPB

Echipa MN CS - 2015:

Profesorii:

- Pantelimon George Popescu - Seria CA;
- Florin Pop - Seria CB, CC;

Asistenti:

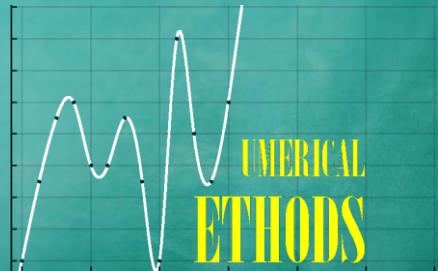
PG Popescu, Clementin Cercel, Sorin Ciolofan, Bogdan Tiganoiaia, Diana Udrea, Alexandru Tifrea, David Iancu, Bogdan Cristian Marchis, Radu Poenaru, Madalina Grosu, Andrei Vlad Postoaca, Radu Constantinescu, Roxana Istrate, Madalina Hristache, Adelina Vidovici;



MN Primavara 2015 CS-UPB
© Craiovea131 (julian) via Flickr Creative Commons Attribution-NonCommercial-ShareAlike 2.0 Generic (CC BY-NC-SA 2.0)



DESPRE MATRICI



MN Primavara 2015 CS-UPB
© Craiovea131 (julian) via Flickr Creative Commons Attribution-NonCommercial-ShareAlike 2.0 Generic (CC BY-NC-SA 2.0)



Matrici si Vectori

O matrice este un bloc dreptunghiular de elemente, unde nu numai valoarea unui element este importantă ci și poziția lui.

$$A = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$$



MN Primavara 2015 CS-UPB
© Craiovea131 (julian) via Flickr Creative Commons Attribution-NonCommercial-ShareAlike 2.0 Generic (CC BY-NC-SA 2.0)



Vectorii sunt un caz special de matrici care au doar o coloană sau doar un rând.

$$\mathbf{x} = [x_i] = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (i = 1, 2, \dots, n)$$

$$\mathbf{y} = [y_j] = [y_1 \ y_2 \ \cdots \ y_n] \quad (j = 1, 2, \dots, n)$$



MN Primavara 2015 CS-UPB
© Craiovea131 (julian) via Flickr Creative Commons Attribution-NonCommercial-ShareAlike 2.0 Generic (CC BY-NC-SA 2.0)



Tipuri de matrici

Patratice $n=m$

$$\mathbf{S} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

Diagonala ($\mathbf{In} \setminus \mathbf{In} D$)

$$\mathbf{D} = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & 0 \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$

Superior triunghiulară

$$\mathbf{U} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$



MN Primavara 2015 CS-UPB
© Craiovea131 (julian) via Flickr Creative Commons Attribution-NonCommercial-ShareAlike 2.0 Generic (CC BY-NC-SA 2.0)



Tipuri de matrici

Inferior triunghiulară

$$\mathbf{L} = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

Superior/Inferior unitriunghiulară: are toate elementele de pe diagonala egale cu 1;

Triadiagonală

$$\mathbf{T} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix}$$

Matrice banda

$$\mathbf{B} = \begin{bmatrix} a_{11} & a_{12} & 0 & a_{14} & 0 \\ a_{21} & a_{22} & a_{23} & 0 & a_{25} \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ a_{41} & 0 & a_{43} & a_{44} & a_{45} \\ 0 & a_{52} & 0 & a_{54} & a_{55} \end{bmatrix}$$



MN Primavara 2015 CS-UPB
© Craiovea131 (julian) via Flickr Creative Commons Attribution-NonCommercial-ShareAlike 2.0 Generic (CC BY-NC-SA 2.0)



Tipuri de matrici

Transpusa: A^T randurile sunt schimbate cu coloanele si invers;

$$(A^T)^T = A;$$

$$(AB)^T = B^T A^T;$$

Conjugata: \bar{A} toate numerele complexe sunt conjugate;

$$\bar{AB} = \bar{A} \bar{B};$$

Conjugata transpusa(sau adjuncta sau Hermitica adjuncta) $A^* = \bar{A}^T$;



MN Primavara 2015 CS-UPB



© Craioveanu, Iulian / Popovici, Florin / Logotipul Matematicii si Calculatoarelor - Facultatea de Matematica si Calculatoare, Universitatea din Bucuresti, www.cs.ub.ro, www.cs.ub.ro/logo

Tipuri de matrici

Involutie: $A^2 = I_n$; $A = A^{\{-1\}}$;

Nilpotenta: $A^k = I_n$, pentru k natural;

Proiectie(Idempotenta): $A^2 = A$;

Proiectie Hermitica: $A^* = A$ si $A^2 = A$

Coninvolutie: $A\bar{A} = I_n$;

A este semipozitiv definita(sau pozitiva) daca $\forall x, x^*Ax \geq 0$ si pozitiv definita(sau strict pozitiva) daca $\forall x \neq 0, x^*Ax > 0$ in C^n , iar in R^n se foloseste x^T ;



MN Primavara 2015 CS-UPB



© Craioveanu, Iulian / Popovici, Florin / Logotipul Matematicii si Calculatoarelor - Facultatea de Matematica si Calculatoare, Universitatea din Bucuresti, www.cs.ub.ro, www.cs.ub.ro/logo

Tipuri de matrici

Bloc diagonală:

$$\det(\bigoplus_{i=1}^k A_{ii}) = \prod_{i=1}^k \det A_{ii}$$

$$(A \oplus B)^{-1} = A^{-1} \oplus B^{-1}$$

$$(\det(A \oplus B))(A \oplus B)^{-1} = (\det A)(\det B)(A^{-1} \oplus B^{-1}) = ((\det B)(\det A)A^{-1} \oplus (\det A)(\det B)B^{-1})$$

$$\text{adj}(A \oplus B) = (\det B) \text{adj } A \oplus (\det A) \text{adj } B$$



MN Primavara 2015 CS-UPB



© Craioveanu, Iulian / Popovici, Florin / Logotipul Matematicii si Calculatoarelor - Facultatea de Matematica si Calculatoare, Universitatea din Bucuresti, www.cs.ub.ro, www.cs.ub.ro/logo

Tipuri de matrici

Inversa: $A^{-1} A = I_n = A A^{-1}$;

Simetrica: $A^T = A$;

Skew Simetrica: $A^T = -A$;

Ortogonala: $A^T A = I_n$;

Hermitica: $A^* = A$;

Skew Hermitica: $A^* = -A$;

Unitara: $A^* A = I_n$;

Normala: $A A^* = A^* A$;



MN Primavara 2015 CS-UPB

© Craioveanu, Iulian / Popovici, Florin / Logotipul Matematicii si Calculatoarelor - Facultatea de Matematica si Calculatoare, Universitatea din Bucuresti, www.cs.ub.ro, www.cs.ub.ro/logo



Tipuri de matrici

Involutie: $A^2 = I_n$; $A = A^{\{-1\}}$;

Nilpotenta: $A^k = I_n$, pentru k natural;

Proiectie(Idempotenta): $A^2 = A$;

Proiectie Hermitica: $A^* = A$ si $A^2 = A$

Coninvolutie: $A\bar{A} = I_n$;

A este semipozitiv definita(sau pozitiva) daca $\forall x, x^*Ax \geq 0$ si pozitiv definita(sau strict pozitiva) daca $\forall x \neq 0, x^*Ax > 0$ in C^n , iar in R^n se foloseste x^T ;



MN Primavara 2015 CS-UPB



© Craioveanu, Iulian / Popovici, Florin / Logotipul Matematicii si Calculatoarelor - Facultatea de Matematica si Calculatoare, Universitatea din Bucuresti, www.cs.ub.ro, www.cs.ub.ro/logo

Tipuri de matrici

Diagonala: are doar elemente pe diag principală, in rest 0;

Bloc diagonala:

$$A = \begin{bmatrix} A_{11} & & & & \mathbf{0} \\ & \ddots & & & \\ \mathbf{0} & & \ddots & & A_{kk} \end{bmatrix}$$

se mai scrie si astfel (suma directa)

$$A = A_{11} \oplus A_{22} \oplus \dots \oplus A_{kk} = \bigoplus_{i=1}^k A_{ii}$$



MN Primavara 2015 CS-UPB

© Craioveanu, Iulian / Popovici, Florin / Logotipul Matematicii si Calculatoarelor - Facultatea de Matematica si Calculatoare, Universitatea din Bucuresti, www.cs.ub.ro, www.cs.ub.ro/logo



Tipuri de matrici

Bloc diagonală:

$$\det(\bigoplus_{i=1}^k A_{ii}) = \prod_{i=1}^k \det A_{ii}$$

$$(A \oplus B)^{-1} = A^{-1} \oplus B^{-1}$$

$$(\det(A \oplus B))(A \oplus B)^{-1} = (\det A)(\det B)(A^{-1} \oplus B^{-1}) = ((\det B)(\det A)A^{-1} \oplus (\det A)(\det B)B^{-1})$$

$$\text{adj}(A \oplus B) = (\det B) \text{adj } A \oplus (\det A) \text{adj } B$$



MN Primavara 2015 CS-UPB



© Craioveanu, Iulian / Popovici, Florin / Logotipul Matematicii si Calculatoarelor - Facultatea de Matematica si Calculatoare, Universitatea din Bucuresti, www.cs.ub.ro, www.cs.ub.ro/logo

Tipuri de matrici

Bloc triunghiulara superioara:

$$A = \begin{bmatrix} A_{11} & \star & \star \\ & \ddots & \\ \mathbf{0} & & A_{kk} \end{bmatrix}$$

Matrice de permutare: o matrice care are exact un singur 1 pe linie sau coloana, in rest 0;



MN Primavara 2015 CS-UPB

© Craioveanu, Iulian / Popovici, Florin / Logotipul Matematicii si Calculatoarelor - Facultatea de Matematica si Calculatoare, Universitatea din Bucuresti, www.cs.ub.ro, www.cs.ub.ro/logo



Tipuri de matrici

Circulanta: $A = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \\ a_n & a_1 & a_2 & \cdots & a_{n-1} \\ a_{n-1} & a_n & a_1 & \cdots & a_{n-2} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ a_2 & a_3 & \cdots & a_n & a_1 \end{bmatrix}$

Matrice **Toeplitz** (Otto Toeplitz(1881-1940), mat. german): $a_{ij}=a_{-(j-i)}$

$$A = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & \cdots & a_n \\ a_{-1} & a_0 & a_1 & a_2 & \cdots & a_{n-1} \\ a_{-2} & a_{-1} & a_0 & a_1 & \cdots & a_{n-2} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ a_{-n} & a_{-n+1} & \cdots & \cdots & a_{-1} & a_0 \end{bmatrix}$$



Tipuri de matrici

Matrice **Hankel**(Hermann Hankel (1839-1873), mat. german):

$$a_{ij}=a_{(i+j-2)}$$

$$A = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_n \\ a_1 & a_2 & \cdots & \ddots & a_{n+1} \\ a_2 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ a_n & a_{n+1} & \cdots & a_{2n-1} & a_{2n} \end{bmatrix}$$

Matrice **Hessenberg** (Karl Adolf Hessenberg (1904-1959, mat. si inginer german):

superioara sau
inferioara

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & \star \\ a_{21} & a_{22} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{32} & \cdots & a_{nn} \end{bmatrix}$$



Tipuri de matrici

Matrice **Vandermonde**(Alexandre-Théophile Vandermonde (1735-1796), mat., muzician si chimist francez):

$$a_{ij}=x^{(1-j)}$$

$$A = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{bmatrix}$$

$$\det A = \prod_{\substack{i,j=1 \\ i>j}}^n (x_i - x_j)$$



Tipuri de matrici

Matrice **Cauchy** (Baron Augustin-Louis Cauchy (1789-1857), mat. francez):

Formata din elemente de tipul $[(a_i + b_j)^{-1}]_{i,j=1}^n$
unde a_i, b_j sunt scalari astfel incat $a_i + b_j \neq 0$;

$$\det A = \frac{\prod_{1 \leq i < j \leq n} (a_j - a_i)(b_j - b_i)}{\prod_{1 \leq i \leq j \leq n} (a_i + b_j)}$$

Matrice **Hilbert** (David Hilbert (1862-1943), mat. german): caz particular Cauchy, $a_{ii}=i$ si $b_{ij}=j-1$;

$$\det H_n = \frac{(1!2! \cdots (n-1)!)^4}{1!2! \cdots (2n-1)!}$$



Despre matrici

URMA unei matrici = suma elementelor diagonalei principale; $\text{tr}(A)=\sum(a_{ii})$;

$$\text{tr}(AA^*)=\text{tr}(A^*A);$$

$$\text{tr}(AA^*)=0 \text{ daca si numai daca } A=0;$$

DETERMINANT

$$\det(A) = |A| = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \dots & \dots & \ddots & \dots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$



Despre matrici

$$\begin{aligned} \det(A) &= \sum_k (-1)^{i+k} a_{ik} \det(A_{ik}) \\ &= \sum_k (-1)^{k+i} a_{kj} \det(A_{kj}) \end{aligned}$$

Expansiunea **Laplace**(Pierre-Simon, marquis de Laplace (1749-1827), mat. francez) dupa minori(pe linia i, respectiv coloana j);

$$\det(AB)=\det(A)\det(B);$$

$$\det A^T=\det(A);$$

$$\det A^*=\overline{\det(A)};$$



Despre matrici

RANG

Rangul unei matrici $A_{m,n}$, este un numar k ce reprezinta numarul maxim de randuri(coloane) linear independente.
 $\text{rang } A \leq \min\{m,n\}$;
 $\text{rang } A^* = \text{rang } \bar{A} = \text{rang } A^T = \text{rang } A$;
 $|\text{rang } A - \text{rang } B| \leq \text{rang}(A+B) \leq \text{rang } A + \text{rang } B$;

If $A \in M_n(F)$, $\det A \neq 0 \Leftrightarrow \text{rang } A = n$;



Factorizarea LU

- Fie A o matrice cu $\text{rang } A = k$, atunci $A = LU$, daca pentru orice i , $i \in \{1, \dots, k\}$, submatricea principala $A[\{1, \dots, i\}]$ este nesingulara; Mai mult oricare dintre factori L sau U pot fi alesi sa aliba diagonala plina de 1;
- Daca $\text{rang } A = n$ atunci A este nesingulara, la fel si toate submatricile principale ale sale si deci $A = LU$ cu L si U nesingularare;



Factorizarea LU

- Daca elementele diagonalei U sunt egale cu $1(u_{ii}=1)$ se obtine factorizarea **Crout**, iar daca elementele diagonalei matricei L sunt egale cu $1(l_{ii}=1)$ atunci se obtine factorizarea **Doolittle**; iar daca $L = U^T$, pentru o matrice A simetrica, atunci se obtine factorizarea $A = LL^T$, numita **Cholesky**;



Factorizari Triunghiulare

Factorizarea LU

$A = LU$, unde L este inferior triunghiulara, iar U este superior triunghiulara;
Este foarte utila o astfel de factorizare deoarece pentru a solutiona un sistem $Ax = b$, unde $A = LU$, putem reduce problema la a solutiona doua sisteme lineare usor de soluat, astfel, notam cu $y = Ux$ si deci vom avea doi pasi:

- Pas 1: sol. sistemul inf triunghiular $Ly = b$ pentru y ;
- Pas 2: sol. sistemul sup triunghiular $Ux = y$ pt x ;



Factorizarea LU

- $A = LU$, cu L nesingular daca si numai daca pentru orice i , $i \in \{1, \dots, n-1\}$, randul $A[\{i+1\}, \{1, \dots, i\}]$ se poate scrie ca o combinatie lineară a randurilor din submatricea principala $A[\{1, \dots, i\}]$; (A are proprietatea de inclusiune a randurilor)
- $A = LU$, cu U nesingular daca si numai daca pentru orice j , $j \in \{1, \dots, n-1\}$, coloana $A[\{1, \dots, j\}, \{j+1\}]$ se poate scrie ca o combinatie lineară a coloanelor din submatricea principala $A[\{1, \dots, j\}]$; (A are proprietatea de inclusiune a coloanelor)



Factorizarea LU

- A=LU** daca si numai daca $A[\{1, \dots, j\}]$ este nesingulara pentru orice $j=1, \dots, n$.
- A=LDU** daca si numai daca $A[\{1, \dots, j\}]$ este nesingulara pentru orice $j=1, \dots, n$, cu

$$D = \text{diag}(d_1, \dots, d_n)$$

$d_1 = a_{11}$, $d_i = \det A[\{1, \dots, i\}] / \det A[\{1, \dots, i-1\}]$, $i = 2, \dots, n$
iar L, D, U sunt unic determinati, iar L si U au 1 pe diagonală.



Factorizarea LU

- Pentru orice matrice A, exista o matrice de permutare P, o matrice unitriunghiulara inferioara L si o matrice triunghiulara superioara U astfel incat $A=PLU$ cu L avand 1 pe diagonala(sau $A=LUP$ cu U avand 1 pe diagonala) si $A=LPDU$ cu L avand 1 pe diagonala(P este unic daca A este nesingulara, doar pentru LPU);



Factorizarea LU - Crout

$$\begin{aligned}l_{11} &= a_{11}; i = 1 : n \\u_{1j} &= \frac{a_{1j}}{l_{11}}; j = 2 : n \\l_{ij} &= a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}; i \geq j \\u_{ij} &= \frac{1}{l_{ii}} * (a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}); i < j\end{aligned}$$

Factorizarea LU - Doolittle

$$\begin{aligned}u_{11} &= a_{11}; i = 1 : n \\l_{11} &= \frac{a_{11}}{u_{11}}; i = 2 : n \\l_{ij} &= \frac{1}{u_{jj}} * (a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}); i > j \\u_{ij} &= a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}\end{aligned}$$



Factorizarea LU

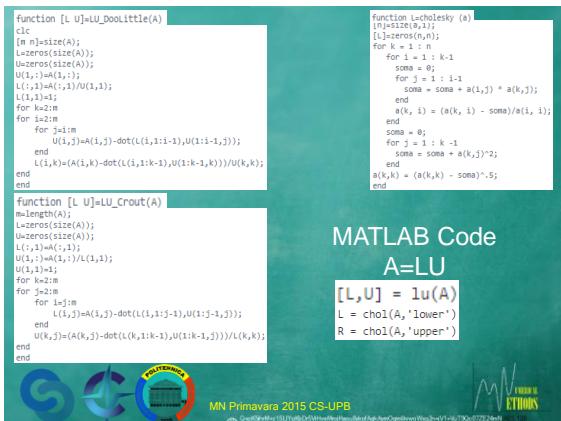
- Pentru orice matrice nesingulara A, exista o unica matrice de permutare P, o unica matrice nesingulara diagonală D, o matrice unitriunghiulară inferioară L și o matrice unitriunghiulară superioară U astfel încât $A=LPDU$ cu L și U având 1 pe diagonala;



Factorizarea LU - Cholesky

- Factorizarea LU - Cholesky** (Andre-Louis Cholesky(1875-1918) ofiter militar francez, implicat în geodezie la începutul anilor 1900)
- Aceasta metoda este un caz particular al metodelor de factorizare LDL^T pt o matrice simetrica, unde $D=L_n$;
- Fie A o matrice simetrica nxn pozitiv definita, atunci A se poate factoriza LL^T cu L unic.

$$\begin{aligned}l_{ii} &= \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}; i = 1 : n \\l_{ij} &= \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik} l_{jk}}{l_{jj}}; j = 1 : i - 1.\end{aligned}$$



Vectori și Valori proprii

Fie $A \in C^{n \times n}$. Un număr $\lambda \in C$ se numește *valoare proprie* a matricei A dacă există un vector nenul $x \in C^n$, numit *vector propriu asociat valoiei proprii* $\lambda \in C$, astfel încât $Ax = \lambda x$.

Acest sistem admite soluții nenele daca si numai daca

$$p(\lambda) = \det(\lambda I_n - A) = 0$$

unde p se numește polinom caracteristic și este polinom monic, iar ecuația se numește ecuație caracteristica. Prin urmare zerourile lui p sunt valorile proprii ale lui A.



Vectori si Valori proprii

Multimea $\lambda(A) = \{\lambda_1, \lambda_2, \dots, \lambda_n\}$ se numeste spectrul lui A

Iar numarul real $\rho(A) = \max_{\lambda \in \text{ext}(A)} (|\lambda_i|)$ raza spectrala a lui A;

Avem si urmatoarele proprietati $\sum_{i=1}^n \lambda_i = \sum_{i=1}^n A(i,i) = \text{tr}(A)$,
 $\prod_{i=1}^n \lambda_i = \det(A)$,

Daca A este de dimensiune mxn si B nxm cu $m \leq n$ atunci cele n valori proprii ale lui BA sunt cele m valori proprii ale lui AB alaturi de inca $n-m$ zeroi.



Vectori si Valori proprii

Matrici pozitiv definite

A este semipozitiv definita daca si numai daca

1. $A = A^*$ (hermitica) si are toate valorile proprii nenegative;
2. $A = A^*$ si toti minorii principali (det din submatricile principale) sunt nenegativi;
3. exista B a.i. $A = B^*B$; A este pozitiv definita da ca si numai daca B este inversabila;
4. exista T, superior triunghiulara (se poate considera si diagonalala cu elemente nenegative) a.i. $A = T^*T$; Daca A este pozitiv definita T este unic (Cholesky); A este pozitiv definita daca si numai daca T este nesingulara;
5. exista B semipozitiv definita, unica, a.i. $A = B^2$;



Vectori si Valori proprii

Transformari de asemanare.

- Daca A este asemenea cu o matrice diagonalala, spunem ca A este **diagonalizabila**.
- Daca A are n valori proprii distincte atunci este **diagonalizabila**.
- Fie B suma directa a blocurilor B_1, B_2, \dots, B_d .

B este diagonalizabila daca si numai daca B_i sunt diagonalizabile. $B = \begin{bmatrix} B_1 & & 0 \\ & \ddots & \\ 0 & & B_d \end{bmatrix} = B_1 \oplus \dots \oplus B_d$



Vectori si Valori proprii

Localizarea valorilor proprii.

Cercurile lui Gershgorin 1931 (Semyon Aranovich Gershgorin (1901-1933) mat. belarus).

$$\lambda(A) \subset \bigcup_{i=1}^n D_i, \quad D_i = \left\{ z \in C \mid |z - a_i| \leq \sum_{j \neq i} |a_j| \right\}.$$

unde D_i sunt cercurile lui Gershgorin



Vectori si Valori proprii

Transformari de asemanare.

O matricile A si B sunt asemenea daca exista T, nesingulara astfel incat

$$B = T A T^{-1}$$

Daca T este ortogonală (unitara) atunci A si B sunt asemenea ortogonal (unitar);

Daca A si B sunt asemenea atunci $\lambda(A) = \lambda(B)$ iar daca x este un vector propriu pt A asociat unei valori proprii, atunci Tx este vector propriu pentru B, asociat aceleiasi valori proprii;



Vectori si Valori proprii

Transformari de asemanare.

Daca x este un vector propriu pt A asociat valorii proprii λ_i , atunci ax este tot un vector propriu, cu alte cuvinte **vectorii proprii nu sunt unici ci au directie unica**. Atunci putem spune ca multimea vectorilor proprii asociati unei valori proprii λ_i , formeaza un subspatiu linear E_i de dimensiune $\dim(E_i)$, ce reprezinta multiplicitatea geometrica a valorii proprii λ_i .



Vectori si Valori proprii

Transformari de asemanare.

Multiplicitatea algebrica, n_i este data de multiplicitatea lui λ_i ca zero pentru polinomul caracteristic p .

Suma multiplicitatilor algebrice este egală cu gradul polinomului.

Daca $\dim(E_i) = n_i$, pt orice i , atunci există X nesingulară astfel incat

$$X^{-1}AX = \Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n).$$



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS/CC-BY-SA-NC-ND/Attribution Non-Commercial NoDerivs 4.0 International License



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS/CC-BY-SA-NC-ND/Attribution Non-Commercial NoDerivs 4.0 International License



Vectori si Valori proprii

Transformari de asemanare.

Forma Jordan (Marie Ennemond Camille Jordan (1838-1922) mat. francez)

In cazul general, pentru o matrice A , există X nesingulară astfel incat $X^{-1}AX = \text{diag}(J_1, J_2, \dots, J_q)$, unde J_i este un bloc Jordan

asociat lui λ_i ce are multiplicitatea algebrica n_i .

$$J = J_{n_1}(\lambda_1) \oplus J_{n_2}(\lambda_2) \oplus \dots \oplus J_{n_q}(\lambda_q)$$



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS/CC-BY-SA-NC-ND/Attribution Non-Commercial NoDerivs 4.0 International License



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS/CC-BY-SA-NC-ND/Attribution Non-Commercial NoDerivs 4.0 International License



Vectori si Valori proprii

Transformari de asemanare.

- Multiplicitatea geometrică a unei valori proprii este egală cu numărul de blocuri jordan corespunzătoare valorii proprii.
- Multiplicitatea algebrică a unei valori proprii este egală cu suma totală a dimensiunilor tuturor blocurilor jordan asociate valorii proprii.
- MG \leq MA pentru o anumita valoare proprie.



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS/CC-BY-SA-NC-ND/Attribution Non-Commercial NoDerivs 4.0 International License



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS/CC-BY-SA-NC-ND/Attribution Non-Commercial NoDerivs 4.0 International License



Vectori si Valori proprii

Transformari de asemanare.

De reținut următoarele ce se deduc din precedenta
Daca A are n vectori proprii linear independenti
atunci există X nesingulară astfel incat

$$X^{-1}AX = \Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n).$$

unde X are coloanele vectorii proprii linear independenti ai lui A ;

Daca o matrice A are toate valorile proprii distincte, atunci există X nesingulară astfel incat

$$X^{-1}AX = \Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n).$$

Vectori si Valori proprii

Transformari de asemanare.

- Multiplicitatea geometrică a unei valori proprii este egală cu numărul de blocuri jordan corespunzătoare valorii proprii.
- Multiplicitatea algebrică a unei valori proprii este egală cu suma totală a dimensiunilor tuturor blocurilor jordan asociate valorii proprii.
- MG \leq MA pentru o anumita valoare proprie.



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS/CC-BY-SA-NC-ND/Attribution Non-Commercial NoDerivs 4.0 International License



Vectori si Valori proprii

Transformari de asemanare.

Pentru ca orice bloc jordan are proprietatile

$$J_k(\lambda) = \lambda I_k + J_k(0) \quad J_k(0)^k = 0$$

atunci mai general o matrice J se poate scrie $J = D + N$, unde D este diagonală, iar N nilpotentă. Deci pentru o matrice A , cu $A = SJ_A S^{-1}$

$A = S(D + N)S^{-1} = SDS^{-1} + SNS^{-1} = A_D + A_N$
adică este suma de două matrici una diagonalizabilă și una nilpotentă.

Vectori si Valori proprii

Transformari de asemanare.

- Multiplicitatea geometrică a unei valori proprii este egală cu numărul de blocuri jordan corespunzătoare valorii proprii.
- Multiplicitatea algebrică a unei valori proprii este egală cu suma totală a dimensiunilor tuturor blocurilor jordan asociate valorii proprii.
- MG \leq MA pentru o anumita valoare proprie.



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS/CC-BY-SA-NC-ND/Attribution Non-Commercial NoDerivs 4.0 International License



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS/CC-BY-SA-NC-ND/Attribution Non-Commercial NoDerivs 4.0 International License



Vectori si Valori proprii

Transformari de asemanare.

In cazul general, cea mai simpla forma ce poate fi obtinuta prin transformari de asemanare este forma Jordan.

Structura Jordan (respectiv nr si dimensiunile blocurilor) este foarte sensibila la perturbatiile numerice in elementele matricei si, din acest motiv, calculul numeric al formei canonice Jordan este dificil si nu este recomandat pentru calculul valorilor proprii intr-o aritmetică aproximativa.



MN Primavara 2015 CS-UPB

© Craioveanu Mihai / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs



MN Primavara 2015 CS-UPB

© Craioveanu Mihai / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs

Vectori si Valori proprii

Transformari de asemanare.

Forma Schur

O consecinta este urmatoarea

Pt orice matrice A exista Q unitara astfel incat $Q^*HQ=S$, unde S este superior triunghiulara si contine pe diagonală principala valorile proprii ale lui A.



MN Primavara 2015 CS-UPB

© Craioveanu Mihai / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs



MN Primavara 2015 CS-UPB

© Craioveanu Mihai / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs

Norme

Despre Norme

O norma vectoriala se defineste astfel:

$$\| \cdot \| : \mathbb{R}^n \rightarrow \mathbb{R}^+, \quad x \mapsto \| x \|$$

si are urmatoarele proprietati

- 1º $\| x \| \geq 0,$
- 2º $\| x \| = 0 \Rightarrow x = 0,$
- 3º $\| x + y \| \leq \| x \| + \| y \|,$
- 4º $\| \alpha \cdot x \| = |\alpha| \cdot \| x \|, \quad \forall \alpha \in \mathbb{C}.$



MN Primavara 2015 CS-UPB

© Craioveanu Mihai / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs



MN Primavara 2015 CS-UPB

© Craioveanu Mihai / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs

Vectori si Valori proprii

Transformari de asemanare.

Forma Schur(Issai Schur(1875-1941) mat. German, nascut in Belarus)

Pt orice matrice A exista T matrice astfel incat $A=T^{-1}UT$, unde U este o matrice superior triunghiulara.

Vectorii coloana ale matricei T reprezinta o baza pentru spatiul C^n si pot inlocui vectorii proprii ai matricei A.



MN Primavara 2015 CS-UPB

© Craioveanu Mihai / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs



MN Primavara 2015 CS-UPB

© Craioveanu Mihai / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs

Vectori si Valori proprii

Transformari de asemanare.

Forma Schur

In cazul real consecinta este urmatoarea

Pt orice matrice A exista Q ortogonală astfel incat $Q^*AQ=S$, unde S este cvasi-superior triunghiulara si contine pe diagonală principala valorile proprii ale lui A patratice

Vectori si Valori proprii

Transformari de asemanare.

Forma Schur

In cazul real consecinta este urmatoarea

Pt orice matrice A exista Q ortogonală astfel incat $Q^*AQ=S$, unde S este cvasi-superior triunghiulara si contine pe diagonală principala valorile proprii ale lui A patratice

$$S = \begin{bmatrix} S_{11} & S_{12} & \dots & S_{1q} \\ 0 & S_{22} & \dots & S_{2q} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & S_{qq} \end{bmatrix} \quad \text{unde } S_{ii} \text{ sunt matrici de ordin 1 sau 2, cu}$$

$$\lambda(A) = \lambda(S) = \bigcup_{i=1}^q \lambda(S_{ii})$$



MN Primavara 2015 CS-UPB

© Craioveanu Mihai / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs



MN Primavara 2015 CS-UPB

© Craioveanu Mihai / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs

Norme

Despre Norme

O norma vectoriala se defineste astfel:

Norme

Despre Norme

O norma vectoriala generala este norma Holder (Otto Ludwig Hölder (1859 - 1937) matematician german)

$$\| x \|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{1/p},$$

si are urmatoarele particularizari

- $p = 1 \Rightarrow \| x \|_1 = \sum_{i=1}^n |x_i|,$ norma Napoleon.
- $p = 2 \Rightarrow \| x \|_2 = \sqrt{\sum_{i=1}^n |x_i|^2},$ norma Euclid.
- $p = \infty \Rightarrow \| x \|_\infty = \max_i |x_i|.$



MN Primavara 2015 CS-UPB

© Craioveanu Mihai / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs



MN Primavara 2015 CS-UPB

© Craioveanu Mihai / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs

Norme

Despre Norme

Alte proprietati ale normelor vectoriale

$$\begin{aligned}\| \mathbf{x} \|_2 &\leq \| \mathbf{x} \|_1 \leq \sqrt{n} \cdot \| \mathbf{x} \|_2, \\ \| \mathbf{x} \|_\infty &\leq \| \mathbf{x} \|_1 \leq n \cdot \| \mathbf{x} \|_\infty, \\ \| \mathbf{x} \|_\infty &\leq \| \mathbf{x} \|_2 \leq \sqrt{n} \cdot \| \mathbf{x} \|_\infty.\end{aligned}$$



MN Primavara 2015 CS-UPB



FACULTATEA DE MATEMATICA SI INFORMATICA

Norme

Despre Norme

Cateva exemple de norme matriciale

Frobenius/Euclid:

$$\| A \|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^m |A_{ij}|^2}.$$

Norme subordonate:

$$\| A \|_p = \sup_{x \neq 0} \frac{\| A \cdot x \|_p}{\| x \|_p}, \quad \| A \|_{pq} = \sup_{x \neq 0} \frac{\| A \cdot x \|_p}{\| x \|_q}.$$

Altele:

$$\begin{aligned}\| A \|_1 &= \max_j \sum_{i=1}^n |A_{ij}|, \\ \| A \|_\infty &= \max_i \sum_{j=1}^n |A_{ij}|, \\ \| A \|_2 &= \sqrt{\rho(A \cdot A^T)} = \sigma_2(A).\end{aligned}$$



MN Primavara 2015 CS-UPB



FACULTATEA DE MATEMATICA SI INFORMATICA

Norme

Matrici rau conditionate

Un sistem $Ax=b$ este **Rau Conditionat** atunci cand pentru mici perturbatii ale lui A si b apar mari perturbatii in vectorul solutie.

Si invers, un sistem $Ax=b$ este **Bine Conditionat** atunci cand pentru mici perturbatii ale lui A si b apar mici perturbatii ale solutiei.



MN Primavara 2015 CS-UPB



FACULTATEA DE MATEMATICA SI INFORMATICA

Norme

Despre Norme

La fel ca si la vectori o norma matriciala se defineste asemanator si are urmatoarele proprietati:

- 1° $\| A \| \geq 0,$
- 2° $\| A \| = 0 \Rightarrow A = 0,$
- 3° $\| A + B \| \leq \| A \| + \| B \|,$
- 4° $\| \alpha \cdot A \| = |\alpha| \cdot \| A \|,$
- 5° $\| A * B \| \leq \| A \| \cdot \| B \|,$

ultima proprietate se refera la norme consistente.



MN Primavara 2015 CS-UPB



Norme

Despre Norme

Cateva exemple de norme matriciale

Frobenius/Euclid:

$$\| A \|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^m |A_{ij}|^2}.$$

Norme subordonate:

$$\| A \|_p = \sup_{x \neq 0} \frac{\| A \cdot x \|_p}{\| x \|_p}, \quad \| A \|_{pq} = \sup_{x \neq 0} \frac{\| A \cdot x \|_p}{\| x \|_q}.$$

Altele:

$$\begin{aligned}\| A \|_1 &= \max_j \sum_{i=1}^n |A_{ij}|, \\ \| A \|_\infty &= \max_i \sum_{j=1}^n |A_{ij}|, \\ \| A \|_2 &= \sqrt{\rho(A \cdot A^T)} = \sigma_2(A).\end{aligned}$$



MN Primavara 2015 CS-UPB



FACULTATEA DE MATEMATICA SI INFORMATICA

Norme

Despre Norme

Alte proprietati ale normelor matriciale

$$\begin{aligned}\| A \|_2 &\leq \| A \|_F \leq \sqrt{n} \cdot \| A \|_2, \\ \frac{1}{\sqrt{n}} \cdot \| A \|_2 &\leq \| A \|_\infty \leq \sqrt{m} \cdot \| A \|_\infty, \\ \frac{1}{\sqrt{m}} \cdot \| A \|_1 &\leq \| A \|_F \leq \sqrt{n} \cdot \| A \|_1.\end{aligned}$$

si $|\lambda| \leq p(A) \leq \|A\|$. (deoarece $AX=\lambda X$, unde $X=[x \ x \dots \ x]$)



MN Primavara 2015 CS-UPB



Norme

Matrici rau conditionate

Un sistem $Ax=b$ este **Rau Conditionat** atunci cand pentru mici perturbatii ale lui A si b apar mari perturbatii in vectorul solutie.

Si invers, un sistem $Ax=b$ este **Bine Conditionat** atunci cand pentru mici perturbatii ale lui A si b apar mici perturbatii ale solutiei.



MN Primavara 2015 CS-UPB



FACULTATEA DE MATEMATICA SI INFORMATICA

Norme

Matrici rau conditionate

Rau conditionarea nu este o problema cand folosim o **precizie aritmetica infinita**, dar devine o problema cand avem **precizie finita** deoarece erorile de rotunjire modifica elementele din A si b , iar la modificarile mari spunem ca sistemul este rau conditionat.



MN Primavara 2015 CS-UPB



Norme

Matrici rau conditionate

Exemplu sistem rau conditionat

$$\begin{aligned}x_1 + x_2 &= 2 \\x_1 + 1.0001x_2 &= 2.0001\end{aligned}$$

cu solutia $x_1 = 1$, $x_2 = 1$;

Daca aplicam o mica modificare coeficientului a₂₂ din 1.0001 in 0.9999, noul sistem devine

$$\begin{aligned}x_1 + x_2 &= 2 \\x_1 + 0.9999x_2 &= 2.0001\end{aligned}$$

si are noua solutie $x_1 = 3$ si $x_2 = -1$;



MN Primavara 2015 CS-UPB



MN Primavara 2015 CS-UPB



Norme

Matrici rau conditionate

De asemenea se mai considera ca o matrice este rau conditionata daca modulul determinantului sau este mic. Cu toate acestea niciuna dintre aceste metode nu este sigura. Si deci cea mai sigura metoda de a determina daca o matrice este sau nu rau conditionata este sa-i calculam **Numarul de Conditionare**.



MN Primavara 2015 CS-UPB



MN Primavara 2015 CS-UPB



Norme

Numarul de Conditionare

si solutinonand pentru δx , obtinem

$$\delta x = A^{-1} \delta b, \text{ de unde deducem}$$

$$\|\delta x\| \leq \|A^{-1}\| \|\delta b\|$$

si comparand cele doua inegalitati vom avea

$$\|\delta x\| \leq \|A\| \|A^{-1}\| \|\delta b\|, \text{ adica}$$

$$\boxed{\frac{\|\delta x\|}{\|x\|} \leq \|A\| \|A^{-1}\| \frac{\|\delta b\|}{\|b\|} = C(A) \frac{\|\delta b\|}{\|b\|}}$$



MN Primavara 2015 CS-UPB



MN Primavara 2015 CS-UPB



Norme

Matrici rau conditionate

Sunt mai multe variante de a verifica daca o matrice A este rau conditionata, amintim de urmatoarele:

- pentru o matrice A, calculam inversa A^{-1} si apoi calculam AA^{-1} si comparam cu I_n , daca rezultatele difera cu mult atunci avem probabil o matrice rau conditionata;
- la fel si daca calculam $(A^{-1})^{-1}$ si comparam cu A;
- daca elementele lui A si b s-au modificar usor, iar solutia sistemului s-a schimbat drastic atunci iara avem probabil un sistem rau conditionat.



MN Primavara 2015 CS-UPB



Norme

Numarul de Conditionare(pentru SEL)

Numarul de conditionare este o masura a sensibilitatii unui sistem la mici modificari ale oricarui parametru.

Pentru un sistem $Ax=b$ avem

$$\|\delta b\| \leq \|A\| \|x\|$$

Daca modificam, sa spunem vectorul b cu δb , se va produce o modificare a solutiei x cu δx , deci

$$A(x+\delta x) = b + \delta b, \text{ de unde obtinem}$$

$$\delta x = \delta b$$



MN Primavara 2015 CS-UPB



Norme

Numarul de Conditionare

Unde $C(A)$ este numarul de conditionare al matricei A

$$C(A) = \|A\| \|A^{-1}\|$$

Analog, daca se considera o modificare a lui A cu δA , se obtine

$$\frac{\|\delta x\|}{\|x + \delta x\|} \leq C(A) \frac{\|\delta A\|}{\|A\|}$$



MN Primavara 2015 CS-UPB



Norme

Numarul de Conditionare

Daca $C(A)$ este un numar cu o singura zecimala spunem ca matricea A este bine conditionata, iar daca $C(A)$ este nu numar mare spunem ca A este rau conditionata.



Bibliografie Curs 2

- Joe D. Hoffman, Numerical Methods for Engineers and Scientists, 2nd edition, Marcel Dekker, 2001;
- Roger A. Horn, Charles R. Johnson, Matrix Analysis, 2nd edition, Cambridge, 2012;



METODE NUMERICE: Laborator #2

Operații cu matrice în Matlab. Rezolvarea recursivă a sistemelor triunghiulare. Inversarea matricelor prin partitōnare.

Noțiuni teoretice

Factorizarea LU

Această metodă presupune descompunerea unei matrice A astfel:

$$A = LU$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \dots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \dots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \dots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

unde L - matrice inferior triunghiulară, U - matrice superior triunghiulară.

În funcție de condițiile impuse, se disting factorizările: Crout ($u_{ii} = 1$), Doolittle ($l_{ii} = 1$) și Cholesky (pentru matrice simetrice și pozitiv-definite). Importanța factorizării LU constă în transformarea unui sistem cu matrice pătratică în două sisteme triunghiulare.

Factorizarea Crout

Sistemul de ecuații din forma generală a factorizării LU este supradeterminat. Pentru rezolvarea lui, condiția ce se impune este: $u_{ii} = 1$, pentru $i = 1 : n$. Factorizarea Crout este:

Pentru $p = 1 : n$

$$\begin{aligned} l_{ip} &= a_{ip} - \sum_{k=1}^{p-1} l_{ik} \cdot u_{kp}, \quad i = p : n \\ u_{pj} &= \frac{1}{l_{pp}} \cdot \left(a_{pj} - \sum_{k=1}^{p-1} l_{pk} \cdot u_{kj} \right), \quad j = p + 1 : n \end{aligned}$$

Factorizarea Doolittle

În acest caz, condiția ce se impune este: $l_{ii} = 1$, pentru $i = 1 : n$, iar factorizarea Doolittle este:

Pentru $p = 1 : n$

$$u_{pj} = a_{pj} - \sum_{k=1}^{p-1} l_{pk} \cdot u_{kj}, \quad j = p : n$$

$$l_{ip} = \frac{1}{u_{pp}} \cdot \left(a_{ip} - \sum_{k=1}^{p-1} l_{ik} \cdot u_{kp} \right), \quad i = p + 1 : n$$

Factorizarea Cholesky

Factorizarea Cholesky impune ca $U = L^T$. Ea se aplică numai pentru o matrice simetrică ($A = A^T$) și pozitiv definită ($x^T Ax > 0, \forall x \in R^n, x \neq 0$). Factorizarea Cholesky este:

Pentru $i = 1 : n$

Pentru $j = 1 : i - 1$

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot l_{jk}}{l_{jj}}$$

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$$

Observație: În implementarea metodei în Octave, în bucla principală `for i=1:n`, se vor calcula mai întâi elementele l_{ij} , apoi elementele l_{ii} .

Rezolvarea sistemelor triunghiulare

Sistem inferior triunghiular:

Formă:
$$\begin{cases} a_{11}x_1 & = b_1 \\ a_{21}x_1 + a_{22}x_2 & = b_2 \\ \vdots & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n & = b_n \end{cases}$$

Soluția:
$$x_i = \frac{b_i - \sum_{j=1}^{i-1} A_{ij} \cdot x_j}{A_{ii}}, \text{ pentru } i = 1 : n.$$

Sistem superior triunghiular:

Formă:
$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & = b_1 \\ a_{22}x_2 + \cdots + a_{2n}x_n & = b_2 \\ \vdots & \vdots \\ a_{nn}x_n & = b_n \end{cases}$$

Soluția:
$$x_i = \frac{b_i - \sum_{j=i+1}^n A_{ij} \cdot x_j}{A_{ii}}, \text{ pentru } i = n : 1.$$

Inversarea matricelor prin metoda partitioanării

În unele cazuri (de exemplu când anumite zone ale matricei conțin elemente nule), se poate diviza matricea de inversat în patru submatrice A_1, A_2, A_3 și A_4 astfel încât matricele de pe diagonala principală A_1 și A_4 să fie pătratice:

$$A = \begin{bmatrix} A_1 & A_3 \\ A_2 & A_4 \end{bmatrix}$$

Dacă se notează inversa matricei A cu

$$X = A^{-1} = \begin{bmatrix} X_1 & X_3 \\ X_2 & X_4 \end{bmatrix},$$

este valabilă ecuația matriceală:

$$A \cdot A^{-1} = \begin{bmatrix} A_1 & A_3 \\ A_2 & A_4 \end{bmatrix} \cdot \begin{bmatrix} X_1 & X_3 \\ X_2 & X_4 \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}.$$

În final, rezultă:

$$\begin{cases} X_1 = (A_1 - A_3 \cdot A_4^{-1} \cdot A_2)^{-1} \\ X_2 = -A_4^{-1} \cdot A_2 \cdot X_1 \\ X_3 = -A_1^{-1} \cdot A_3 \cdot X_4 \\ X_4 = (A_4 - A_2 \cdot A_1^{-1} \cdot A_3)^{-1} \end{cases}$$

Probleme rezolvate

Problema 1

Scriptă o funcție OCTAVE ce realizează factorizarea Crout. Funcția primește ca parametru matricea supusă factorizării și returnează matricele L și U rezultante.

Soluție:

```

1 function [L U] = crout(A)
2 [n n] = size(A);
3 L = zeros(n);
4 U = eye(n);
5 L(1:n,1) = A(1:n,1);

6
7 for p = 1 : n
8   for i = p : n
9     s = 0;
10    for k = 1 : p-1
11      s = s+L(i,k)*U(k,p);
12    endfor

13
14    % echivalent pentru calculul sumei
15    % s = L(i,1:k-1)*U(1:k-1,k);
16

```

```

17      L(i,p) = A(i,p)-s;
18  endfor
19
20  for j = p+1 : n
21      % -----
22      s = 0;
23      for k = 1 : p-1
24          s = s+L(p,k)*U(k,j);
25      endfor
26
27      % echivalent pentru calculul sumei
28      % s = L(k,1:k-1)*U(1:k-1,j);
29
30      U(p,j) = (A(p,j)-s)/L(p,p);
31  endfor
32 endfor
33 endfunction

```

Listing 1: Factorizare Crout.

Problema 2

Determinați care sunt matricele L și U rezultate din factorizarea Crout pentru matricea:

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

Soluție:

Din definiție, matricele L și U vor avea forma următoare:

$$L = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix}, \text{ respectiv } U = \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

Din $A = L * U$ se pot deduce relațiile:

$$\left\{ \begin{array}{l} l_{11} = 2 \\ l_{11}u_{12} = -1 \\ l_{11}u_{13} = 3 \\ l_{21} = 4 \\ l_{21}u_{12} + l_{22} = 5 \\ l_{21}u_{13} + l_{22}u_{23} = 1 \\ l_{31} = 2 \\ l_{31}u_{12} + l_{32} = 1 \\ l_{31}u_{13} + l_{32}u_{23} + l_{33} = 2 \end{array} \right.$$

Din acest sistem de 9 ecuații cu 9 necunoscute, rezultă elementele matricelor căutate:

$$L = \begin{bmatrix} 2 & 0 & 0 \\ 4 & 7 & 0 \\ 2 & 2 & \frac{3}{7} \end{bmatrix} \quad U = \begin{bmatrix} 1 & \frac{-1}{2} & \frac{3}{2} \\ 0 & 1 & \frac{-5}{7} \\ 0 & 0 & 1 \end{bmatrix}$$

Problema 3

Pentru aceeași matrice A dată la exercițiul anterior, stabiliți care sunt matricele L și U din factorizarea Doolittle.

Soluție:

Matricele L și U vor avea forma următoare:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix}, \text{ respectiv } U = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Folosind relația $A = L * U$, se obține următorul sistem de ecuații:

$$\left\{ \begin{array}{l} u_{11} = 2 \\ u_{12} = -1 \\ u_{13} = 3 \\ l_{21}u_{11} = 4 \\ l_{21}u_{12} + u_{22} = 5 \\ l_{21}u_{13} + u_{23} = 1 \\ l_{31}u_{11} = 2 \\ l_{31}u_{12} + l_{32}u_{22} = 1 \\ l_{31}u_{13} + l_{32}u_{23} + u_{33} = 2 \end{array} \right.$$

Din sistemul de 9 ecuații cu 9 necunoscute, rezultă elementele matricelor L și U :

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & \frac{2}{7} & 1 \end{bmatrix} \quad U = \begin{bmatrix} 2 & -1 & 3 \\ 0 & 7 & -5 \\ 0 & 0 & \frac{3}{7} \end{bmatrix}$$

Problema 4

Script o funcție OCTAVE care aplică factorizarea Doolittle pentru o matrice A , primită ca parametru la intrare și returnează matricele L și U corespunzătoare.

Soluție:

```

1 function [L U] = doolittle(A)
2 [n n] = size(A);
3 L = eye(n);
4 U = zeros(n);
5
6 for p = 1 : n
7   for j = p : n
8     s = 0;
9     for k = 1 : p-1
10       s = s+L(p,k)*U(k,j);
11     endfor
12     U(p,j) = A(p,j)-s;
13   endfor
14
15   for i = p+1 : n

```

```

16     s = 0;
17     for k = 1 : p-1
18         s = s+L(i,k)*U(k,p);
19     endfor
20     L(i,p) = (A(i,p)-s)/U(p,p);
21     endfor
22 endfor
23 endfunction

```

Listing 2: Factorizare Doolittle.

Problema 5

Scriptă o funcție OCTAVE care aplică factorizarea Cholesky pentru o matrice A primită ca parametru la intrare și returnează matricea L corespunzătoare.

Soluție:

```

1 function L = cholesky(A)
2 [n n] = size(A);
3 L = zeros(n);
4
5 for i = 1 : n
6     for j = 1 : i-1
7         s = 0;
8         for k = 1 : j-1
9             s = s+L(i,k)*L(j,k);
10        endfor
11        L(i,j) = (A(i,j)-s)/L(j,j);
12    endfor
13
14    s = 0;
15    for k = 1 : i-1
16        s = s + L(i,k)*L(i,k);
17    endfor
18    L(i,i) = sqrt(A(i,i)-s);
19 endfor
20 endfunction

```

Listing 3: Factorizare Cholesky.

Problema 6

Considerându-se matricea A specificată mai jos (simetrică și pozitiv definită), determinați matricea L care satisfac condițiile din factorizarea Cholesky.

$$A = \begin{bmatrix} 14 & 8 & 3 \\ 8 & 5 & 2 \\ 3 & 2 & 1 \end{bmatrix}$$

Soluție:

Matricea L căutată va avea următoarea formă:

$$L = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix}$$

Condiția din cadrul factorizării Cholesky este: $A = L * L^t$. Se pot deduce relațiile următoare:

$$\begin{cases} l_{11}^2 = 14 \\ l_{11}l_{21} = 8 \\ l_{11}l_{31} = 3 \\ l_{21}^2 + l_{22}^2 = 5 \\ l_{21}l_{31} + l_{22}l_{32} = 2 \\ l_{31}^2 + l_{32}^2 + l_{33}^2 = 1 \end{cases}$$

Astfel, obținem matricea L :

$$L = \begin{bmatrix} \sqrt{14} & 0 & 0 \\ \frac{4\sqrt{14}}{7} & \frac{\sqrt{21}}{7} & 0 \\ \frac{3\sqrt{14}}{14} & \frac{2\sqrt{21}}{21} & \frac{\sqrt{6}}{6} \end{bmatrix}$$

Problema 7

Dându-se matricea A de mai jos, calculați inversa ei prin metoda partitioanării.

$$A = \begin{bmatrix} 2 & 0 & 1 & 3 \\ 1 & 1 & 0 & 2 \\ 3 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Soluție:

Considerăm următoarea partitioanare: $A = \begin{bmatrix} A_1 & A_3 \\ A_2 & A_4 \end{bmatrix}$, unde

$$A_1 = \begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix}, A_2 = \begin{bmatrix} 3 & -1 \\ 0 & 0 \end{bmatrix}, A_3 = \begin{bmatrix} 1 & 3 \\ 0 & 2 \end{bmatrix}, A_4 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

De asemenea, putem determina următoarele inverse: $A_1^{-1} = \begin{bmatrix} \frac{1}{2} & 0 \\ -\frac{1}{2} & 1 \end{bmatrix}$, $A_4^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

Înlocuind toate aceste matrice în relațiile metodei, putem determina submatricele X_1, X_2, X_3 și X_4 care compun inversa matricei A , și anume X . Astfel:

$$X_1 = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}, X_2 = \begin{bmatrix} 2 & -1 \\ 0 & 0 \end{bmatrix}, X_3 = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & -\frac{5}{2} \end{bmatrix}, X_4 = \begin{bmatrix} -1 & -4 \\ 0 & 1 \end{bmatrix}$$

De aici, rezultă matricea finală X , adică inversa matricei A :

$$X = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{5}{2} \\ 2 & -1 & -1 & -4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Probleme propuse

Problema 1

Pentru matricea A dată mai jos, determinați matricele L și U rezultate din factorizarea Crout.

$$A = \begin{bmatrix} 1 & 3 & 0 \\ 1 & 1 & 2 \\ -2 & 0 & 1 \end{bmatrix}$$

Problema 2

Fie matricea A de la *Problema 1*, determinați matricele L și U rezultate din factorizarea Doolittle.

Problema 3

Fiind dată matricea $A \in R^{n \times n}$ și un vector coloană $b \in \mathbb{R}$, să se scrie o funcție OCTAVE care rezolvă sistemul $Ax = b$, folosindu-vă de o funcție care aplică factorizarea LU pe matricea A și de două funcții care rezolvă sisteme superior/inferior triunghiulare:

```
function x = SST(A, b)
function x = SIT(A, b)
```

Problema 4

Calculați inversa pentru matricea A dată mai jos, folosind metoda partitioanării.

$$A = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 2 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Metode Numerice @ CS-UPB

Echipa MN CS - 2015:

Profesorii:

- Pantelimon George Popescu - Seria CA;
- Florin Pop - Seria CB, CC;

Asistenti:

PG Popescu, Clementin Cercel, Sorin Ciolofan, Bogdan Tiganoia, Diana Udrea, Alexandru Tifrea, David Iancu, Bogdan Cristian Marchis, Radu Poenaru, Madalina Grosu, Andrei Vlad Postoaca, Radu Constantinescu, Roxana Istrate, Madalina Hristache, Adelina Vidovici;



MN Primavara 2015 CS-UPB
© Craiovea Mihai (131) | Flickr - Photo Sharing! | Flickr - Photo Sharing!



NM
NUMERICAL
METHODS

Transformari ortogonale Polinoame ortogonale



MN Primavara 2015 CS-UPB
© Craiovea Mihai (131) | Flickr - Photo Sharing! | Flickr - Photo Sharing!



Transformari ortogonale

Produsul scalar $\langle x, y \rangle = y^* x$ se numeste produs euclidian pentru doi vectori $x, y \in \mathbb{C}^n$.

Norma euclidiana pentru un vector complex x este valoarea reala $\|x\|_2 = \sqrt{\langle x, x \rangle}$.

- $\langle ax_1 + bx_2, y \rangle = a\langle x_1, y \rangle + b\langle x_2, y \rangle$;
- $\langle x, ay_1 + by_2 \rangle = \bar{a}\langle x, y_1 \rangle + \bar{b}\langle x, y_2 \rangle$;
- CS ineq: $|\langle x, y \rangle| \leq \|x\|_2 \|y\|_2$;



MN Primavara 2015 CS-UPB
© Craiovea Mihai (131) | Flickr - Photo Sharing! | Flickr - Photo Sharing!



NM
NUMERICAL
METHODS

Transformari ortogonale

- Vectorii v_1, v_2, \dots, v_n sunt ortogonali daca $v_i^T v_j = 0$ pentru orice $i \neq j$;
- Vectorii v_1, v_2, \dots, v_n sunt ortonormali daca $v_i^T v_j = 0$ pentru orice $i \neq j$ si $v_i^T v_i = 1$ pt orice i ;

Orice lista de vectori ortonormali este linear independenta si orice lista de vectori ortogonali, diferiti de vectorul zero este linear independenta.



MN Primavara 2015 CS-UPB
© Craiovea Mihai (131) | Flickr - Photo Sharing! | Flickr - Photo Sharing!



Transformari ortogonale

Din orice lista de vectori linear independenti S pot obtine o lista de vectori ortonormali cu aceeasi span.

$$\text{span}(S) = \left\{ \sum_{i=1}^k \lambda_i v_i \mid k \in \mathbb{N}, v_i \in S, \lambda_i \in \mathbb{K} \right\}$$

Procesul se poate face in multe moduri dar exista un proces sistematizat, Gram-Schmidt (Jørgen Pedersen Gram((1850 - 1916) - mat. danez) si Erhard Schmidt((1876 - 1959) - mat. german))



MN Primavara 2015 CS-UPB
© Craiovea Mihai (131) | Flickr - Photo Sharing! | Flickr - Photo Sharing!



NM
NUMERICAL
METHODS

Transformari ortogonale

Daca v_1, v_2, \dots, v_n linear independenti, prin GS obtin o lista z_1, z_2, \dots, z_n de vectori ortonormali cu $\text{span}\{v_1, v_2, \dots, v_n\} = \text{span}\{z_1, z_2, \dots, z_n\}$ pt orice $k=1 \dots n$; Vectori z_i se calculeaza astfel:

$$y_1 = x_1; z_1 = y_1 / \|y_1\|;$$

$$y_2 = x_2 - \langle x_2, z_1 \rangle z_1; z_2 = y_2 / \|y_2\|;$$

...

$$y_k = x_k - \langle x_k, z_{k-1} \rangle z_{k-1} - \langle x_k, z_{k-2} \rangle z_{k-2} - \dots - \langle x_k, z_1 \rangle z_1$$

$$z_k = y_k / \|y_k\|;$$



MN Primavara 2015 CS-UPB
© Craiovea Mihai (131) | Flickr - Photo Sharing! | Flickr - Photo Sharing!



Transformari ortogonale

Daca consideram $Z=[z_1 \dots z_n]$ si $X=[x_1 \dots x_n]$ atunci procesul GS ne obtine o factorizare $X=ZR$ unde R este nesingulara si superior triunghiulara.

Daca x_1, \dots, x_k sunt ortonormali si $x_1, \dots, x_k, x_{k+1}, \dots, x_n$ sunt linear independenti atunci procesul GS obtine $x_1, \dots, x_k, x_{k+1}, \dots, x_n$ ortonormali.



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%C3%A3u_Sistemelor_Bucure%C5%83ti_2015_CS-UPB.jpg&oldid=1673062722) de la Wikipedia, licen%C3%A7ă CC-BY-SA 3.0

MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%C3%A3u_Sistemelor_Bucure%C5%83ti_2015_CS-UPB.jpg&oldid=1673062722) de la Wikipedia, licen%C3%A7ă CC-BY-SA 3.0

Transformari ortogonale

Daca H este o matrice $n \times n$, ortogonală atunci

- Vectorii coloane sunt ortogonali;
- Vectorii linii sunt ortogonali;

Daca U, V sunt unitare(ortogonale în \mathbb{R}) atunci UV este unitara(ortogonală în \mathbb{R}).



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%C3%A3u_Sistemelor_Bucure%C5%83ti_2015_CS-UPB.jpg&oldid=1673062722) de la Wikipedia, licen%C3%A7ă CC-BY-SA 3.0

MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%C3%A3u_Sistemelor_Bucure%C5%83ti_2015_CS-UPB.jpg&oldid=1673062722) de la Wikipedia, licen%C3%A7ă CC-BY-SA 3.0

Transformari ortogonale

Householder

- Matricea H are valori proprii 1 si -1, defapt are o valoare proprie -1, prima si restu 1;
- $\det(H)=-1$;
- Orice x vector real poate fi transfromat de o matrice Householder reală în orice y vector real, cu $\|x\|=\|y\|$ (normă euclidiană);



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%C3%A3u_Sistemelor_Bucure%C5%83ti_2015_CS-UPB.jpg&oldid=1673062722) de la Wikipedia, licen%C3%A7ă CC-BY-SA 3.0

MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%C3%A3u_Sistemelor_Bucure%C5%83ti_2015_CS-UPB.jpg&oldid=1673062722) de la Wikipedia, licen%C3%A7ă CC-BY-SA 3.0

Transformari ortogonale

Daca H este o matrice $n \times n$, ortogonală atunci

- $H^{-1}=H^t$;
- $\det(H)=+1$;
- $\|Hx\|_2 = \|x\|_2$ - conservă normă euclidiană a unui vector;
- $\|H\|_2 = 1$ - normă euclidiană subordonată 1;
- $\|H \cdot A\|_2 = \|A\|_2$ - conservă normă euclidiană a unei matrici;



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%C3%A3u_Sistemelor_Bucure%C5%83ti_2015_CS-UPB.jpg&oldid=1673062722) de la Wikipedia, licen%C3%A7ă CC-BY-SA 3.0

Transformari ortogonale

Householder (Alston Scott Householder (1904 -1993) matematician american)

- Fie w din \mathbb{R}^n astfel încât $w^t w = 1$, matricea $n \times n$ $H = I_n - 2ww^t$ se numește transformare Householder (matrice Householder).
- Pentru a evita condiția $w^t w = 1$ se poate alege $H = I_n - 2ww^t / w^t w$.
- Matricea H este simetrică și ortogonală (unitara și hermitică).



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%C3%A3u_Sistemelor_Bucure%C5%83ti_2015_CS-UPB.jpg&oldid=1673062722) de la Wikipedia, licen%C3%A7ă CC-BY-SA 3.0

Transformari ortogonale

Householder

- Ne propunem să aducem o matrice simetrică A la o formă triagonală.
- Metoda constă la fiecare pas în a găsi matricea H_k care să facă zero sub a două diagonale.
- Initial se caută H_1 astfel încât matricea $A_2 = H_1 A H_1$ să aibă zero în prima coloană din linia 3, și pentru că înmulțim cu H_1 la dreapta se obține 0 pe prima linie de la coloana 3.



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%C3%A3u_Sistemelor_Bucure%C5%83ti_2015_CS-UPB.jpg&oldid=1673062722) de la Wikipedia, licen%C3%A7ă CC-BY-SA 3.0

Transformari ortogonale

Householder

$$\alpha = -\operatorname{sgn}(a_{k+1,k}^{(k)}) \left(\sum_{j=k+1}^n (a_{jk}^{(k)})^2 \right)^{1/2}, \text{ pentru } k=1,2,\dots,n-2$$

$$r = \left(\frac{1}{2}\alpha^2 - \frac{1}{2}\alpha a_{k+1,k}^{(k)} \right)^{1/2},$$

$$w_1^{(k)} = w_2^{(k)} = \dots = w_k^{(k)} = 0,$$

$$w_{k+1}^{(k)} = \frac{a_{k+1,k}^{(k)} - \alpha}{2r},$$

$$w_j^{(k)} = \frac{a_{jk}^{(k)}}{2r}, \quad j = k+2, k+3, \dots, n,$$

$$P^{(k)} = I - 2w^{(k)} \cdot (w^{(k)})^T, \quad A^{(k+1)} = P^{(k)} A^{(k)} P^{(k)}$$



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%2C_UB_2015_2016.jpg)



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%2C_UB_2015_2016.jpg)

Transformari ortogonale

Householder

Cand se doreste transformarea unei matrici intr-o matrice triunghiular superioara.

Fie A o matrice $m \times n$, atunci exista H $m \times m$ ortogonală astfel incat $HA=R$, unde R este o matrice $m \times n$ superior triunghiulară.



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%2C_UB_2015_2016.jpg)



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%2C_UB_2015_2016.jpg)

Transformari ortogonale

Exemplu Householder(triunghiular):

Alegeti ce vreti voi!



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%2C_UB_2015_2016.jpg)



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%2C_UB_2015_2016.jpg)

Transformari ortogonale

Householder

$$A^{(k+1)} = \begin{bmatrix} a_{11}^{(k+1)} & a_{12}^{(k+1)} & 0 & \cdots & 0 \\ a_{21}^{(k+1)} & a_{22}^{(k+1)} & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & a_{k-1,k}^{(k+1)} & a_{k+1,k+1}^{(k+1)} & \cdots & a_{k+1,k+2}^{(k+1)} & \cdots & a_{k+1,n}^{(k+1)} \\ 0 & \cdots & 0 & \ddots & a_{n,k+1}^{(k+1)} & \cdots & a_{n,k+2}^{(k+1)} & \cdots & a_{nn}^{(k+1)} \end{bmatrix}.$$

se continua pana cand $A=A_{n-1}$ devine triunghiulară

$$A^{(n-1)} = P^{(n-2)} P^{(n-3)} \cdots P^{(1)} A P^{(1)} \cdots P^{(n-3)} P^{(n-2)}$$



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%2C_UB_2015_2016.jpg)



Transformari ortogonale

Householder

$$H_p = I_m - 2 \frac{\mathbf{v}_p \mathbf{v}_p^T}{\mathbf{v}_p^T \mathbf{v}_p}$$

$$\mathbf{v}_p = [0 \dots 0 \mathbf{v}_{pp} \dots \mathbf{v}_{mp}]^T$$

$$\sigma_p = \operatorname{sign}(a_{pp}) \sqrt{\sum_{i=p}^m a_{ip}^2}$$

$$\mathbf{v}_{pp} = \mathbf{a}_{pp} + \sigma_p, \quad \mathbf{v}_{ip} = \mathbf{a}_{ip}, \quad i > p.$$

$$Ap+1=H_p A p, \quad A1=A;$$

$$HA=H_n \dots H_2 H_1 A;$$



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%2C_UB_2015_2016.jpg)



Transformari ortogonale

Givens (James Wallace Givens(1910-1993))

- O matrice G care difera de matricea unitate in cel mult patru elemente, care sunt de forma $g_{ii}=g_{jj}=\cos\theta$ si $g_{ij}=-g_{ji}=\sin\theta$, pentru un anume θ si $i \neq j$, se numeste matrice de rotatie.
- GA difera de A doar pe randurile i si j ;
- AG difera de A doar pe coloanele i si j ;
- Orice matrice de rotatie G este ortogonală;



MN Primavara 2015 CS-UPB

© Craiovea131 (https://commons.wikimedia.org/w/index.php?title=File:Logo_Facultatea_de_Matematica_si_Comp%2C_UB_2015_2016.jpg)



Transformari ortogonale

Givens

Pentru $i < j$,

$$G_{i,j} = G(\theta, i, j) =$$

$$\begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & \cos \theta & & -\sin \theta \\ & & & & 1 & \\ & & & & & \ddots \\ & & & & & & 1 \\ & & & & & & & \ddots \\ & & & & & & & & 1 \end{bmatrix}$$



MN Primavara 2015 CS-UPB

© Craiovea131 (julie) CC-BY-SA 4.0 International license



ETHOS



MN Primavara 2015 CS-UPB

© Craiovea131 (julie) CC-BY-SA 4.0 International license

Transformari ortogonale

Givens

Cand se doreste ca prin transformari Givens sa obtinem o matrice superior triunghiulara, atunci alegerea lui θ se face astfel incat $GA_{ji}=0$, $i \neq j$, adica din $ga_{ji}=a_{ii}\sin \theta + a_{jj}\cos \theta = 0$, de unde rezulta $\tan \theta = -a_{ii}/a_{jj}$, $\rho = \sqrt{a_{ii}^2 + a_{jj}^2}$ iar $\cos \theta = a_{ii}/\rho$ si $\sin \theta = -a_{jj}/\rho$



MN Primavara 2015 CS-UPB

© Craiovea131 (julie) CC-BY-SA 4.0 International license



ETHOS



MN Primavara 2015 CS-UPB

© Craiovea131 (julie) CC-BY-SA 4.0 International license

Transformari ortogonale

Exemplu Givens:

Alegeti voi!



MN Primavara 2015 CS-UPB

© Craiovea131 (julie) CC-BY-SA 4.0 International license



ETHOS



MN Primavara 2015 CS-UPB

© Craiovea131 (julie) CC-BY-SA 4.0 International license

Transformari ortogonale

Givens

- Matricea Givens, G se mai numeste matrice de rotatie plana.
- G_{ij} este ortogonală pentru orice pereche $i < j$ si orice θ din $[0, 2\pi]$.
- $G(\theta, i, j)^{-1} = G(-\theta, i, j)$.

Transformari ortogonale

Factorizari QR Householder & Givens

Daca inmultim cu inversa lui H , respectiv inversa lui G in stanga, identitatile vor deveni

$$A = (H_n \dots H_2 H_1)^{-1} R_H \text{ sau } A = (G_n \dots G_2 G_1)^{-1} R_G$$

Echivalent cu

$$A = (H_1)^{-T} (H_2)^{-T} \dots (H_n)^{-T} R_H \text{ sau}$$

$$A = (G_1)^{-T} (G_2)^{-T} \dots (G_n)^{-T} R_G$$

Si daca notam produsul inverselor transformatorilor corespunzator, obtinem



MN Primavara 2015 CS-UPB
© Craiovea1311 (julie) | Descarcă imaginea originală din Arhivele Openverse | Wikipedie în limba română | MN Primavara 2015 CS-UPB



MN Primavara 2015 CS-UPB
© Craiovea1311 (julie) | Descarcă imaginea originală din Arhivele Openverse | Wikipedie în limba română | MN Primavara 2015 CS-UPB



Transformari ortogonale

Householder despre Givens(o nota interesanta)

Unitary Triangularization of a Nonsymmetric Matrix*

ALSTON S. HOUSEHOLDER
Oak Ridge National Laboratory, Oak Ridge, Tennessee

A method for the inversion of a nonsymmetric matrix, due to J. W. Givens, has been in use at Oak Ridge National Laboratory and has proved to be highly stable numerically but to require a rather large number of arithmetic operations, including a total of $n(n - 1)/2$ square roots. Strictly, the method achieves the triangularization of the matrix, after which any standard method may be employed for inverting the triangle. The triangular form is brought about by means of a sequence of $n(n - 1)/2$ plane rotations, whose product is an orthogonal matrix. Each rotation requires the extraction of a square root. The advantage in using the method lies in the fact that an orthogonal matrix is perfectly conditioned. Hence the condition of the matrix cannot deteriorate through successive transformations. In fact, if one defines the condition number of a matrix A to be [1]

$$\gamma(A) = \|A\| \|A^{-1}\|,$$



MN Primavara 2015 CS-UPB
© Craiovea1311 (julie) | Descarcă imaginea originală din Arhivele Openverse | Wikipedie în limba română | MN Primavara 2015 CS-UPB



MN Primavara 2015 CS-UPB
© Craiovea1311 (julie) | Descarcă imaginea originală din Arhivele Openverse | Wikipedie în limba română | MN Primavara 2015 CS-UPB



Transformari ortogonale

Householder despre Givens(o nota interesanta)

If the matrix

$$A = (a_1, a_2, \dots, a_n)$$

is scaled at the outset so that $\|a_i\| \leq 1$ for every i , then all elements remain within range throughout the triangularization, since a unitary transformation leaves the Euclidean norm invariant. Hence no scaling problems arise in the actual triangularization. Moreover, when R^{-1} is formed, if this is similarly scaled no further scaling is required in multiplying by the matrices U_i to form A^{-1} . Only in forming R^{-1} itself may intermediate scaling be required.



MN Primavara 2015 CS-UPB
© Craiovea1311 (julie) | Descarcă imaginea originală din Arhivele Openverse | Wikipedie în limba română | MN Primavara 2015 CS-UPB



MN Primavara 2015 CS-UPB
© Craiovea1311 (julie) | Descarcă imaginea originală din Arhivele Openverse | Wikipedie în limba română | MN Primavara 2015 CS-UPB



Transformari ortogonale

Factorizari QR Householder & Givens

$$A = Q_H R_H \text{ sau } A = Q_G R_G$$

unde Q sunt matrici ortogonale, iar R matrici superioare triunghiulare. Deci transformarile ortogonale pot fi private si ca factorizari QR.



MN Primavara 2015 CS-UPB
© Craiovea1311 (julie) | Descarcă imaginea originală din Arhivele Openverse | Wikipedie în limba română | MN Primavara 2015 CS-UPB



Transformari ortogonale

Householder despre Givens(o nota interesanta)

The purpose of the present note is to point out that the same result can be obtained with fewer arithmetic operations, and, in particular, for inverting a square matrix of order n , at most $2(n - 1)$ square roots are required, instead of $n(n - 1)/2$. For $n > 4$, this is a saving of $(n - 4)(n - 1)/4$ square roots.

After $n - 1$ steps, at most, the matrix A is triangularized:

$$U = U_{n-1} U_{n-2} \cdots U_1, \quad UA = R.$$



MN Primavara 2015 CS-UPB
© Craiovea1311 (julie) | Descarcă imaginea originală din Arhivele Openverse | Wikipedie în limba română | MN Primavara 2015 CS-UPB



Polinoame ortogonale

Daca vi este un sir de vectori linear independenti atunci exista un sir de vectori ui ortonormati, cu $\text{span}\{vi\} = \text{span}\{ui\}$ (construiti de exemplu cu GS).

La fel se intampla si la polinoame, inlocuim sirul vi cu polinoamele $\{1, x, x^2, \dots, x^n\}$ din care putem obtine un sir de polinoame ortogonale $\{p_0, p_1, \dots, p_n\}$.

Cand vorbim de ortogonalitate ne referim la un anumit produs scalar.

O familie de polinoame ortogonale se defineste in mod unic in raport cu intervalul de definitie $[a, b]$ si o functie pondere $w(x)$.



MN Primavara 2015 CS-UPB
© Craiovea1311 (julie) | Descarcă imaginea originală din Arhivele Openverse | Wikipedie în limba română | MN Primavara 2015 CS-UPB



Polinoame ortogonale

Cand vorbim de ortogonalitate ne referim la un anumit produs scalar. De obicei, pentru functii se considera urmatoarele produse scalare

$$\begin{aligned}\langle g, h \rangle &= \int_a^b g(x)h(x)w(x) dx, \\ \langle g, h \rangle &= \sum_{i=1}^n g(x_i)h(x_i)w(x_i),\end{aligned}$$



Polinoame ortogonale

Anumite produse scalare satisfac relatia de simetrie, adica $\langle xf, g \rangle = \langle f, xg \rangle$ si in acest caz sirul polinoamelor ortogonale se deduce din relatiile

$$\begin{aligned}p_0(x) &= 1, & p_1(x) &= x - \alpha_0 \\ p_{k+1}(x) &= (x - \alpha_k)p_k(x) - \beta_k p_{k-1}(x), & k &= 1 : n-1 \\ \alpha_k &= \frac{\langle xp_k, p_k \rangle}{\|p_k\|^2}, & k &= 0 : n-1, \\ \beta_k &= \frac{\|p_k\|^2}{\|p_{k-1}\|^2}, & k &= 1 : n-1.\end{aligned}$$



Polinoame ortogonale

Pentru un polinom

$$p_n(x) = a_{nn}x^n + a_{n-1,n}x^{n-1} + a_{0n}$$

care este ortonormat, avem relatia

$$\frac{a_{n,n}}{a_{n+1,n+1}} \cdot p_{n+1}(x) + \left(\frac{a_{n-1,n}}{a_{n,n}} - \frac{a_{n,n+1}}{a_{n+1,n+1}} - x \right) \cdot p_n(x) + \frac{a_{n-1,n-1}}{a_{n,n}} \cdot p_{n-1}(x) = 0$$

iar daca este doar ortogonal avem relatia

$$\frac{a_{n,n}}{a_{n+1,n+1}} \cdot p_{n+1}(x) + \left(\frac{a_{n-1,n}}{a_{n,n}} - \frac{a_{n,n+1}}{a_{n+1,n+1}} - x \right) \cdot p_n(x) + \frac{a_{n-1,n-1}}{a_{n,n}} \cdot \frac{\|p_n\|^2}{\|p_{n-1}\|^2} p_{n-1}(x) = 0$$



Polinoame ortogonale

Polinoame des utilizate

Cebășev	$T_{n+1} - 2xT_n + T_{n-1} = 0,$	$T_0 = 1, \quad T_1 = x$
Legendre	$(n+1)L_{n+1} - (2n+1)xL_n + nL_{n-1} = 0,$	$L_0 = 1, \quad L_1 = x$
Laguerre	$G_{n+1} - (2n+1-x)G_n + n^2 G_{n-1} = 0,$	$G_0 = 1, \quad G_1 = 1-x$
Hermite	$H_{n+1} - 2xH_n + 2nH_{n-1} = 0,$	$H_0 = 1, \quad H_1 = 2x$

Pafnuty Lvovich Chebyshev ((1821-1894) mat. rus)

Adrien-Marie Legendre ((1752-1833) mat. francez)

Edmond Nicolas Laguerre ((1834-1886) mat. francez)

Charles Hermite ((1822-1901) mat. francez)



Polinoame ortogonale

Proprietati Polinoame ortogonale

- Radacini reale distincte situate in $[a,b]$;
- Mentin prop. de ortogonalitate cu orice polinom(neortogonal) de grad mai mic;
- Radacinile polinomului P_n determina intervalele de separare a zerourilor pentru polinomul P_{n+1} ;



Polinoame ortogonale

Proprietati Polinoame ortogonale

- Minimul integralei $\int_a^b w(x)q_n^2(x)dx$ pentru orice polinom de grad n , este realizat de polinomul ortogonal $P_n(x)$, definit in mod unic in raport cu intervalui $[a,b]$ si ponderea $w(x)$;



Bibliografie Transformari ortogonale / Polinoame ortogonale

- Richard L. Burden, J. Douglas Faires, Numerical Analysis, 9th edition, Brooks/Cole, Cengage Learning, 2011;
- Roger A. Horn, Charles R. Johnson, Matrix Analysis, 2th edition, Cambridge, 2013;
- Valeriu Iorga, Boris Jora, *Metode Numerică*, Editura Albastră, 2005;



Transformări ortogonale: Householder și Givens. Algoritmul Gram-Schmidt. Polinoame ortogonale

Noțiuni teoretice

Vectori ortogonali. Matrice ortogonală

Fie vectorii coloană $x, y \in R^n$ de forma
 $x = [x_1 \ x_2 \ \dots \ x_n]^T$ și $y = [y_1 \ y_2 \ \dots \ y_n]^T$. Definim:

- *produsul scalar*: $\langle x, y \rangle = \sum_{i=1}^n x_i y_i = y^T x$;
- *norma euclidiană*: $\|x\|_2 = \sqrt{\langle x, x \rangle} = \sqrt{x^T x} = \left(\sum_{i=1}^n x_i^2 \right)^{1/2}$.

Spunem că:

- vectorii x și y sunt *ortogonali* dacă $\langle x, y \rangle = 0$;
- vectorii x și y sunt *ortonormați* dacă sunt ortogonali, $\|x\|_2 = 1$ și $\|y\|_2 = 1$.

O matrice $H \in R^{n \times n}$ este *ortogonală* dacă are coloanele vectori ortonormați. Mai mult, o matrice ortogonală are proprietățile următoare:

1. $H^T H = H H^T = I_n$;
2. $H^{-1} = H^T$;
3. $\|Hx\|_2 = \|x\|_2$;
4. $\|H\|_2 = 1$;

5. $\|HA\|_2 = \|A\|_2$;
6. $\det(H) = \pm 1$.

Transformarea Householder

Metoda propusă inițial de Alston Scott Householder este folosită pentru a transforma o matrice $A \in R^{n \times n}$ simetrică într-o matrice tridiagonală cu aceleași valori proprii. Prezentarea generală a metodei poate fi gasită la adresa ¹.

Pentru o matrice $A \in R^{m \times n}$, definim transformarea Householder $R = HA$ folosind reflectoari elementari Householder H_p , astfel încât:

$$H = H_{min(m-1,n)} \dots H_p \dots H_2 H_1, \quad A = H^T R.$$

Un reflector elementar Householder H_p este dat de relația:

$$H_p = I_m - 2 \frac{v_p v_p^T}{v_p^T v_p}$$

unde:

- $v_p = [0 \ 0 \ \dots \ v_{pp} \ \dots \ v_{mp}]^T$ se numește vector Householder;
- $v_{pp} = a_{pp} + \sigma_p$;
- $v_{ip} = a_{ip}$, $\forall i > p$;
- $a_p = [a_{1p} \ a_{2p} \ \dots \ a_{pp} \ \dots \ a_{mp}]^T$ este coloana p din matricea A ;
- $\sigma_p = \text{sign}(a_{pp}) \sqrt{\sum_{i=p}^m a_{ip}^2}$;
- $\beta_p = \sigma_p v_{pp}$.

Folosind un reflector elementar Householder putem aduce o matrice la forma superior triunghiulară astfel:

Datorită formei reflectoarelor elementare Householder, înmulțirea $A = H_p A$ se efectuează astfel:

- Coloanele $1 : p - 1$ din matricea A rămân neschimbate;
- Coloana p din matricea A se modifică astfel:
 - elementele de pe liniile $1 : p - 1$ rămân neschimbate;
 - $a_{pp} = -\sigma_p$;

¹<http://mathfaculty.fullerton.edu/mathews/n2003/HouseholderMod.html>

Algorithm 1 Transformarea unei matrice la forma superior triunghiulară

```

1: procedure [Q, R] = ST(A)
2:   [m,n] = size(A);
3:   H = In;
4:   for p = 1 : min(m - 1, n) do
5:     Hp = In - 2  $\frac{v_p v_p^T}{v_p^T v_p}$ ;
6:     A = Hp A;
7:     H = Hp H;
8:   end for
9:   Q = HT;
10:  R = A;
11: end procedure

```

- $a_{ip} = 0, \forall i > p;$
- Coloanele $j = p + 1 : n$ din matricea A se modifică astfel:
 - elementele de pe liniile $1 : p - 1$ rămân neschimbate;
 - $a_{ij} = a_{ij} - \tau_j \cdot v_{ip}, \forall i \geq p, \tau_j = \frac{\sum_{i=p}^m v_{ip} \cdot a_{ij}}{\beta_p}.$

Transformarea Givens

Metoda Givens este folosită pentru a descompune o matrice $A \in R^{m \times n}$ astfel:

$$A = G^T R$$

unde:

$$G = G_{n-1,m} G_{n-2,m} G_{n-2,m-1} \dots G_{1n} \dots G_{13} G_{12}, \quad R = GA.$$

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

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

Pentru a determina matricea de rotație Givens G_{kl} , vom folosi relațiile:

$$\rho = \sqrt{A(k, k)^2 + A(l, k)^2}$$

$$s = \sin \theta = -\frac{A(l, k)}{\rho}$$

$$c = \cos \theta = \frac{A(k, k)}{\rho}$$

Datorită formei matricelor de rotație Givens, înmulțirea $G_{kl}x$, unde x este vector coloană, se efectuează astfel:

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

Algoritmul Gram-Schmidt

Vom considera relația $A = QR$, adică:

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

cu necunoscutele q_i și r_{ij} , ($i \leq j$), unde:

- a_i este coloana i din matricea A ;
- q_i este coloana i din matricea Q ;
- $r_{ij}, i \leq j$, reprezintă elementele din matricea superior triunghiulară R .

Algoritmul Gram-Schmidt este schițat în continuare:

Pentru $j = 1 : n$

$$\begin{aligned} r_{ij} &= q_i^T a_j, \quad i = 1 : j - 1; \\ aux &= a_j - \sum_{i=1}^{j-1} r_{ij} q_i; \\ r_{jj} &= \|aux\|_2; \\ q_j &= \frac{aux}{r_{jj}}. \end{aligned}$$

Algoritmul Gram-Schmidt modificat

Algoritmul Gram-Schmidt clasic prezintă o stabilitate numerică slabă. Acest algoritm poate fi îmbunătățit folosind următoarea variantă:

Pentru $i = 1 : n$

$$\begin{aligned} r_{ii} &= \|a_i\|_2; \\ q_i &= \frac{a_i}{r_{ii}}; \end{aligned}$$

Pentru $j = i + 1 : n$

$$\begin{aligned} r_{ij} &= q_i^T a_j; \\ a_j &= a_j - q_i r_{ij}. \end{aligned}$$

Polinoame ortogonale

Un polinom ortogonal este definit prin:

- relația de recurență;
- cazurile de bază ale relației de recurență;
- intervalul $/(a, b)/$ pe care este definit;
- funcția pondere $w(x)$, folosită la calcularea produsului scalar.

Exemple de polinoame ortogonale:

- *Cebâșev*:

$$\begin{aligned} T_{n+1} - 2xT_n + T_{n-1} &= 0, \quad T_0 = 1, \quad T_1 = x; \\ (-1, 1); \quad w(x) &= \frac{1}{\sqrt{1-x^2}}; \end{aligned}$$

- Legendre:

$$(n+1)L_{n+1} - (2n+1)xL_n + nL_{n-1} = 0, \quad L_0 = 1, \quad L_1 = x;$$

$$[-1, 1]; \quad w(x) = 1;$$

- Laguerre:

$$G_{n+1} - (2n+1-x)G_n + n^2G_{n-1} = 0, \quad G_0 = 1, \quad G_1 = 1-x;$$

$$[0, \infty); \quad w(x) = e^{-x};$$

- Hermite:

$$H_{n+1} - 2xH_n + 2nH_{n-1} = 0, \quad H_0 = 1, \quad H_1 = 2x;$$

$$(-\infty, \infty); \quad w(x) = e^{-x^2}$$

Proprietățile polinoamelor ortogonale sunt:

1. Orice polinom ortogonal are radacinile în intervalul $[(a, b)]$, reale și distințe;
2. Orice polinom ortogonal este ortogonal cu orice polinom de grad mai mic decât el.

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

- $\|p_i\| = 1, \quad \forall i$; și
- $\langle p_i, p_j \rangle = 0, \quad \forall i \neq j$, unde $\langle p_i, p_j \rangle = \int_a^b p_i(x)p_j(x)w(x)dx$.

Operații utile cu polinoame în Octave:

```

1 function test_poly
2     %% Coeficientii polinomului p
3     p = [ 2 1 -1]
4     %% Afisarea polinomului
5     polyout(p, 'x')
6     %% Coeficientii polinomului q
7     q = [ 1 2]
8     %% Afisarea polinomului
9     polyout(q, 'x')
10    %% Produsul dintre p si q
11    r = conv(p,q)

```

```

12 %% Afisarea rezultatului
13 polyout(r, 'x')
14 endfunction

```

Listing 1: Polinoame în Octave.

Probleme rezolvate

Problema 1

Dacă H_1, H_2 sunt matrici ortogonale, arătați că produsul H_1H_2 este o matrice ortogonală.

Soluție:

Din proprietățile matricelor ortogonale, avem: H ortogonală $\Rightarrow HH^T = I_n$ și

$$\begin{cases} H_1H_1^T = I_n \\ H_2H_2^T = I_n \end{cases} \Rightarrow (H_1H_2)(H_1H_2)^T = H_1H_2H_2^TH_1^T = I_n$$

Problema 2

Fie un reflector elementar Householder

$$H = I_n - \frac{2uu^T}{\|u\|^2}$$

a) Arătați că $H^T = H$;

b) Calculați Hu .

Soluție:

a) $H^T = \left(I_n - \frac{2uu^T}{\|u\|^2} \right)^T = I_n - \frac{2uu^T}{\|u\|^2} = H$

b) $Hu = \left(I_n - \frac{2uu^T}{\|u\|^2} \right) u = u - \frac{2uu^T}{\|u\|^2} u = u - 2u = -u$

Problema 3

Determinați factorizarea QR folosind transformarea Householder pentru matricea:

$$A = \begin{bmatrix} 2 & 4 & 5 \\ 1 & -1 & 1 \\ 2 & 1 & -1 \end{bmatrix}.$$

Soluție:

Prima iterare:

$$a_1 = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}, \quad \sigma_1 = \|a_1\|_2 = 3 \quad u_1 = \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix} \quad \|u_1\|_2^2 = 6;$$

$$H_1 = I_3 - \frac{2u_1 u_1^T}{\|u_1\|_2^2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \frac{1}{3} \begin{bmatrix} 1 & -1 & -2 \\ -1 & 1 & 2 \\ -2 & 2 & 4 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & -2 \\ 2 & -2 & -1 \end{bmatrix}$$

$$A_2 = H_1 A_1 = \frac{1}{3} \begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & -2 \\ 2 & -2 & -1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 4 & 5 \\ 1 & -1 & 1 \\ 2 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 3 & 3 & 3 \\ 0 & 0 & 3 \\ 0 & 3 & 3 \end{bmatrix}$$

A doua iterare:

$$\bar{a}_2 = \begin{bmatrix} 0 \\ 3 \\ 3 \end{bmatrix} \quad \sigma_2 = \|\bar{a}_2\|_2 = 3 \quad u_2 = \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} \quad \|u_2\|_2^2 = 2;$$

$$H_2 = I_3 - \frac{2u_2 u_2^T}{\|u_2\|_2^2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$A_3 = H_2 A_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 3 & 3 & 3 \\ 0 & 0 & 3 \\ 0 & 3 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 3 & 3 \\ 0 & 3 & 3 \\ 0 & 0 & 3 \end{bmatrix}$$

Atunci:

$$R = A_3 = \begin{bmatrix} 3 & 3 & 3 \\ 0 & 3 & 3 \\ 0 & 0 & 3 \end{bmatrix}$$

$$Q = (H_2 H_1)^T = H_1 H_2 = \frac{1}{3} \begin{bmatrix} 2 & 2 & 1 \\ 1 & -2 & 2 \\ 2 & -1 & -2 \end{bmatrix}$$

Problema 4

Să se scrie un program OCTAVE care să implementeze transformarea Givens. Programul primește ca date de intrare: A - matricea sistemului; b - vectorul termenilor liberi. Rezultatele programului vor fi: Q - matricea factor ortogonală; R - matricea factor superior triunghiular; b - vectorul modificat al termenilor liberi.

```

1 function [Q R b] = givens(A, b)
2 [m n] = size(A);
3 Q = eye(m);
4
5 for k = 1 : min(m-1, n)
6   for l = k + 1 : m
7     r = sqrt(A(k,k)^2 + A(l,k)^2);
8     c = A(k,k)/r;
9     s = -A(l,k)/r;
10
11    aux = [c -s; s c]*[A(k,k:n); A(l,k:n)];
12    A(k,k:n) = aux(1, :);
13    A(l,k:n) = aux(2, :);
14
15    aux = [c -s; s c]*[b(k); b(l)];
16    b(k) = aux(1);
17    b(l) = aux(2);
18
19    aux = [c -s; s c]*[Q(k,1:m); Q(l,1:m)];
20    Q(k,1:m) = aux(1, :);
21    Q(l,1:m) = aux(2, :);
22  end
23 end

```

Transformări ortogonale: Householder și Givens. Algoritmul Gram-Schmidt.
Polinoame ortogonale

```
24
25   Q = Q';
26   R = A;
27 end
```

Listing 2: Transformarea Givens.

Problema 5

Să se scrie un program OCTAVE care să implementeze algoritmul Gram-Schmidt. Programul primește ca date de intrare: A - matrice. Rezultatele programului vor fi: Q - matricea factor ortogonală; R - matricea factor superior triunghiular.

Soluție:

```
1 function [Q, R] = Gram_Schmidt (A)
2 [m n] = size (A);
3 Q = zeros (m, n);
4 R = zeros (n);

5
6 for j = 1 : n
7   for i = 1 : j-1
8     R(i,j) = Q(:,i)' * A(:,j);
9   endfor

10
11 s = zeros (m, 1);
12 for i = 1 : j-1
13   s = s + R(i,j) * Q(:,i);
14 endfor
15 %% Echivalent pentru instructiunea for de mai sus:
16 %% s = Q(:, 1:j-1) * R(1:j-1, j);

17 aux = A(:,j) - s;
18
19 R(j,j) = norm (aux, 2);
20 Q(:,j) = aux/R(j,j);
21 endfor
22 endfunction
```

Listing 3: Algoritmul Gram-Schmidt.

Problema 6

Să se scrie un program OCTAVE pentru calculul coeficienților unui polinom ortogonal de grad n . Selecția polinomului se face printr-un parametru sir de caractere care poate avea valorile: ‘cebasev’, ‘legendre’, ‘laguerre’ sau ‘hermite’.

Soluție:

```
1 function P = poliOrtogonal(nume_polinom, n)
2
3 if strcmp(nume_polinom, 'legendre')
4     P0 = [1];
5     P1 = [1 0];
6     C0 = @ (n) ([-n/(n+1)]);
7     C1 = @ (n) ([ (2*n+1)/(n+1) 0]);
8 endif
9
10 if strcmp(nume_polinom, 'cebasev')
11     P0 = [1];
12     P1 = [1 0];
13     C0 = @ (n) ([-1]);
14     C1 = @ (n) ([2 0]);
15 endif
16
17 if strcmp(nume_polinom, 'laguerre')
18     P0 = [1];
19     P1 = [-1 1];
20     C0 = @ (n) ([-n^2]);
21     C1 = @ (n) ([-1 2*n+1]);
22 endif
23
24 if strcmp(nume_polinom, 'hermite')
25     P0 = [1];
26     P1 = [2 0];
27     C0 = @ (n) ([-2*n]);
28     C1 = @ (n) ([2 0]);
29 endif
30
31 P = poliOrtogonalGeneral(n, P0, P1, C0, C1);
32 endfunction
```

Transformări ortogonale: Householder și Givens. Algoritmul Gram-Schmidt.
Polinoame ortogonale

```
1 function P = poliOrtogonalGeneral(n, P0, P1, C1, C2)
2     %Pn+1 = C1(n)*Pn + C2(n)*Pn-1
3     %Valori initiale: P0, P1
4     %Iesire: Pn
5
6     %Cazuri de baza
7     if n == 0
8         P = P0;
9     endif
10
11    if n == 1
12        P = P1;
13    endif
14
15    %Ultimii 2 termeni ai recurentei sunt pastrati in P0 si P1
16    for i=2:n
17        Paux1 = conv(P0, C1(i));
18        Paux2 = conv(P1, C2(i));
19
20        m = length(Paux1);
21        n = length(Paux2);
22        r = max(m, n);
23
24        Paux1(r-m+1:r) = Paux1(1:m);
25        Paux1(1:r-m) = zeros(1,r-m);
26
27        Paux2(r-n+1:r) = Paux2(1:n);
28        Paux2(1:r-n) = zeros(1,r-n);
29
30        Paux = Paux1 + Paux2;
31
32        P0 = P1;
33        P1 = Paux;
34    endfor
35
36    P = P1;
37 endfunction
```

Probleme propuse

Problema 1

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

- Să se dea exemplu de doi vectori ortonormați;
- Să se calculeze $\|x\|_2$;
- Se formează matricea $H = I_n - xx^T$. Să se calculeze Hu, Hv și $\|H\|_2$;
- Dacă $A = uv^T$, calculați $B = H^{-n}AH^n$.

Problema 2

Implementați transformarea Householder pentru o matrice A , astfel:

- Implementați o funcție care primește un vector x , un index p și calculează parametrii σ, v_p, β definiți mai sus:

```
function [vp, sigma, beta] = GetHSReflector(x, p)
```

- Implementați o funcție care primește un vector x , un index p , parametrul $sigma$ și calculează transformarea Householder aplicată asupra vectorului, considerând că acest vector a fost folosit la calculul parametrilor (coloana p din A).

```
function x = ApplyHSToPColumn(x, p, sigma)
```

- Implementați o funcție care primește un vector oarecare x , un index p , vectorul Householder v_p parametrul $beta$ și calculează transformarea Householder aplicată asupra vectorului (coloanele $p+1:n$ din A).

```
function x = ApplyHSToRandomColumn(x, vp, p, beta)
```

- Implementați funcția

```
function [Q, R] = Householder(A),
```

folosind funcțiile definite mai sus.

Problema 3

Să se determine descompunerea QR pentru matricea $A = \begin{bmatrix} 3 & 1 & -2 \\ 1 & 3 & 1 \\ -2 & 1 & 3 \end{bmatrix}$ folosind transformarea Givens.

Problema 4

Să se scrie un program OCTAVE pentru a implementa algoritmul Gram-Schmidt modificat.

Metode Numerice @ CS-UPB

Echipa MN CS - 2015:

Profesorii:

- Pantelimon George Popescu - Seria CA;
- Florin Pop - Seria CB, CC;

Asistenti:

PG Popescu, Clementin Cercel, Sorin Ciolofan, Bogdan Tiganoia, Diana Udrea, Alexandru Tifrea, David Iancu, Bogdan Cristian Marchis, Radu Poenaru, Madalina Grosu, Andrei Vlad Postoaca, Radu Constantinescu, Roxana Istrate, Madalina Hristache, Adelina Vidovici;



MN Primavara 2015 CS-UPB
© Craiovea131 (julie) | Flickr | Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License



NUMERICAL METHODS

SISTEME de ECUATII LINEARE



MN Primavara 2015 CS-UPB
© Craiovea131 (julie) | Flickr | Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License



Sisteme de ecuatii lineare

Un sistem linear

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ &\vdots && \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n, \end{aligned}$$

se reprezinta matricial astfel $\mathbf{Ax} = \mathbf{b}$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$



MN Primavara 2015 CS-UPB
© Craiovea131 (julie) | Flickr | Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License



NUMERICAL METHODS

Metode

- Exakte** - sunt metodele care furnizeaza solutia exacta a sistemelor daca se negligeaza erorile de rotunjire;
- Aproximative/Iterative** - sunt metodele care construiesc un sir x_k convergent catre solutia exacta a sistemului, x ; (Jacobi, Gauss-Seidal)

Dintre metodele exacte ne ocupam de cele **Directe**.
 (Inversare matrici, Eliminare Gauss, Gauss-Jordan, Cholesky, Crout, Doolittle, Algoritmul Thomas)



MN Primavara 2015 CS-UPB
© Craiovea131 (julie) | Flickr | Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License



Metode Directe

- sunt metode care teoretic ne ofera solutia exacta a unui sistem de ecuatii intr-un numar finit de pasi. In practica solutia evident ca va fi contaminata de efectul erorii de rotunjire, implicata in aritmetica folosita;
- reduc sistemul prin transformari de echivalenta la un sistem particular care se rezolva prin mijloace elementare;
- se bazeaza pe urmatorul rezultat

Daca A si T sunt matrici nesingulare, atunci x este solutie pentru sistemul $\mathbf{Ax} = \mathbf{b}$ daca si numai daca x este solutie pentru sistemul $\mathbf{T}Ax = \mathbf{ Tb}$.



MN Primavara 2015 CS-UPB
© Craiovea131 (julie) | Flickr | Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License



Metode Directe

Regula lui Cramer 1750 (Gabriel Cramer (1704-1752), amintim si de Colin Maclaurin care a publicat ceva asemănător pentru cazuri speciale in 1748)

Solutia unui sistem $\mathbf{Ax} = \mathbf{b}$ este data de

$$x_j = \frac{\det(A^j)}{\det(A)} \quad (j = 1, \dots, n)$$

unde A^j este matricea obtinuta prin inlocuirea coloanei j din matricea A cu vectorul b .



MN Primavara 2015 CS-UPB
© Craiovea131 (julie) | Flickr | Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License



Metode Directe

Regula lui Cramer

Numerul de inmultiri si impartiri N necesar pentru calculul determinantilor folositi in aceasta metoda este $N=(n-1)(n+1)!$ Deci pentru un sistem relativ mic de ecuatii cu $n=10$, avem $N=360,000,000$ de operatii de inmultire si impartire, enorm!

Regula lui Cramer este extrem de ineficienta ca si toate metodele ce se bazeaza pe calculul determinantilor.



MN Primavara 2015 CS-UPB

© Craioveanu Mihai Iulian / CC BY-NC-SA 4.0 International License

Metode Directe

Metode bazate pe ELIMINARI

- Se folosesc urmatoarele trei operatii matriciale (cele trei operatii fundamentale)

 1. Orice rand (ecuatie) poate fi inmultita cu o constanta; - **scalare a $E_i \rightarrow E_i$**
 2. Ordinea randurilor (ecuatiilor) poate fi interschimbată; - **pivotare $E_j \leftrightarrow E_i$**
 3. Orice rand (ecuatie) poate fi inlocuită cu o combinatie ponderată, lineară între acel rand și orice alt rand; - **eliminare $E_i - aE_j \rightarrow E_i$**



MN Primavara 2015 CS-UPB

© Craioveanu Mihai Iulian / CC BY-NC-SA 4.0 International License

Metode Directe

Eliminare Gaussiana

- foloseste cele 3 op. scalare, pivotare, eliminare;
- se bazeaza pe urmatorul rezultat

Daca matricile $A = [a_{ij}]$, $1 \leq i, j \leq p$ sunt nesingulare pentru $p=1, \dots, n$, atunci exista T nesingulara, inferior triunghiulara, astfel incat matricea $TA = U$ este superior triunghiulara.

Matricea T se alege astfel $T = T_{n-1} * \dots * T_2 * T_1$, cu

$$T_p = I_n - t_p * e_p^T$$



MN Primavara 2015 CS-UPB

© Craioveanu Mihai Iulian / CC BY-NC-SA 4.0 International License

Metode Directe

Metode bazate pe ELIMINARI

- Procesul de eliminare se refera la faptul ca, sa spunem, x_1 este obtinut din E_1 , in functie de x_2, x_3, \dots, x_n si apoi este introdus in toate ecuatii ramase, in E_2, E_3, \dots, E_n . Procesul se repeta de $n-1$ ori pana se obtine o ecuatie doar in x_n , care se rezolva;
- Substitutia inapoi se referea la faptul ca in momentul in care il cunosc pe x_n atunci din ultima ecuatie, care implica termenii x_n si x_{n-1} , il pot obtine pe x_{n-1} samd;



MN Primavara 2015 CS-UPB

© Craioveanu Mihai Iulian / CC BY-NC-SA 4.0 International License

Metode Directe

Eliminare Gaussiana (Johann Carl Friedrich Gauss (30 April 1777 - 23 February 1855) - mat german - anecdota: suma 1-100)

- O tehnica similara eliminarii gaussiene a aparut prima oara in timpul dinastiei Han din China, in textul *9 capitole despre Arta Matematica*;
- Joseph Louis Lagrange (1736-1813) a descris o tehnica similara in 1778 pentru cazul in care valoarea fiecarei ecuatii este 0;



MN Primavara 2015 CS-UPB

© Craioveanu Mihai Iulian / CC BY-NC-SA 4.0 International License

Metode Directe

Eliminare Gaussiana

Unde I_n este matricea unitate, e_p este coloana p a lui I_n , iar t_p este un vector coloana, numit vector Gauss definit astfel

$$t_p = [0 \dots 0 \ t_{p+1,p} \dots t_{np}]^T$$

unde componentele necunoscute se deduc din impunerea conditiei ca aplicand transformarea T_p unui vector x sa ii lase identice primele p componente iar pe celelalte sa le faca 0, deci



MN Primavara 882015 CS-UPB

© Craioveanu Mihai Iulian / CC BY-NC-SA 4.0 International License

Metode Directe

Eliminare Gaussiană

$$\begin{aligned} T_p \mathbf{x} &= (\mathbf{I}_n - t_p e_p^T) \mathbf{x} = \mathbf{x} - t_p (e_p^T \mathbf{x}) = \mathbf{x} - t_p \mathbf{x}_p \\ (T_p \cdot \mathbf{x})_i &= x_i - t_{ip} \cdot x_p = \begin{cases} x_i, & i \leq p \\ x_i - t_{ip} x_p, & i > p \end{cases} \end{aligned}$$

impunand conditia ca pentru $i > p$ elementele sa fie 0 deducem

$$t_{ip} = \frac{x_i}{x_p}, \quad i = p + 1 : n$$



MN Primavara 2015 CS-UPB
© Craioveanu, I.S. (1971) Probleme de Algebra Lineara. Editura Tehnica, Bucuresti. ISBN 977-24-0045-0

Metode Directe

Eliminare Gaussiană

- Scenariul pentru aceasta metoda este acela in care consideram matricea sistemului, A patratica, ca vector x consideram coloana p a matricei A, iar ca vector y consideram pe rand coloanele j ale matricei A situate la dreapta coloanei p;
- Se aplica algoritmul pentru $p=1,\dots,n$;
- Se obtine, dupa aplicarea transformarii $T=T_{n-1}\dots T_{2T_1}$ asupra matricei A, o matrice superior triunghiulara, i.e. $TA=U$;



MN Primavara 2015 CS-UPB
© Craioveanu, I.S. (1971) Probleme de Algebra Lineara. Editura Tehnica, Bucuresti. ISBN 977-24-0045-0

Metode Directe

Pivotare totală - se fac aranjamente intre randuri si coloane (ecuatii si valori), inainte de fiecare pas, astfel incat elementul maxim de pe rand si coloana sa fie pus pe diagonală;

Pivotare parțială $\left(\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ 2 & 2 & 3 & 3 \end{array} \right) \rightarrow \left(\begin{array}{ccc|c} 2 & 2 & 3 & 1 \\ 1 & 1 & 2 & 2 \\ 1 & 1 & 1 & 3 \end{array} \right)$

Pivotare totală $\left(\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ 2 & 2 & 3 & 3 \end{array} \right) \rightarrow \left(\begin{array}{ccc|c} 2 & 2 & 3 & 1 \\ 1 & 1 & 2 & 2 \\ 1 & 1 & 1 & 3 \end{array} \right) \rightarrow \left(\begin{array}{ccc|c} 3 & 2 & 2 & 1 \\ 2 & 1 & 1 & 2 \\ 1 & 1 & 1 & 3 \end{array} \right)$



MN Primavara 2015 CS-UPB
© Craioveanu, I.S. (1971) Probleme de Algebra Lineara. Editura Tehnica, Bucuresti. ISBN 977-24-0045-0

Metode Directe

Eliminare Gaussiană

deci vectorul Gauss este

$$t_p = \begin{bmatrix} 0 & \dots & 0 & \frac{x_{p+1}}{x_p} & \dots & \frac{x_n}{x_p} \end{bmatrix}^T$$

Aplicand transformarea T_p unui vector oarecare

$$T_p \cdot \mathbf{y} = (\mathbf{I}_n - t_p e_p^T) \cdot \mathbf{y} = \mathbf{y} - t_p \cdot (e_p^T \cdot \mathbf{y}) = \mathbf{y} - t_p \cdot y_p$$

$$(T_p \cdot \mathbf{y})_i = \begin{cases} y_i, & i \leq p \\ y_i - \frac{x_i}{x_p} \cdot y_p, & i > p \end{cases}$$



MN Primavara 2015 CS-UPB
© Craioveanu, I.S. (1971) Probleme de Algebra Lineara. Editura Tehnica, Bucuresti. ISBN 977-24-0045-0

Metode Directe

Eliminare Gaussiană - Eliminare cu pivotare - Metoda de eliminare gaussiană pică dacă pe diagonala principală gaseste 0, de asemenea metoda poate produce chiar ea însăși zerouri pe diagonala principală și iarăși pică! Din acest motiv se practică pivotarea;

Pivotare parțială - se fac aranjamente între randuri (ecuații), înainte de fiecare pas, astfel încât elementul maxim de pe rand și coloana să fie pus pe diagonala;



MN Primavara 2015 CS-UPB
© Craioveanu, I.S. (1971) Probleme de Algebra Lineara. Editura Tehnica, Bucuresti. ISBN 977-24-0045-0

Metode Directe

Eliminare Gaussiană - Eliminare cu scalare - Metoda de eliminare gaussiană produce erori mari de rotunjire dacă ordinul coeficientilor dintr-o ecuație este semnificativ diferit. Din acest motiv se practică scalarea; Scalarea este un pas intermediu prin care se alege elementul pivot;

- se impart elementele din coloana p, la pasul p cu cel mai mare număr din linia respectivă și se alege ca pivot elementul al carui raport este mai mare;



MN Primavara 2015 CS-UPB
© Craioveanu, I.S. (1971) Probleme de Algebra Lineara. Editura Tehnica, Bucuresti. ISBN 977-24-0045-0

Metode Directe

Contorizarea operatiilor la eliminari

• Eliminare simpla:

$$\begin{aligned} * & \text{ sau } / (2n^3 + 3n^2 - 5n)/6; \\ + & \text{ sau } - (n^3 - n)/3; \end{aligned}$$

• Substitutie inapoi:

$$\begin{aligned} * & \text{ sau } / (n^2 + n)/2; \\ + & \text{ sau } - (n^2 - n)/2; \end{aligned}$$



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS Craciunescu - Rezultatul unei Aplicatii Geostatistice WGS-84-V1 = 6173000/72224m



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS Craciunescu - Rezultatul unei Aplicatii Geostatistice WGS-84-V1 = 6173000/72224m

Metode Directe

Factorizarea LU - Crout

$$\begin{aligned} l_{i1} &= a_{i1}; i = 1 : n \\ u_{1j} &= \frac{a_{1j}}{l_{11}}; j = 2 : n \\ l_{ij} &= a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}; i \geq j \\ u_{ij} &= \frac{1}{l_{ii}} * (a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}); i < j \end{aligned}$$

Factorizarea LU - Doolittle

$$\begin{aligned} u_{i1} &= a_{i1}; i = 1 : n \\ l_{i1} &= \frac{a_{11}}{u_{11}}; i = 2 : n \\ l_{ij} &= \frac{1}{u_{jj}} * (a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}); i > j \\ u_{ij} &= a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj} \end{aligned}$$



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS Craciunescu - Rezultatul unei Aplicatii Geostatistice WGS-84-V1 = 6173000/72224m



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS Craciunescu - Rezultatul unei Aplicatii Geostatistice WGS-84-V1 = 6173000/72224m

Metode Directe

- Eliminarea Gauss - Jordan (Wilhelm Jordan (1842 - 1899) geodez german)
- Este o variatie a metodei de eliminare gaussiene in care si elementele de sub diagonală principala, dar si elementele de deasupra diagonalei principale sunt reduse la 0;
- De obicei randurile sunt scalate pentru a se obtine matricea unitate;



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS Craciunescu - Rezultatul unei Aplicatii Geostatistice WGS-84-V1 = 6173000/72224m



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS Craciunescu - Rezultatul unei Aplicatii Geostatistice WGS-84-V1 = 6173000/72224m

Metode Directe

Factorizarea LU

$A=LU$, unde L este inferior triunghiular, iar U este superior triunghiular;

Este foarte utila o astfel de factorizare deoarece pentru a solutiona un sistem $Ax=b$, unde $A=LU$, putem reduce problema la a solutiona doua sisteme lineare usor de soluat, astfel, notam cu $y=Ux$ si deci vom avea doi pasi:

- Pas 1: sol. sistemul inf triunghiular $Ly=b$ pentru y ;
- Pas 2: sol. sistemul sup triunghiular $Ux=y$ pt x ;



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS Craciunescu - Rezultatul unei Aplicatii Geostatistice WGS-84-V1 = 6173000/72224m



Metode Directe

Factorizarea LU - Cholesky

$$\begin{aligned} l_{ii} &= \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}; i = 1 : n \\ l_{ij} &= \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik}l_{jk}}{l_{jj}}; j = 1 : i - 1. \end{aligned}$$



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS Craciunescu - Rezultatul unei Aplicatii Geostatistice WGS-84-V1 = 6173000/72224m



Metode Directe

• Eliminarea Gauss - Jordan

$$\text{- normalizare: } A_{pj} = \frac{A_{pj}}{A_{pp}}, \quad p = 1 : n, \quad j = p : 2n$$

$$\text{- reducere: } A_{ij} = A_{ij} - A_{ip}A_{pj}, \quad p = 1 : n, \quad i = 1 : n, \quad j = p : 2n$$



MN Primavara 2015 CS-UPB
© Craiovea Mihai IULYIUS Craciunescu - Rezultatul unei Aplicatii Geostatistice WGS-84-V1 = 6173000/72224m



Metode Directe

- Eliminarea Gauss - Jordan

Daca inlocuim vectorul b cu matricea I_n, atunci aplicand metoda Gauss-Jordan cu scalare, vom obtine in stanga, matricea I_n, iar in dreapta inversa matricei A, A^-1;



MN Primavara 2015 CS-UPB

Metode Directe

A - matrice triunghiular inferioara cu substitutie inainte;

$$\mathbf{x}_i = \frac{\mathbf{b}_i - \sum_{j=1}^{i-1} \mathbf{A}_{ij} \mathbf{x}_j}{\mathbf{A}_{ii}}, \quad i = 1 : n$$

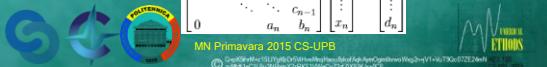
A - tridiagonala - Algoritmul Thomas (Llewellyn Hilleth Thomas 1903 -1992) fizician si matematician englez;

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i,$$

$$a_1 = 0 \quad c_n = 0$$

$$\begin{bmatrix} b_1 & c_1 & & 0 \\ a_2 & b_2 & c_2 & \\ & a_3 & b_3 & \ddots \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}$$

MN Primavara 2015 CS-UPB



SEL Code Matlab (x=A\b)

```
function b = gee_its_short (A, b)
n = size (A,1) ;
for k = 1:n
    [x i] = max (abs (A (k:n,k))) ;
    i = i+k-1 ;
    A ([k i],:) = A ([i k],:) ;
    b ([k i],:) = b ([i k],:) ;
    A (k+1:n,k) = A (k+1:n,k) / A (k,k) ;
    A (k+1:n,k+1:n) = A (k+1:n,k+1:n) - A (k+1:n,k) * A (k,k+1:n) ;
    b (k+1:n,:) = b (k+1:n,:)- A (k+1:n,k) * b (k,:) ;
end
for k = n:-1:1
    b (k,:) = b (k,:) / A (k,k) ;
    b (1:k-1,:) = b (1:k-1,:) - A (1:k-1,k) * b (k,:) ;
end
```

MN Primavara 2015 CS-UPB



Metode Directe

- Rezolvarea sistemelor lineare particulare

A - matrice triunghiular superioara cu substitutie inapoi;

$$\mathbf{x}_i = \frac{\mathbf{b}_i - \sum_{j=i+1}^n \mathbf{A}_{ij} \mathbf{x}_j}{\mathbf{A}_{ii}}, \quad i = n : -1 : 1$$

daca i=n, sum =0;

Numarul total de operatii(+,-,/,*) este

$$\sum_{i=n}^1 [2(n - (i + 1) + 1) - 1 + 1 + 1] = n^2$$

iar complexitatea este O(n^2);



MN Primavara 2015 CS-UPB

Metode Directe

A - matrice triunghiular inferioara cu substitutie inainte;

$$\mathbf{x}_i = \frac{\mathbf{b}_i - \sum_{j=1}^{i-1} \mathbf{A}_{ij} \mathbf{x}_j}{\mathbf{A}_{ii}}, \quad i = 1 : n$$

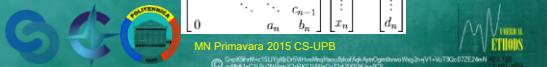
A - tridiagonala - Algoritmul Thomas (Llewellyn Hilleth Thomas 1903 -1992) fizician si matematician englez;

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i,$$

$$a_1 = 0 \quad c_n = 0$$

$$\begin{bmatrix} b_1 & c_1 & & 0 \\ a_2 & b_2 & c_2 & \\ & a_3 & b_3 & \ddots \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}$$

MN Primavara 2015 CS-UPB



Metode Directe

- Pas1: (eliminare subdiag)

$$\mathbf{b}_p = \mathbf{b}_p - \frac{\mathbf{a}_p}{\mathbf{b}_{p-1}} \cdot \mathbf{c}_{p-1}$$

$$\mathbf{d}_p = \mathbf{d}_p - \frac{\mathbf{a}_p}{\mathbf{b}_{p-1}}, \quad p = 2 : n$$

- Pas2: (rezolvare sistem 2 diag)

cu i=n-1:-1:1

Complexitate O(n);

$$\mathbf{x}_n = \frac{\mathbf{d}_n}{\mathbf{b}_n}$$

$$\mathbf{x}_i = \frac{\mathbf{d}_i - \mathbf{c}_i \mathbf{x}_{i+1}}{\mathbf{b}_i},$$



MN Primavara 2015 CS-UPB

Bibliografie Metode Directe SEL

- Joe D. Hoffman, Numerical Methods for Engineers and Scientists, 2nd edition, Marcel Dekker, 2001;
- Jaan Kiusalaas, NUMERICAL METHODS IN ENGINEERING, 2nd edition, Cambridge University Press, 2010;
- Rao V. Dukkipati, Numerical Methods, New Age International (P) Ltd., Publishers, 2010;
- Richard L. Burden, J. Douglas Faires, Numerical Analysis, 9th edition, Brooks/Cole, Cengage Learning, 2011;



MN Primavara 2015 CS-UPB

Eliminare gaussiană cu pivotare totală și scalare. Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal

Noțiuni teoretice

Acest capitol are ca scop familiarizarea cu metodele numerice directe de transformare a unei matrice nesingulare la forma superior triunghiulară sau inferior triunghiulară. Aceste metode poartă numele de *eliminări Gaussiene*. Cea mai simplă metodă de eliminare este *eliminarea gaussiană cu pivotare parțială*. Pentru această metodă vom prezenta algoritmul GPP. Cea mai bună metodă este *eliminarea gaussiană cu pivotare totală sau completă*, numit GPT în cadrul acestui capitol. O tehnică de scalare care micșorează eroarea și elimină anularea flotantă este *pivotarea parțială cu pivot scalat pe coloană* (algoritmul GPPS).

Eliminarea gaussiană este echivalentă cu metoda factorizării LU care trebuie să fie utilizată împreună cu o strategie de pivotare adecvată (factorizarea LUP). O strategie de pivotare este necesară în general deoarece este posibil ca eliminarea gaussiană să nu poată transforma o matrice dată la o formă triunghiulară. Pentru o matrice simetrică și pozitiv definită, cea mai bună metodă de aducere la forma triunghiulară este folosirea factorizării Cholesky.

Algoritmul Thomas este o formă simplificată a eliminării gaussiene pentru o matrice tridiagonală. Acest algoritm este folosit pentru rezolvarea sistemelor de ecuații liniare tridiagonale (des întâlnite în metodele de interpolare cu funcții spline). Complexitatea algoritmului Thomas este $O(n)$ în timp ce eliminările gaussiene au complexitatea $O(n^3)$.

Eliminare gaussiană - G

Eliminarea gaussiană este o tehnică pentru transformarea matricei A la forma superior triunghiulară. Matricea de transformare T este o matrice inferior triunghiulară unitară obținută ca o secvență (produs) de transformări inferior triunghiulare elementare de forma $T = T_{n-1}T_{n-2}\dots T_1$, unde matricele T_p sunt inferior triunghiulare, de ordin n , de forma:

$$T_p = I_n - t_p e_p^T,$$

e_p este coloana p din matricea unitate și

$$t_p = [0 \quad \cdots \quad 0 \quad \mu_{p+1p} \quad \cdots \quad \mu_{np}].$$

Metoda de mai sus se poate realiza dacă toate submatricele de forma $A^{[p]} = A(1:p, 1:p)$ sunt nesingulare. Scalarii μ_{ip} , numiți *multiplicatori gaussiani*, ce asigură satisfacerea condiției de anulare a elementelor din coloana p , de sub diagonala principală, au expresia:

$$\mu_{ip} = a_{ip}/a_{pp}, \quad i = p + 1 : n$$

În efectuarea operației $A \leftarrow T_p A$, se vor memora multiplicatorii gaussiani în locul zerourilor create sub diagonala principală, primele $p - 1$ coloane ale lui A nu sunt afectate, iar coloanele a_j , $j = p + 1 : n$ sunt transformate astfel:

$$(T_p a_j)_i = ((I_n - t_p e_p^T) a_j)_i = (a_j - t_p a_{pj})_i = a_{ij} - \mu_{ip} a_{pj}, \quad i = p + 1 : n.$$

Algoritmul G de eliminare gaussiană este:

Algorithm 1 Eliminare gaussiană

```

procedure G( $A$ )
    for  $p = 1 : n - 1$  do
        for  $i = p + 1 : n$  do
             $\mu_{ip} = a_{ip}/a_{pp};$ 
             $a_{ip} = 0;$ 
            for  $j = p + 1 : n$  do
                 $a_{ij} = a_{ij} - \mu_{ip} a_{pj};$ 
            end for
        end for
    end for
    return  $A;$ 
end procedure

```

Numărul de operații pentru algoritmul G este $O(n^3)$ (aproximativ $\frac{2n^3}{3}$ operații), iar memoria folosită, conform cu schema descrisă este $O(n^2)$. Nesingularitatea submatricelor nu este o condiție necesară pentru existența și unicitatea soluției unui sistem de forma $Ax = b$, unde A este adusă prin transformare la forma superior triunghiulară. Pentru a elimina această condiție se introduc strategiile de pivotare:

- Eliminarea gaussiană cu pivotare parțială - GPP;
- Eliminarea gaussiană cu pivotare parțială cu pivot scalat - GPPS;
- Eliminarea gaussiană cu pivotare totală - GPT.

Singularitatea submatricelor $A^{[p]}$ este echivalentă cu anularea elementului a_{pp} , numit pivot, la pasul p al algoritmului G. Consecința acestei anulări conduce la imposibilitatea calculului multiplicatorilor gaussieni.

Este necesară aducerea pe poziția pivotului (linia p și coloana p) a unui element nenul, preferabil de modul cât mai mare, prin permutare de linii și/sau coloane.

O matrice de permutare este o matrice care are un singur element nenul egal cu 1 pe orice linie și pe orice coloană a sa. Ea se obține din matricea unitate prin interschimbarea (o dată sau de mai multe ori) a două linii și/sau a două coloane. Iată un exemplu (interschimbarea liniilor 1 și 2, apoi a coloanelor 2 și 3):

$$P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Interschimbarea a două linii ale unei matrice este echivalentă cu multiplicarea acelei matrice cu o matrice de permutare la stânga. Interschimbarea a două coloane este echivalentă cu multiplicarea matricei cu o matrice de permutare la dreapta.

Eliminare gaussiană cu pivotare parțială - GPP

Pivotarea parțială are loc numai prin permutarea liniilor. La pasul p al algoritmului G se aduce în poziția (p, p) a pivotului cel mai mare element în modul dintre elementele subdiagonale din coloana p , fie acesta $a_{i_p p} \neq 0$, prin permutarea liniilor p și i_p . Acest lucru este echivalent cu multiplicarea matricei A la stânga cu matricea de permutare $P_{i_p p} \stackrel{\text{not}}{=} P_p$, astfel încât pasul p calculează $A \leftarrow T_p P_p A$, întregul algoritm fiind:

$$A \leftarrow U = T_{n-1} P_{n-1} T_{n-2} P_{n-2} \dots T_1 P_1 A$$

Eliminare gaussiană cu pivotare totală și scalare. Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal

Algoritmul GPP de eliminare gaussiană cu pivotare parțială este:

Algorithm 2 Eliminarea gaussiană cu pivotare parțială

```

1: procedure GPP( $A$ )
2:   for  $p = 1 : n - 1$  do
3:     Determină primul  $i_p$  ( $p \leq i_p \leq n$ ) a.î.  $|a_{i_p p}| = \max_{i=p:n} \{|a_{ip}| \}$ ;  $v(p) = i_p$ ;
4:      $v(p) = i_p$ ;
5:     for  $j = p : n$  do
6:        $a_{pj} \leftrightarrow a_{i_p j}$ ;
7:     end for
8:     for  $i = p + 1 : n$  do
9:        $\mu_{ip} = a_{ip}/a_{pp}$ ;
10:       $a_{ip} = 0$ ;
11:      for  $j = p + 1 : n$  do
12:         $a_{ij} = a_{ij} - \mu_{ip}a_{pj}$ ;
13:      end for
14:    end for
15:  end for
16:  return  $A, v$ ;
17: end procedure
```

Pașii 8-14 reprezintă algoritmul G pentru matricea permuatată. Vectorul v memorizează permutările de linii.

Eliminare gaussiană cu pivotare parțială cu pivot scalat - GPPS

Această tehnică este similară cu precedenta, doar că definim la început un factor de scalare pentru fiecare linie i , factor de forma:

$$s_i = \max_{j=p:n} \{|a_{ij}| \} \quad sau \quad s_i = \sum_{j=p}^n |a_{ij}|$$

Dacă există un i astfel încât $s_i = 0$, atunci matricea este singulară. Pașii următori vor stabili interschimbările care se vor face. La pasul p se va găsi întregul i_p astfel încât:

$$\frac{|a_{i_p p}|}{s_{i_p}} = \max_{i=p:n} \frac{|a_{ip}|}{s_i}$$

Eliminare gaussiană cu pivotare totală și scalare. Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal

Scalarea ne garantează că cel mai mare element din fiecare coloana are înainte de comparațiile necesare pentru schimbare mărimea relativă 1. Scalarea se realizează doar în comparații, nu efectiv în matrice, astfel că impărțirea cu factorul de scalare nu produce nici o eroare de rotunjire.

Algoritmul GPPS de eliminare gaussiană cu pivotare parțială cu pivot scalat este:

Algorithm 3 Eliminarea gaussiană cu pivotare parțială cu pivot scalat

```
1: procedure GPPS( $A$ )
2:   for  $p = 1 : n - 1$  do
3:     Determină  $i_p$  a.î.  $\frac{|a_{i_p p}|}{s_{i_p}} = \max_{i=p:n} \frac{|a_{i p}|}{s_i}$ ;  $v(p) = i_p$ ;
4:     for  $j = 1 : n$  do
5:        $a_{pj} \leftrightarrow a_{i_p j}$ ;
6:     end for
7:     for  $i = p + 1 : n$  do
8:        $\mu_{ip} = a_{ip}/a_{pp}$ ;
9:        $a_{ip} = 0$ ;
10:      for  $j = p + 1 : n$  do
11:         $a_{ij} = a_{ij} - \mu_{ip}a_{pj}$ ;
12:      end for
13:    end for
14:  end for
15:  return  $A, v$ ;
16: end procedure
```

Pașii 7-13 reprezintă algoritmul G pentru matricea permuatată. Vectorul v memorează permutările de linii. Complexitatea algoritmului GPP este aceeași cu a lui G.

Eliminarea gaussiană cu pivotare totală - GPT

Stabilitate numerică mai bună se obține dacă pivotul de la pasul p se alege drept cel mai mare element în modul dintre elementele a_{ij} , cu $i = p : n$, $j = p : n$, fie el $a_{i_p j_p}$, și este adus în poziția (p, p) a pivotului prin permutarea liniilor p și i_p și a coloanelor p și j_p . Acest lucru este echivalent cu multiplicarea matricei A la stânga cu matricea de permutare $P_{i_p p}^{st} \stackrel{\text{not}}{=} P_p^{st}$ și cu multiplicarea matricei A la dreapta cu matricea de

Eliminare gaussiană cu pivotare totală și scalare. Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal

permutare $P_{j_p p}^{dr} \stackrel{\text{not}}{=} P_p^{dr}$. La pasul p se calculează $A \leftarrow T_p P_p^{st} A P_p^{dr}$, întregul algoritm fiind:

$$A \leftarrow U = T_{n-1} P_{n-1}^{st} T_{n-2} P_{n-2}^{st} \dots T_1 P_1^{st} A P_1^{dr} P_2^{dr} \dots P_{n-1}^{dr}.$$

Algoritmul GPPS de eliminare gaussiană cu pivotare totală este:

Algorithm 4 Eliminarea gaussiană cu pivotare totală

```
1: procedure GPT( $A$ )
2:   for  $p = 1 : n - 1$  do
3:     Determină  $i_p$  și  $j_p$  ( $p \leq i_p, j_p \leq n$ ) a.i  $|a_{i_p j_p}| = \max_{i=p:n, j=p:n} \{|a_{ij}|\};$ 
4:      $v_{st}(p) = i_p; v_{dr}(p) = j_p;$ 
5:     for  $j = p : n$  do
6:        $a_{pj} \leftrightarrow a_{i_p j};$ 
7:     end for
8:     for  $i = 1 : n$  do
9:        $a_{ip} \leftrightarrow a_{ij_p};$ 
10:      end for
11:      for  $i = p + 1 : n$  do
12:         $\mu_{ip} = a_{ip}/a_{pp};$ 
13:         $a_{ip} = 0;$ 
14:        for  $j = p + 1 : n$  do
15:           $a_{ij} = a_{ij} - \mu_{ip} a_{pj};$ 
16:        end for
17:      end for
18:      return  $A, v_{st}, v_{dr};$ 
19: end procedure
```

Pașii 10-16 reprezintă algoritmul G pentru matricea permuatată. Vectorul v_{st} memorează permutările de linii iar v_{dr} memorează permutările de coloane. Complexitatea algoritmului GPP este aceeași cu a lui G.

Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal

Sistemul tridiagonal se poate scrie compact sub forma $a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i$, $i = 1 : n$ cu mențiunea că $a_1 = 0$ și $c_n = 0$, iar componentele x_0 și x_{n+1} nu sunt

definite. Forma generală este:

$$\begin{bmatrix} b_1 & c_1 & & 0 \\ a_2 & b_2 & c_2 & \\ & a_3 & b_3 & \ddots \\ & \ddots & \ddots & c_{n-1} \\ 0 & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}$$

Algoritmul presupune modificarea coeficienților sistemului și aducerea lor la o formă prin care sistemul de poate rezolva direct prin substituție înapoi.

$$c'_i = \begin{cases} \frac{c_i}{b_i} & i = 1 \\ \frac{c_i}{b_i - c'_{i-1}a_i} & i = 2 : n-1 \end{cases}$$

$$d'_i = \begin{cases} \frac{d_i}{b_i} & i = 1 \\ \frac{d_i - d'_{i-1}a_i}{b_i - c'_{i-1}a_i} & i = 2 : n \end{cases}$$

Prin formulele de mai sus s-a executat de fapt un pas al eliminării gaussiene. Soluția sistemului va fi dată de formulele:

$$\begin{cases} x_n = d'_n \\ x_i = d'_i - c'_i x_{i+1} & i = n-1 : 1 \end{cases}$$

Sistemul nu se mai păstrează în memorie prin întregă lui matrice, ci doar prin trei vectori de valori corespunzătoare celor trei diagonale.

Algoritmul Thomas este utilizat deoarece este rapid și apare frecvent în practică: interpolări, ecuații diferențiale, etc. Deși situația este rară, algoritmul poate fi instabil dacă $b_i - c'_{i-1}a_i = 0$ sau numeric zero pentru orice i . Aceasta se întâmplă dacă matricea este singulară, dar în cazuri rare se poate întampla și pentru o matrice nesingulară. Condiția de stabilitate este, $\forall i$:

$$|b_i| > |a_i| + |c_i|$$

condiție care indică diagonal-dominanța matricei A . Dacă algoritmul este numeric instabil, se pot aplica strategii de pivotare, ca în cazul eliminării gaussiene.

Eliminare gaussiană cu pivotare totală și scalare. Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal

Probleme rezolvate

Problema 1

Fie sistemul de ecuații:

$$\begin{cases} 5.2x_1 + 7.1x_2 = 19.8 \\ 2.4x_1 + 3.2x_2 = 4.1 \end{cases}$$

Rezolvați sistemul folosind eliminare gaussiană cu pivotare parțială.

Soluție:

Valoarea maximă a coeficientilor de pe prima coloană este 5.2. Așadar, nu vom schimba ordinea ecuațiilor.

$$\mu_{21} = \frac{2.4}{5.2} = 0.46154$$

Sistemul inițial este echivalent cu:

$$\begin{cases} 5.2 \cdot x_1 + 7.1 \cdot x_2 = 19.8 \\ -0.07692 \cdot x_2 = -5.0385 \end{cases}$$

În final, obținem soluția:

$$\begin{cases} x_1 = -85.625 \\ x_2 = 65.5 \end{cases}$$

Problema 2

Să se scrie un program OCTAVE pentru a implementa algoritmul de eliminare gaussiană.

Soluție:

```
1 function [A, b] = G(A, b)
2     [n n] = size(A);
3
4     for p = 1 : n -1
```

Eliminare gaussiană cu pivotare totală și scalare. Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal

```
5   for i = p + 1 : n
6     if A(p, p) == 0
7       continue;
8     endif
9
10    tp = A(i, p)/A(p, p);
11    A(i, p) = 0;
12    for j = p + 1 : n
13      A(i, j) = A(i, j)-tp*A(p, j);
14    endfor
15
16    b(i) = b(i)-tp*b(p);
17  endfor
18 endfor
19 endfunction
```

Listing 1: Eliminarea gaussiană.

Problema 3

Să se scrie un program OCTAVE pentru a implementa algoritmul de eliminarea gaussiană cu pivotare parțială.

Soluție:

```
1 function [A b] = GPP(A, b)
2 [n n] = size(A);
3
4 for p = 1 : n -1
5   pivot = -inf;
6   linie_pivot = -1;
7
8   %calculez maximul dintre elementele A(p : n, p)
9   for i = p : n
10     if pivot < abs(A(i, p));
11       pivot = abs(A(i, p));
12       linie_pivot = i;
13     endif
14   endfor
15
```

Eliminare gaussiană cu pivotare totală și scalare. Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal

```
16 %permutarea liniilor linie_pivot si p
17 if p ~= linie_pivot
18   for j = p : n
19     t = A(p, j);
20     A(p, j) = A(linie_pivot, j);
21     A(linie_pivot, j) = t;
22   endfor
23
24   %permutarea elementelor b(linie_pivot) si b(p)
25   t = b(linie_pivot);
26   b(linie_pivot) = b(p);
27   b(p) = t;
28 endif
29
30 %eliminare gaussiana
31 for i = p + 1 : n
32   if A(p, p) == 0
33     continue;
34   endif
35
36   tp = A(i, p)/A(p, p);
37   A(i, p) = 0;
38   for j = p + 1 : n
39     A(i, j) = A(i, j)-tp*A(p, j);
40   endfor
41
42   b(i) = b(i)-tp*b(p);
43   endfor
44 endfor
45 endfunction
```

Listing 2: Eliminarea gaussiană cu pivotare parțială.

Problema 4

Să se scrie un program OCTAVE pentru a implementa algoritmul de eliminarea gaussiană cu pivotare totală.

Soluție:

Eliminare gaussiană cu pivotare totală și scalare. Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal

```
1 function [A, b] = GPT(A, b)
2     [n n] = size(A);
3
4     for p = 1 : n - 1
5         pivot = -inf
6         linie_pivot = -1;
7         coloana_pivot = -1;
8
9         %calculez maximul in submatricea A(p : n, p : n)
10        for i = p : n
11            for j = p : n
12                if pivot < abs(A(i, j));
13                    pivot = abs(A(i, j));
14                    linie_pivot = i;
15                    coloana_pivot = j;
16                endif
17            endfor
18        endfor
19
20        if p ~= linie_pivot
21            %permutarea liniilor linie_pivot si p
22            for j = p : n
23                t = A(p, j);
24                A(p, j) = A(linie_pivot, j);
25                A(linie_pivot, j) = t;
26            endfor
27
28            %permutarea elementelor b(linie_pivot) si b(p)
29            t = b(linie_pivot);
30            b(linie_pivot) = b(p);
31            b(p) = t;
32        endif
33
34        %permutarea coloanelor coloana_pivot si p
35        if p ~= coloana_pivot
36            for i = 1 : n
37                t = A(i, p);
38                A(i, p) = A(i, coloana_pivot);
39                A(i, coloana_pivot) = t;
40            endfor
```

Eliminare gaussiană cu pivotare totală și scalare. Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal

```
41      endif
42
43  %eliminare gaussiana
44  for i = p + 1 : n
45    if A(p, p) == 0
46      continue;
47    endif
48
49    tp = A(i, p)/A(p, p);
50    A(i, p) = 0;
51    for j = p + 1 : n
52      A(i, j) = A(i, j)-tp*A(p, j);
53    endfor
54
55    b(i) = b(i)-tp*b(p);
56  endfor
57 endfor
58 endfunction
```

Listing 3: Eliminarea gaussiană cu pivotare totală.

Problema 5

Să se scrie un program OCTAVE pentru a implementa algoritmul Thomas de rezolvare a unui sistem 3-diagonal.

Soluție:

```
1 function x = Thomas(a, b, c, d)
2   n = length(d);
3
4   %transform a astfel incat sa aiba n elemente, cu primul
5   %element 0
6   a = [0; a];
7
8   % Operatiile la limita;
9   c(1) = c(1)/b(1);
10  d(1) = d(1)/b(1);
11
12  % calculul coeficientilor pe caz general.
```

Eliminare gaussiană cu pivotare totală și scalare. Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal

```
12 |     for i = 2 : n-1
13 |         temp = b(i)-a(i)*c(i-1);
14 |         c(i) = c(i)/temp;
15 |         d(i) = (d(i)-a(i)*d(i-1))/temp;
16 |     endfor
17 |     d(n) = (d(n)-a(n)*d(n-1))/(b(n)-a(n)*c(n-1));
18 |
19 | % Substitutia inapoi pentru rezolvarea sistemului de ecuatii
20 | x(n) = d(n);
21 | for i = n-1 : -1 : 1
22 |     x(i) = d(i)-c(i)*x(i+1);
23 | endfor
24 | x=x';
25 | endfunction
```

Listing 4: Algoritmul Thomas.

Date de intrare:

$$a = \begin{bmatrix} 2 & 9 & 2 & 3 & 6 \end{bmatrix}, b = \begin{bmatrix} 12 & 15 & 2 & 9 & 1 & 0 \end{bmatrix}, \\ c = \begin{bmatrix} 10 & 3 & 9 & 1 & 4 \end{bmatrix}, d = \begin{bmatrix} 1 & 5 & 9 & 11 & 13 & 7 \end{bmatrix}$$

Date de ieșire:

$$x = [0.160494 \quad -0.092593 \quad 2.022634 \quad 0.643118 \quad 1.166667 \quad 2.475995]$$

Probleme propuse

Problema 1

Implementați în OCTAVE algoritmul GPPS.

Problema 2

Modificați algoritmii GPP și GPT pentru a lucra cu o matrice superior Hessemberg. Implementați H_GPP și H_GPT.

Eliminare gaussiană cu pivotare totală și scalare. Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal

Problema 3

Construiți o variantă modificată pentru algoritmul lui Thomas care să lucreze cu matricea:

$$A = \begin{bmatrix} b_1 & 0 & c_1 & & & 0 \\ 0 & b_2 & 0 & c_2 & & \\ a_3 & 0 & b_3 & 0 & c_3 & \\ \ddots & \ddots & \ddots & \ddots & \ddots & \\ & a_{n-2} & 0 & b_{n-2} & 0 & c_{n-2} \\ & & a_{n-1} & 0 & b_{n-1} & 0 \\ 0 & & & a_n & 0 & b_n \end{bmatrix}$$

Scrieți o funcție OCTAVE cu semnătura `function x = solve(a,b,c,d)`, care rezolvă sistemul de ecuații $Ax = d$.

Problema 4

Fie sistemul de ecuații: $\begin{cases} 1.5 \cdot x_1 - 2.1 \cdot x_2 = 8.3 \\ -7.6 \cdot x_1 + 3.11 \cdot x_2 = 6.7 \end{cases}$

Rezolvați sistemul folosind eliminările GPP, GPPS, GPT.

Metode Numerice @ CS-UPB

Echipa MN CS - 2015:

Profesorii:

- Pantelimon George Popescu - Seria CA;
- Florin Pop - Seria CB, CC;

Asistenti:

PG Popescu, Clementin Cercel, Sorin Ciolofan, Bogdan Tiganoaia, Diana Udrea, Alexandru Tifrea, David Iancu, Bogdan Cristian Marchis, Radu Poenaru, Madalina Grosu, Andrei Vlad Postoaca, Radu Constantinescu, Roxana Istrate, Madalina Hristache, Adelina Vidovici;



MN Primavara 2015 CS-UPB
© Craiovea131 (julie) via Wikimedia Commons / CC-BY-SA 3.0 license



NM

Metode Iterative

Un sistem linear mare $Ax=b$ este indicat sa fie rezolvat prin metode iterative atunci cand matricea coeficientilor A este dominant diagonală.

Vom prezenta cateva metode iterative, ca **Jacobi**, **Gauss-Seidel**, **Successive-over-relaxation(SOR)**, etc.

Metodele iterative pornesc de la o soluție initială aleasă x_0 și generează o altă soluție x_1 , mai bună care se bazează pe reducerea diferenței dintre soluția x_k , la pasul k și soluția x_{k-1} , de la pasul anterior.



MN Primavara 2015 CS-UPB
© Craiovea131 (julie) via Wikimedia Commons / CC-BY-SA 3.0 license



NM

Metode Iterative

Procedura se repeta pana se atinge un anumit criteriu de convergenta. Daca metoda, la un moment dat produce o modificare nesemnificativa a vectorului solutie, metoda trebuie opresa. De asemenea metoda se opreste daca o marime a modificarii, relative sau absolute, vectorului solutie este mai mica decat un criteriu de convergenta specificat.



MN Primavara 2015 CS-UPB
© Craiovea131 (julie) via Wikimedia Commons / CC-BY-SA 3.0 license



NM

Metode Iterative/Aproximative pt. SEL



MN Primavara 2015 CS-UPB
© Craiovea131 (julie) via Wikimedia Commons / CC-BY-SA 3.0 license

**NUMERICAL
METHODS**



Metode iterative

Procedura se repeta pana converge catre solutia reala. Metoda este convergenta daca la fiecare pas se apropi cat mai mult de solutia reala a sistemului, si deci cu cat iteram mai mult cu atat ne apropiem mai mult de solutie.

Acstea metode nu converg pentru toate SEL. O conditie suficienta pentru convergenta unei metode iterative de solutionare SEL este diagonal-dominanta matricei coeficientilor, pentru orice vector solutie ales initial.



MN Primavara 2015 CS-UPB
© Craiovea131 (julie) via Wikimedia Commons / CC-BY-SA 3.0 license



Metode iterative

Jacobi (Carl Gustav Jacob Jacobi (1804 - 1851) matematician german)

Rescriem un sistem SEL $Ax=b$ in forma

$$\sum_{j=1}^n a_{ij}x_j = b_i \quad (i = 1, 2, \dots, n)$$

Metoda considera la fiecare pas elementul necunoscut x_i asociat elementului diagonal de la resp pas.



MN Primavara 2015 CS-UPB
© Craiovea131 (julie) via Wikimedia Commons / CC-BY-SA 3.0 license



Metode Iterative

si deci

$$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} x_j - \sum_{j=i+1}^n a_{i,j} x_j^{(0)} \right) \quad (i = 1, 2, \dots, n)$$

Considerand $x^{(0)}$ solutie initiala, il vom calcula pe $x^{(1)}_i$ in functie de $x^{(0)}$ astfel

$$x_i^{(1)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(0)} - \sum_{j=i+1}^n a_{i,j} x_j^{(0)} \right) \quad (i = 1, 2, \dots, n)$$



Metode Iterative

Ultima ecuatie se poate pune si sub forma

$$\begin{aligned} x_i^{(k+1)} &= x_i^{(k)} + \frac{R_i^{(k)}}{a_{i,i}} \quad (i = 1, 2, \dots, n) \\ R_i^{(k)} &= b_i - \sum_{j=1}^n a_{i,j} x_j^{(k)} \quad (i = 1, 2, \dots, n) \end{aligned}$$

$R_i^{(k)}$ reprezentand restul ecuatiei i.



Metode Iterative

Acuratetea metodelor iterative este masurata raportat la eroarea metodei, ce poate fi specificata in doua feluri: eroare absoluta si eroare relativa, astfel

$$\begin{aligned} \text{EroareaAbsoluta} &= \text{ValoareaAprox} - \text{ValoareaExacta} \\ \text{EroareaRelativa} &= \text{EroareaAbsoluta}/\text{ValoareaExacta} \end{aligned}$$



Metode iterative

si general pentru un pas oarecare k

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k)} - \sum_{j=i+1}^n a_{i,j} x_j^{(k)} \right) \quad (i = 1, 2, \dots, n)$$

Si mai departe adunand si scazand a_iix_i^(k) avem

$$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^n a_{i,j} x_j^{(k)} \right) \quad (i = 1, 2, \dots, n)$$



Metode iterative

Acuratetea si Convergenta metodelor iterative

Metodele iterative sunt mai putin vulnerabile la erorile de rotunjire deoarece: 1. sistemul este dominant diagonal; 2. de obicei sistemul este sparse; 3. fiecare iteratie nu este afectata de erorile de rotunjire de la precedenta iteratie;

Termenul de acuratete asociat metodelor iterative se refera la numarul de zecimale obtinut/dorit in calcule, iar convergenta se refera la pasul de iteratie cand acuratetea a fost atinsa.

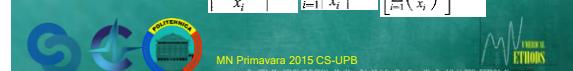


Metode Iterative

Convergenta metodelor iterative este atinsa cand un criteriu de acuratete a fost satisfacut.

Pentru ca nu se cunoaste valoarea exacta a solutiei, eroarea este masurata la fiecare pas ca fiind diferente dintre ce s-a obtinut la respectivul pas fata de precedentul. Cateva criterii de convergenta se practica:

- Eroare Absoluta $|(Ax)_\text{max}| \leq \varepsilon$ $\left| \sum_{i=1}^n \Delta x_i \right| \leq \varepsilon$ $\left[\sum_{i=1}^n (\Delta x_i)^2 \right]^{1/2} \leq \varepsilon$
- Eroare Relativa $\left| \frac{(Ax)_\text{max}}{x_i} \right| \leq \varepsilon$ $\left| \sum_{i=1}^n \frac{\Delta x_i}{x_i} \right| \leq \varepsilon$ $\left[\sum_{i=1}^n \left(\frac{\Delta x_i}{x_i} \right)^2 \right]^{1/2} \leq \varepsilon$



Metode Iterative

Gauss-Seidel (Johann Carl Friedrich Gauss (1777-1855) matematician si fizician german) - (Philip Ludwig von Seidel (1821-1896) matematician german)

Daca la Jacobi toate valorile vectorului $x^{(k+1)}$ se calculau in functie de $x^{(k)}$, aici valorile vectorului $x^{(k+1)}$ se calculeaza in functie de cele care au fost calculate deja in pasul respectiv si de cele ramase din pasul precedent, adica



MN Primavara 2015 CS-UPB

(C) Facultatea de Matematică și Calculatoare și Institutul de Matematică și Criptologie, Universitatea Politehnica București, 2015. Toate drepturile rezervate.

Metode Iterative

Successive Over Relaxation (SOR - 1950) ((David M. Young Jr. (1923 - 2008) matematician si calculatorist american, si H. Frankel) dar si mai inainte de (Lewis Fry Richardson(1881-1953) matematician, fizician, meteorolog si psiholog englez si Richard Vynne Southwell(1888 - 1970) matematician englez-mecanica aplicata))



MN Primavara 2015 CS-UPB

(C) Facultatea de Matematică și Calculatoare și Institutul de Matematică și Criptologie, Universitatea Politehnica București, 2015. Toate drepturile rezervate.

Metode Iterative

Successive Over Relaxation

Pentru $w < 1$ sistemul este under-relaxed si de obicei se foloseste la SEL atunci cand Gauss-Seidel se depareaza de solutie, dar de obicei se foloseste la Sisteme de Ecuatii Nelineare;

Pentru $w >= 2$ metoda este divergenta.

Acest factor w nu perturba solutia deoarece la finalul procesului de iteratie restul $R_i^{(k)}$ este aproape 0 si deci inmultit cu ceva tot aproximativ 0 ramane.



MN Primavara 2015 CS-UPB

(C) Facultatea de Matematică și Calculatoare și Institutul de Matematică și Criptologie, Universitatea Politehnica București, 2015. Toate drepturile rezervate.

Metode iterative

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i+1}^n a_{i,j} x_j^{(k)} \right) \quad (i = 1, 2, \dots, n)$$

Si considerand resturi, rescriem astfel:

$$x_i^{(k+1)} = x_i^{(k)} + \frac{R_i^{(k)}}{a_{i,i}} \quad (i = 1, 2, \dots, n)$$

$$R_i^{(k)} = b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i+1}^n a_{i,j} x_j^{(k)} \quad (i = 1, 2, \dots, n)$$



MN Primavara 2015 CS-UPB

(C) Facultatea de Matematică și Calculatoare și Institutul de Matematică și Criptologie, Universitatea Politehnica București, 2015. Toate drepturile rezervate.

Metode iterative

Successive Over Relaxation

Se inmulteste result $R_i^{(k)}$ cu un factor w (aceste metode se mai numesc metode de relaxare) si deci metoda se prezinta astfel

$$x_i^{(k+1)} = x_i^{(k)} + w \frac{R_i^{(k)}}{a_{i,i}} \quad (i = 1, 2, \dots, n)$$

$$R_i^{(k)} = b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i+1}^n a_{i,j} x_j^{(k)} \quad (i = 1, 2, \dots, n)$$

Se observa ca pentru $w=1$ -> SOR = Gauss-Seidel. Pentru $1 < w < 2$ sistemul este over-relaxed si de obicei se foloseste la SEL;



MN Primavara 2015 CS-UPB

(C) Facultatea de Matematică și Calculatoare și Institutul de Matematică și Criptologie, Universitatea Politehnica București, 2015. Toate drepturile rezervate.

Metode Iterative

Successive Over Relaxation

Problema acestei metode ramane selectia lui w optim astfel incat metoda sa conveargă mai repede. Din nefericire nu există metode de selectare pentru w optim si deci nu are rost sa folosim o astfel de metoda punctual ci doar daca avem spre exemplu de calculat mai multe solutii pentru mai multi vectori, b termeni liberi (avand aceasi matrice a coeficientilor). Si dupa cateva alegeri ale lui w , testand convergenta, alegem w optim si il folosim pentru restul de SEL(restul de vectori b).



MN Primavara 2015 CS-UPB

(C) Facultatea de Matematică și Calculatoare și Institutul de Matematică și Criptologie, Universitatea Politehnica București, 2015. Toate drepturile rezervate.

Metode Iterative

Forma Generala a Metodelor Iterative

Putem scrie $Ax=b$ astfel $x=(I-A)x+b$, adica $x=Tx+b$.

$$x_{k+1} = (I - A)x_k + b.$$

Iar daca $A=M-N$, atunci sistemul devine

$$x=M^{-1}Nx+M^{-1}b \quad Mx_{k+1} = Nx_k + b.$$

si sistemul

$Mz=r$ este mai usor de rezolvat decat $Ax=b$.



MN Primavara 2015 CS-UPB



Metode iterative

Forma Generala a Metodelor Iterative

Convergenta este data de valorile proprii ale lui

$$M^{-1}N = I - M^{-1}A$$

adica metoda converge data valorile proprii ale matricei de iteratie $M^{-1}N$, sunt mai mici ca 1 si cu cat sunt mai mici cu atat converge mai repede.



MN Primavara 2015 CS-UPB



Metode Iterative

Forma Generala a Metodelor Iterative

este data de formula

$$x^{(k)} = Tx^{(k-1)} + c$$

unde pentru Jacobi $A=D-L-U=D-(L+U)$ si deci

$$x = D^{-1}(L + U)x + D^{-1}b.$$

de unde $T_j = D^{-1}(L + U)$ si $c_j = D^{-1}b$



MN Primavara 2015 CS-UPB



Metode iterative

Forma Generala a Metodelor Iterative

Iar pentru Gauss-Seidel, unde $A=D-L-U=(D-L)-U$

$$x^{(k)} = (D - L)^{-1}Ux^{(k-1)} + (D - L)^{-1}b$$

de unde $T_k = (D - L)^{-1}U$ si $c_k = (D - L)^{-1}b$

Iar pentru SOR, unde

$$A=w(D-L-U)=(D-wL)-[(1-w)D+wU]$$

$$x^{(k)} = (D - \omega L)^{-1}[(1 - \omega)D + \omega U]x^{(k-1)} + \omega(D - \omega L)^{-1}b$$

de unde $T_\omega = (D - \omega L)^{-1}[(1 - \omega)D + \omega U]$

iar $c_\omega = \omega(D - \omega L)^{-1}b$



MN Primavara 2015 CS-UPB



Metode Iterative

Metoda Gradientului Conjugat

(Magnus Rudolph Hestenes (1906-1991) mat. american si Eduard L. Steifel (1907-1998) mat. elvetian)

Retinem produsul scalar si ale sale proprietati $\langle x, y \rangle = x^T y$

O matrice este semipozitiv definita SPD daca $x^T Ax > 0$ pentru orice x diferit de 0, ie $\langle x, Ax \rangle = x^T Ax > 0$

x^* este solutie pentru sistemul SPD $Ax=b$ daca si numai daca x^* minimizeaza

$$g(x) = \langle x, Ax \rangle - 2\langle x, b \rangle.$$



MN Primavara 2015 CS-UPB



Metode iterative

Metoda Gradientului Conjugat

Sau echivalent, se doreste x care minimizeaza

$$\phi(x) = \frac{1}{2}x^T Ax - b^T x.$$

Fie x o solutie aproximativa initiala pentru sistemul $Ax^*=b$ si v diferit de 0, o directie de cautare(adica in ce parte ma deplasez fata de x pentru cautarea solutiei), consideram $r=b-Ax$, vector reziduu si

$$t = \frac{\langle v, b - Ax \rangle}{\langle v, Av \rangle} = \frac{\langle v, r \rangle}{\langle v, Av \rangle}.$$



MN Primavara 2015 CS-UPB



Metode Iterative

Metoda Gradientului Conjugat

Daca r este diferit de 0 si r, v nu sunt ortogonali, atunci $g(x+tv) < g(x)$, ceea ce inseamna ca $x+tv$ este mai aproape de x^* decat x .

Si de aici deducem metoda.

Fie $x(0)$ o aproximatie initiala a lui x^* si fie $v(1)$ o directie de cautare initiala, atunci pentru $k=1, 2, \dots$ calculam

$$t_k = \frac{\langle v^{(k)}, b - Ax^{(k-1)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle}, \quad \text{si alegem noua directie de cautare } v(k+1).$$

$$x^{(k)} = x^{(k-1)} + t_k v^{(k)}$$



Metode Iterative

Metoda Gradientului Conjugat

Cum $\frac{\partial g}{\partial x_k}(\mathbf{x}) = 2 \sum_{i=1}^n a_{ki} x_i - 2b_k$, atunci

$$\nabla g(\mathbf{x}) = \left(\frac{\partial g}{\partial x_1}(\mathbf{x}), \frac{\partial g}{\partial x_2}(\mathbf{x}), \dots, \frac{\partial g}{\partial x_n}(\mathbf{x}) \right)^T = 2(A\mathbf{x} - \mathbf{b}) = -2r,$$

deci directia este data de vectorul r .

$$v^{(k+1)} = r^{(k)} = b - Ax^{(k)}$$

Pentru aceasta alegere metoda poarta numele de metoda de pasi descrescatori.



Metode Iterative

Metoda Gradientului Conjugat

Atunci $t_k = \frac{\langle v^{(k)}, b - Ax^{(k-1)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle} = \frac{\langle v^{(k)}, r^{(k-1)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle}$ iar $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + t_k v^{(k)}$.

Daca $v(k)$ sunt astfel alesi, iar A este SPD atunci $Ax(n)=b$, adica se atinge solutia exacta dupa exact n pasi.



Metode Iterative

Metoda Gradientului Conjugat

Pentru alegerea directiei de cautare $v(k+1)$, consideram functia $g(\mathbf{x})=g(x_1, x_2, \dots, x_n)$

$$g(x_1, x_2, \dots, x_n) = \langle \mathbf{x}, A\mathbf{x} \rangle - 2\langle \mathbf{x}, \mathbf{b} \rangle = \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i x_j - 2 \sum_{i=1}^n x_i b_i.$$

Directia de cautare este, in sensul in care vreau ca $g(\mathbf{x})$ sa descreasca cu fiecare pas, data de

$$-\nabla g(\mathbf{x})$$



Metode Iterative

Metoda Gradientului Conjugat

Desi metoda cu pasi descrescatori este buna pentru sisteme de ecuatii nelineare sau pentru probleme de optimizare, in cazul SEL aceasta metoda nu se foloseste deoarece are convergenta slaba.

Metode Iterative

Metoda Gradientului Conjugat

O alternativa pentru alegerea directiilor de cautare ar fi ca vectorii $\{v^{(1)}, \dots, v^{(n)}\}$ sa satisfaca A-ortogonalitatea, adica

$$\langle v^{(i)}, Av^{(j)} \rangle = 0, \quad i \neq j.$$



Metode Iterative

Metoda Gradientului Conjugat

Atunci $t_k = \frac{\langle v^{(k)}, b - Ax^{(k-1)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle} = \frac{\langle v^{(k)}, r^{(k-1)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle}$ iar $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + t_k v^{(k)}$.

Daca $v(k)$ sunt astfel alesi, iar A este SPD atunci $Ax(n)=b$, adica se atinge solutia exacta dupa exact n pasi.



Metode Iterative

Metoda Gradientului Conjugat

Aceasta alegerea a vectorilor directii de cautare se numeste metoda directiei conjugate.

Intre vectorii reziduu si vectorii directie avem ortogonalitate, ie pentru orice $k=1, 2, \dots, n$, $\langle r^{(k)}, v^{(j)} \rangle = 0$, pentru orice $j=1, 2, \dots, k$.



Metode Iterative

Metoda Gradientului Conjugat

Metoda Gradientului Conjugat alege directiile de cautare astfel incat vectorii reziduu sa fie ortogonali intre ei. Procedam astfel:

sa spunem ca am ajuns la pasul k si deci $v(1), \dots, v(k-1)$
si $x(0), \dots, x(k-1)$ au fost calculati astfel
 $x^{(k-1)} = x^{(k-2)} + t_{k-1}v^{(k-1)}$
unde $\langle v^{(i)}, Av^{(j)} \rangle = 0$ | $\langle r^{(i)}, r^{(j)} \rangle = 0$, $i \neq j$.

Acum daca $x(k-1)$ verifica $Ax=b$, ne oprim, daca nu atunci $r^{(k-1)} = b - Ax^{(k-1)} \neq 0$



Metode Iterative

Metoda Gradientului Conjugat

Deci il avem pe $v(k)$, tremem la $t(k)$

$$\begin{aligned} t_k &= \frac{\langle v^{(k)}, r^{(k-1)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle} = \frac{\langle r^{(k-1)} + s_{k-1}v^{(k-1)}, r^{(k-1)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle} \\ &= \frac{\langle r^{(k-1)}, r^{(k-1)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle} + s_{k-1} \frac{\langle v^{(k-1)}, r^{(k-1)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle}. \end{aligned}$$

dar $\langle v^{(k-1)}, r^{(k-1)} \rangle = 0$, atunci $t_k = \frac{\langle r^{(k-1)}, r^{(k-1)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle}$.

Si deci $x^{(k)} = x^{(k-1)} + t_k v^{(k)}$.



Metode Iterative

Metoda Gradientului Conjugat

Pe scurt $r^{(0)} = b - Ax^{(0)}$; $v^{(1)} = r^{(0)}$;

Pentru $k=1, 2, \dots, n$

$$t_k = \frac{\langle r^{(k-1)}, r^{(k-1)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle}, \quad x^{(k)} = x^{(k-1)} + t_k v^{(k)}, \quad r^{(k)} = r^{(k-1)} - t_k A v^{(k)}, \quad s_k = \frac{\langle r^{(k)}, r^{(k)} \rangle}{\langle r^{(k-1)}, r^{(k-1)} \rangle},$$

Si $v^{(k+1)} = r^{(k)} + s_k v^{(k)}$.



Metode iterative

Metoda Gradientului Conjugat

Stim ca $\langle r^{(k-1)}, v^{(i)} \rangle = 0$ pentru $i=1, 2, \dots, k-1$ si ne folosim de $r^{(k-1)}$ pentru a obtine vectorul directie $v(k)$

$$v^{(k)} = r^{(k-1)} + s_{k-1}v^{(k-1)}.$$

$$Av^{(k)} = Ar^{(k-1)} + s_{k-1}Av^{(k-1)}$$

$$\langle v^{(k-1)}, Av^{(k)} \rangle = \langle v^{(k-1)}, Ar^{(k-1)} \rangle + s_{k-1}\langle v^{(k-1)}, Av^{(k-1)} \rangle,$$

Si vrem ca ultima sa fie 0, deci rezulta s_{k-1}

$$s_{k-1} = -\frac{\langle v^{(k-1)}, Ar^{(k-1)} \rangle}{\langle v^{(k-1)}, Av^{(k-1)} \rangle}.$$



Metode Iterative

Metoda Gradientului Conjugat

Deci il avem pe $v(k)$, tremem la $t(k)$

Metode iterative

Metoda Gradientului Conjugat

Sau $Ax^{(k)} - b = Ax^{(k-1)} - b + t_k Av^{(k)}$ de unde rezulta

$$r^{(k)} = r^{(k-1)} - t_k Av^{(k)}.$$

$$\langle r^{(k)}, r^{(k)} \rangle = \langle r^{(k-1)}, r^{(k)} \rangle - t_k \langle Av^{(k)}, r^{(k)} \rangle = -t_k \langle r^{(k)}, Av^{(k)} \rangle.$$

si inlocuind t_k obtinem $\langle r^{(k-1)}, r^{(k-1)} \rangle = t_k \langle v^{(k)}, Av^{(k)} \rangle$,

$$\text{de unde } s_k = -\frac{\langle v^{(k)}, Ar^{(k)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle} = -\frac{\langle r^{(k)}, Av^{(k)} \rangle}{\langle v^{(k)}, Av^{(k)} \rangle} = \frac{(1/t_k) \langle r^{(k)}, r^{(k)} \rangle}{(1/t_k) \langle r^{(k-1)}, r^{(k-1)} \rangle} = \frac{\langle r^{(k)}, r^{(k)} \rangle}{\langle r^{(k-1)}, r^{(k-1)} \rangle}.$$



Metode Iterative

Metoda Gradientului Conjugat

Pe scurt $r^{(0)} = b - Ax^{(0)}$; $v^{(1)} = r^{(0)}$;

Pentru $k=1, 2, \dots, n$

Metode Iterative

Preconditionare

Daca matricea A este rau conditionata atunci metoda iterativa este supusa la erori de rotunjire. Metoda nu este aplicata direct matricei A ci unei alte matrici, tot SPD, dar cu un numar de conditionare mai mic.

Preconditionarea inlocuieste sistemul initial cu un altul, ce are aceiasi solutie dar care are conditii mai bune pentru convergenta.



Metode Iterative

Preconditionare

Ca sa pastram proprietatea de SPD, inmultim in ambele parti sistemul cu o matrice C^{-1} , nesingulara. Consideram $A \sim \tilde{A} = C^{-1}A(C^{-1})^t$ si speram ca are un numar de conditionare mai mic ca A.

Noul sistem va fi $\tilde{A}\tilde{x} = \tilde{b}$, cu $\tilde{x} = C^t x$ si $\tilde{b} = C^{-1}b$.

$$\tilde{A}\tilde{x} = (C^{-1}AC^{-t})(C^t x) = C^{-1}Ax.$$

Si rezolvam sistemul pentru $x \sim$ si apoi inmultiind cu $C^{-t} \equiv (C^{-1})^t$ obtinem x.



MN Primavara 2015 CS-UPB

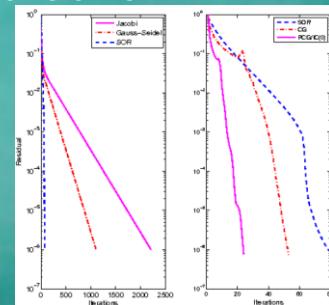
© 2015 Faculty of Computer Science and Technology, University Politehnica of Bucharest. All rights reserved.



MN Primavara 2015 CS-UPB

© 2015 Faculty of Computer Science and Technology, University Politehnica of Bucharest. All rights reserved.

Metode iterative



Bibliografie Metode Iterative SEL

- Chen Greif, Numerical Solution of Linear Systems, Department of Computer Science, The University of British Columbia, Vancouver B.C., Tel Aviv University, December 17, 2008
- Richard L. Burden, J. Douglas Faires, Numerical Analysis, 9th edition, Brooks/Cole, Cengage Learning, 2011;



MN Primavara 2015 CS-UPB

© 2015 Faculty of Computer Science and Technology, University Politehnica of Bucharest. All rights reserved.



MN Primavara 2015 CS-UPB

© 2015 Faculty of Computer Science and Technology, University Politehnica of Bucharest. All rights reserved.

Metode iterative pentru rezolvarea sistemelor de ecuații liniare: Jacobi, Gauss-Siedel, Suprarelaxare

Noțiuni teoretice

Metodele exacte de rezolvare a sistemelor de ecuații liniare, având complexitate $O(n^3)$, au aplicabilitate limitată la ordine de sisteme ce nu depășesc 1000. Pentru sisteme de dimensiuni mai mari se utilizează metode cu complexitate $O(n^2)$ într-un singur pas de iteratie. Acestea utilizează relații de recurență, care prin aplicare repetată furnizează aproximări, cu precizie controlată, a soluției sistemului.

Metodele iterative transformă sistemul $Ax = b$ în $x = Gx + c$. Pornindu-se cu o aproximare inițială $x^{(0)}$ a soluției, relația de recurență folosită are forma:

$$x^{(p+1)} = Gx^{(p)} + c$$

unde:

- $x^{(0)}, x^{(1)}, \dots, x^{(p)}, \dots$ sunt aproximările soluției;
- G reprezintă matricea de iteratie;
- c reprezintă vectorul de iteratie.

O metodă este convergentă dacă este stabilă și consistentă. Condiția necesară și suficientă de convergență este:

$$\rho(G) < 1$$

unde $\rho(G) = \max(|\lambda_1|, |\lambda_2|, \dots, |\lambda_n|)$ reprezintă raza spectrală a matricei de iteratie G și $\lambda_i, i = 1 : n$ reprezintă valorile proprii ale matricei.

Metodele iterative se bazează pe descompunerea matricei A sub forma $A = N - P$. Atunci sistemul devine:

$$(N - P)x = b, \text{ adică } x = N^{-1}Px + N^{-1}b.$$

Astfel, rezultă relația de recurență:

$$x^{(p+1)} = N^{-1}Px^{(p)} + N^{-1}b$$

de unde putem identifica $G = N^{-1}P$ și $c = N^{-1}b$.

Se partiziionează matricea A punând în evidență o matrice diagonală D , o matrice strict triunghiular inferioară L și o matrice strict triunghiular superioară U :

$$A = D - L - U.$$

Metoda Jacobi

În metoda Jacobi se aleg:

$$N = D$$

$$P = L + U$$

$$G_J = D^{-1}(L + U)$$

Soluția sistemului este:

$$x_i^{(p+1)} = \frac{b_i - \sum_{j=1, j \neq i}^n a_{ij}x_j^{(p)}}{a_{ii}}$$

Metoda Gauss-Seidel

La această metodă se aleg:

$$N = D - L$$

$$P = U$$

$$G_{GS} = (D - L)^{-1}U$$

Soluția sistemului este:

$$x_i^{(p+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(p+1)} - \sum_{j=i+1}^n a_{ij}x_j^{(p)}}{a_{ii}}$$

Observații:

1. Dacă matricea sistemului este diagonal dominantă pe linii, metoda Gauss Seidel este convergentă. Reciproca nu este adevarată.
2. O matrice A este diagonal dominantă pe linii dacă și numai dacă are următoarea proprietate: pentru fiecare linie i , modulul elementului de pe diagonala principală, $A(i, i)$ este strict mai mare decât suma modulelor elementelor de pe aceeași linie i .

Metoda suprarelaxării

Pentru găsirea unei descompuneri cât mai rapid convergente, se introduce un parametru de relaxare ω :

$$A = N - P = N - \omega N - P + \omega N = (1 - \omega)N - (P - \omega N) = N(\omega) - P(\omega)$$

de unde obținem:

$$N(\omega) = (1 - \omega)N$$

$$P(\omega) = P - \omega N$$

$$G(\omega) = N^{-1}(\omega)P(\omega) = \frac{N^{-1}}{1-\omega}(P - \omega N) = \frac{N^{-1}P - \omega I_n}{1-\omega}$$

Condiția de stabilitate impune $\omega \in (0, 2)$. În practică se face o altă alegere, astfel:

$$N(\omega) = \frac{1}{\omega}D - L, \quad P(\omega) = (\frac{1}{\omega} - 1)D + U, \quad G_\omega = (D - \omega L)^{-1}[(1 - \omega)D + \omega U]$$

Soluția sistemului se poate scrie sub forma:

$$x_i^{(p+1)} = \omega \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(p+1)} - \sum_{j=i+1}^n a_{ij}x_j^{(p)}}{a_{ii}} + (1 - \omega)x_i^{(p)}$$

Dacă se alege $\omega = 1 \Rightarrow$ metoda Gauss-Seidel.

Probleme rezolvate

Problema 1

Să se rezolve sistemul folosind metoda Gauss-Seidel:

$$\begin{cases} 7x_1 + 2x_2 - 4x_3 = 7 \\ 3x_1 + 6x_2 + 2x_3 = 15 \\ 2x_1 - 5x_2 + 8x_3 = 28 \end{cases}$$

Soluție:

Scriem formulele de recurență

$$\begin{cases} x_1^{(k+1)} = -2/7x_2^{(k)} + 4/7x_3^{(k)} + 7/7 \\ x_2^{(k+1)} = -3/6x_1^{(k+1)} - 2/6x_3^{(k)} + 15/6 \\ x_3^{(k+1)} = -2/8x_1^{(k+1)} + 5/8x_2^{(k+1)} + 28/8 \end{cases}$$

Dacă alegem $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 0$ obținem următoarele rezultate:

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0	0	0
1	1	2	4.5
2	3.00	-0.5	2.43
3	2.53	0.41	3.12
4	2.66	0.12	2.91
5	2.62	0.21	2.97

Soluția exactă este: $x_1 = 2.63$, $x_2 = 0.19$ $x_3 = 2.96$.

Problema 2

Folosiți metoda Jacobi pentru a aproxima soluția sistemului:

$$\begin{cases} 10x_1 - 5x_2 + x_3 = 1 \\ x_1 + 4x_2 + 3x_3 = 4 \\ 4x_1 - 3x_2 - 9x_3 = 6 \end{cases}$$

Soluție:

Scriem formulele de recurență

$$\begin{cases} x_1^{(k+1)} = 5/10x_2^{(k)} - 1/10x_3^{(k)} + 1/10 \\ x_2^{(k+1)} = -1/4x_1^{(k)} + 3/4x_3^{(k)} + 4/4 \\ x_3^{(k+1)} = 4/9x_1^{(k)} - 3/9x_2^{(k)} - 6/9 \end{cases}$$

Alegând $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 0 \Rightarrow$

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0	0	0
1	0.1	1.00	-0.66
2	0.66	1.47	-1.95
3	0.93	1.55	-0.86
4	0.96	1.41	-0.76
5	0.88	1.33	-0.71

Soluția exactă este: $x_1 = 0.84$, $x_2 = 1.34$, $x_3 = -0.73$.

Problema 3

Fie sistemul $Ax = b$, $A \in R^{2 \times 2}$, $x, b \in R^2$, $A = \begin{bmatrix} 2 & 2 \\ 1 & 3 \end{bmatrix}$. Matricea A nu este diagonal dominantă pe linii. În aceste condiții este convergentă metoda Gauss-Seidel?

Soluție:

Se determină matricea de iterare a sistemului pentru metoda Gauss-Seidel, G_{GS} .

$$A = D - L - U \Rightarrow D = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}, \quad L = \begin{bmatrix} 0 & 0 \\ -1 & 0 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & -2 \\ 0 & 0 \end{bmatrix}$$

Atunci:

$$G_{GS} = (D - L)^{-1}U = \begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix}^{-1} \begin{bmatrix} 0 & -2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 0 & \frac{1}{3} \end{bmatrix}.$$

$$\det(\lambda I - G_{GS}) = \begin{vmatrix} \lambda & 1 \\ 0 & \lambda - \frac{1}{3} \end{vmatrix} = 0 \Rightarrow \lambda(G_{GS}) = \{0, \frac{1}{3}\} \text{ și } \rho(G_{GS}) = \frac{1}{3} < 1.$$

\Rightarrow metoda Gauss-Seidel este convergentă.

Problema 4

Să se implementeze o funcție OCTAVE care rezolvă un sistem de ecuații liniare folosind metoda iterativă Gauss-Seidel. Date de intrare: A - matricea sistemului; b - vectorul termenilor liberi; x_0 - aproximația inițială a soluției; tol - precizia determinării soluției; $maxiter$ - numărul maxim de iterații. Date de ieșire: x - soluția sistemului; $succes$ - variabilă care indică convergența metodei.

Soluție:

```

1 function [x succes iter] = GaussSeidel(A, b, x0, tol, maxiter)
2 [n n]=size(A);
3 succes=0;
4 iter=maxiter;
5 x=zeros(n,1);
6
7 while maxiter > 0
8   maxiter--;
9
10  for i=1:n
11    suma=A(i,1:i-1)*x(1:i-1)+A(i,i+1:n)*x0(i+1:n);
12    x(i)=(b(i)-suma)/A(i,i);
13  endfor
14
15  if norm(x-x0)<tol
16    succes=1;
17    break;
18  endif
19  x0=x;
20 endwhile
21 iter=iter-maxiter;
22 endfunction

```

Listing 1: Algoritmul Gauss-Seidel.

Date de intrare:

$$A = \begin{bmatrix} 2 & 2 \\ 1 & 3 \end{bmatrix} \quad b = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad tol = 0.0001 \quad maxiter = 100.$$

Date de ieșire:

$$x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Probleme propuse

Problema 1

Fie sistemul $Ax = b$, $A \in R^{2 \times 2}$, $b \in R^2$, cu $A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$. Determinați raza spectrală a matricei de iterare Jacobi. Stabiliți convergența metodei Jacobi.

Problema 2

Fie sistemul liniar:

$$\begin{cases} 2x + y + z = 4 \\ x + 2y + z = 4 \\ x + y + 2z = 4 \end{cases}$$

Stabiliți a) dacă matricea este diagonal dominantă pe linii; b) convergența metodei Jacobi; c) convergența metodei Gauss-Seidel. Dacă metoda este convergentă, calculați soluția iterativă după trei pași. Alegeți voi aproximarea inițială.

Problema 3

Fie o matrice $A \in R^{n \times n}$ tridiagonală¹ și sistemul de ecuații $Ax = b$, cu $b, x \in R^n$. Scrieți o funcție OCTAVE care rezolvă sistemul de ecuații prin metoda Jacobi.

```
1 function x = solJacobi(A, b, x0, tol, maxiter)
2 % Rezolvarea sistemului Ax=b folosind metoda Jacobi
3 % Intrari:
4 %   A - matricea sistemului
5 %   b - vectorul termenilor liberi
6 %   x0 - aproximatie intiala a solutiei
```

¹<http://mathworld.wolfram.com/TridiagonalMatrix.html>

Metode iterative pentru rezolvarea sistemelor de ecuații liniare: Jacobi,
Gauss-Siedel, Suprarelaxare

```
7 % tol - precizia determinarii solutiei
8 % maxiter - numarul maxim de iteratii
9 % Iesiri:
10 % x - solutia sistemului
```

Listing 2: Algoritmul Jacobi.

Problema 4

Să se implementeze o funcție OCTAVE care rezolvă un sistem liniar de ecuații folosind metoda suprarelaxării.

```
1 function [x succes] = sor(A, b, x0, w, tol, maxiter)
2 % Metoda Suprarelaxarii
3 % Functia rezolva sisteme liniare Ax=b folosind metoda
4 % suprarelaxarii
5 % Input:
6 % A - matricea sistemului
7 % b - vectorul termenilor liberi
8 % x0 - aproximarea intiala a sistemului
9 % w - factorul de relaxare
10 % tol - toleranta
11 % maxiter - numarul maxim de iteratii
12 % Output:
13 % x - solutia sistemului
14 % succes - 0 = a fost gasita o solutie / 1 = metoda nu
15 % converge pentru maxiter
```

Listing 3: Algoritmul metode suprarelaxării.

Să se testeze funcția folosind diferite valori pentru ω .

Metode Numerice @ CS-UPB

Echipa MN CS - 2015:

Profesorii:

- Pantelimon George Popescu - Seria CA;
- Florin Pop - Seria CB, CC;

Asistenti:

PG Popescu, Clementin Cercel, Sorin Ciolofan, Bogdan Tiganoiaia, Diana Udrea, Alexandru Tifrea, David Iancu, Bogdan Cristian Marchis, Radu Poenaru, Madalina Grosu, Andrei Vlad Postoaca, Radu Constantinescu, Roxana Istrate, Madalina Hristache, Adelina Vidovici;



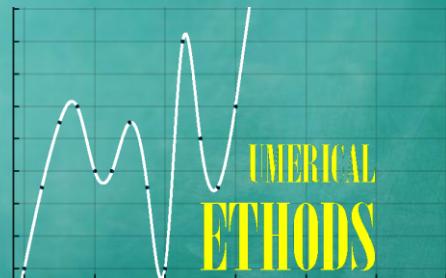
MN Primavara 2015 CS-UPB

© Craiovea131 (131) [Public domain], via Wikimedia Commons



13.10.15

Ecuatii neliniare



MN Primavara 2015 CS-UPB

© Craiovea131 (131) [Public domain], via Wikimedia Commons

Ecuatii neliniare

Metoda bisectiei

- Daca $f(x)=0$ se pune problema localizarii zerourilor acestei ecuatii;
- Se presupune ca f este continua pe un interval $[a,b]$, cu $f(a)f(b)$ negativ;
- Consideram ca avem un zero unic, p , in $[a,b]$, $f(p)=0$;
- Metoda imparte intervalul in doua jumatai si alege subintervalul pentru care functia la capete isi schimba semnul, adica intervalul care contine zeroul;
- Continua pana cand intervalul devine foarte mic;



MN Primavara 2015 CS-UPB

© Craiovea131 (131) [Public domain], via Wikimedia Commons



13.10.15

Ecuatii neliniare

Metoda bisectiei se opreste

- cand intervalul a devenit mai mic decat o anumita toleranta, $b-a<\text{tol}$ si deci zeroul $p=(a+b)/2$;
- cand $f(p)=0$ sau $f(p)<\text{epsilon}$;
- alte criterii;



MN Primavara 2015 CS-UPB

© Craiovea131 (131) [Public domain], via Wikimedia Commons



Ecuatii neliniare

Metoda bisectiei

- are dezvantajul convergentei slabe, adica numarul de pasi N devine destul de mare pana cand solutia de la pasul N se apropie de solutia ecuatiei, adica $|p_p-N|$ devine mic;
- are avantajul ca in orice situatie converge catre solutie;

Daca f este o functie continua pe un interval $[a,b]$ si $f(a)f(b)<0$. Metoda bisectiei genereaza un sir p_n care aproximeaza un zero p al lui f cu $|p_n-p|<(b-a)/2^n$, $n\geq 1$.



MN Primavara 2015 CS-UPB

© Craiovea131 (131) [Public domain], via Wikimedia Commons



13.10.15

Ecuatii neliniare

Puncte fixe (1900 Luitzen Egbertus Jan Brouwer (1881-1966), mat. olandez)

- Un punct fix p , pentru o functie g , este acel punct pentru care $g(p)=p$;
- Daca $f(p)=0$, adica p este zero pentru f , atunci putem defini o functie g care are punct fix in p , astfel $g(x)=x-f(x)$, sau mai general $g(x)=x+af(x)$;
- Invers daca p este punct fix pentru g , atunci functia $f(x)=x-g(x)$ are zero in p ;



MN Primavara 2015 CS-UPB

© Craiovea131 (131) [Public domain], via Wikimedia Commons



Ecuatii neliniare

Puncte fixe

Daca g este continua pe $[a,b]$ si $g(x)$ apartine $[a,b]$, oricare ar fi x , atunci g are cel putin un punct fix in $[a,b]$:

Mai mult, daca exista $g'(x)$ pe (a,b) si $k < 1$ astfel incat $|g'(x)| < k$, oricare ar fi x , atunci g are un unic punct fix in $[a,b]$:



Ecuatii neliniare

Puncte fixe

Daca g este continua pe $[a,b]$ si $g(x)$ apartine $[a,b]$, oricare ar fi x , si daca exista $g'(x)$ pe (a,b) si $0 < k < 1$ astfel incat $|g'(x)| < k$, oricare ar fi x , atunci pentru orice p_0 , sirul p_n , $p_n = g(p_{n-1})$ converge catre un unic punct fix p din $[a,b]$.



Ecuatii neliniare

Metoda Newton-Raphson

$$f(p) = f(p_0) + (p - p_0)f'(p_0) + \frac{(p - p_0)^2}{2}f''(\xi(p))$$

cu $\text{eps}(p)$ intre p si p_0 , iar cum $f(p)=0$, avem

$$0 = f(p_0) + (p - p_0)f'(p_0) + \frac{(p - p_0)^2}{2}f''(\xi(p))$$

iar pentru ca $(p - p_0)^2$ este aprox 0 avem

$$0 \approx f(p_0) + (p - p_0)f'(p_0)$$



Ecuatii neliniare

Puncte fixe

- Ca sa apoximam un punct fix, pornim de la o aproximare initiala p_0 si construim un sir p_n , cu $p_n = g(p_{n-1})$, $n \geq 1$;
- Daca p_n converge si g este continua pe $[a,b]$ atunci

$$p = \lim_{n \rightarrow \infty} p_n = \lim_{n \rightarrow \infty} g(p_{n-1}) = g\left(\lim_{n \rightarrow \infty} p_{n-1}\right) = g(p)$$



Ecuatii neliniare

Ecuatii neliniare

Metoda **Newton-Raphson** (Isaac Newton (1642-1727) fizician si matematician englez, Joseph Raphson (1648-1715) matematician englez)

Fie f o functie continua si derivabila pe $[a,b]$, fie p_0 o aproximare a lui p astfel incat $f(p_0) \neq 0$ si $|p - p_0|$ sa fie mic. Consideram primul polinom **Taylor** (Brook Taylor (1685-1731) extins dupa p_0 si evaluat in p , astfel:



Ecuatii neliniare

Metoda Newton-Raphson

si de aici rescriem

$$p \approx p_0 - \frac{f(p_0)}{f'(p_0)} \equiv p_1$$

ceea ce ne ofera pasul recurentei

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})} \quad \text{pentru } n \geq 1;$$



Ecuatii neliniare

Metoda Newton-Raphson

- nu converge daca la un anumit pas $f'(p_n) = 0$;
- nu converge daca p_0 nu este ales aproape de p , astfel incat termenul $(p - p_0)^2$ sa poate fi neglijat;

Fie $f \in C^2[a, b]$. Daca $p \in (a, b)$ cu $f(p) = 0$ si $f'(p) \neq 0$, atunci exista $\delta > 0$ astfel incat metoda genereaza un sir p_n , $n \geq 1$, care converge la p pentru orice aprox. initiala $p_0 \in [p - \delta, p + \delta]$.



Ecuatii neliniare

Metoda Newton-Raphson

Metoda Secantei

si inlocuind in expresia recurrentiei NR, obtinem

$$p_n = p_{n-1} - \frac{f(p_{n-1})(p_{n-1} - p_{n-2})}{f(p_{n-1}) - f(p_{n-2})}$$



Ecuatii neliniare

Metoda Newton-Raphson

Ordin de convergenta

Ordinul de convergenta se poate deduce si din dezvoltarea in serie Taylor.

Daca $x_{i+1} = g(x_i)$ este o metoda iterativa care solutioneaza ecuatia $x = g(x)$, iar s este solutia exacta si x_n solutia aproximativa la pasul n , atunci daca $s + \epsilon_n = x_n$, eroarea la pasul n si daca consideram ca g este derivabila de multe ori, avem

$$x_{n+1} = g(x_n) = g(s + \epsilon_n) = g(s) + \epsilon_n g'(s) + \frac{1}{2} \epsilon_n^2 g''(s) + \dots$$



Ecuatii neliniare

Metoda Newton-Raphson

Metoda Secantei

Pentru ca de multe ori nu putem calcula derivata unei functii, atunci o putem aproxima prin

$$f'(p_{n-1}) = \lim_{x \rightarrow p_{n-1}} \frac{f(x) - f(p_{n-1})}{x - p_{n-1}}$$

si daca p_{n-2} este apropiat de p_{n-1} , avem

$$f'(p_{n-1}) \approx \frac{f(p_{n-2}) - f(p_{n-1})}{p_{n-2} - p_{n-1}} = \frac{f(p_{n-1}) - f(p_{n-2})}{p_{n-1} - p_{n-2}}$$



Ecuatii neliniare

Metoda Newton-Raphson

Metoda Secantei

si inlocuind in expresia recurrentiei NR, obtinem

$$p_n = p_{n-1} - \frac{f(p_{n-1})(p_{n-1} - p_{n-2})}{f(p_{n-1}) - f(p_{n-2})}$$



Ecuatii neliniare

Metoda Newton-Raphson

Ordin de convergenta

Fie x_n un sir care converge catre s ; $\epsilon_n = |s - x_n|$ si $\epsilon_{n+1} = |s - x_{n+1}|$ erorile la pasul n si $n+1$; Daca exista A si $R > 0$, iar

$$\lim_{n \rightarrow \infty} \frac{|s - x_{n+1}|}{|s - x_n|^R} = \lim_{n \rightarrow \infty} \frac{|\epsilon_{n+1}|}{|\epsilon_n|^R} = A$$

atunci x_n converge catre s cu **ordinul de convergenta R** . A se numeste constanta eroare asimptotica.



Ecuatii neliniare

Metoda Newton-Raphson

Ordin de convergenta

Ordinul de convergenta se poate deduce si din dezvoltarea in serie Taylor.

Daca $x_{i+1} = g(x_i)$ este o metoda iterativa care solutioneaza ecuatia $x = g(x)$, iar s este solutia exacta si x_n solutia aproximativa la pasul n , atunci daca $s + \epsilon_n = x_n$, eroarea la pasul n si daca consideram ca g este derivabila de multe ori, avem

$$x_{n+1} = g(x_n) = g(s + \epsilon_n) = g(s) + \epsilon_n g'(s) + \frac{1}{2} \epsilon_n^2 g''(s) + \dots$$



Ecuatii neliniare

Metoda Newton-Raphson

Ordin de convergenta

Si deci exponentul lui ϵ_n din primul termen diferit de zero este ordinul de convergenta al metodei it. $x_n = g(x_{n-1})$.

Cum $\epsilon_{n+1} = \epsilon_n g'(s) + \frac{1}{2} \epsilon_n^2 g''(s) + \dots$, iar $g(s) = s$, avem

$$\epsilon_{n+1} = \epsilon_n g'(s) + \frac{1}{2} \epsilon_n^2 g''(s) + \dots$$



Ecuatii neliniare

Ordin de convergenta pentru Metoda Newton-Raphson

$$g(x) = x - \frac{f(x)}{f'(x)} \quad g'(x) = \frac{f(x) + f''(x)}{(f'(x))^2} \quad g''(x) = \frac{2f''(x)}{f'(x)}$$

Deci $g'(s)=0$ deoarece $f(s)=0$, atunci din dezvoltarea in serie Taylor vom avea

$$\epsilon_{n+1} = \epsilon_n \cancel{g'(s)} + \frac{1}{2} \epsilon_n^2 g''(s) + \dots$$

De unde rezulta ca metoda Newton-Raphson are ordinul de convergenta 2.



Ecuatii neliniare

Fie p ordinul de convergenta si daca in plus consideram d ca fiind numarul de evaluari de functii la fiecare pas atunci definim indexul de eficienta al unei metode iterative ca fiind

$$\rho = p^{\frac{1}{d}}$$

Atunci o metoda de ordin $p=2^n$, se considera a fi optima*, dpd al indexului de eficienta daca are maxim $n+1$ evaluari de functii.

* H.T. Kung, J.F. Traub, Optimal order of one-point and multipoint iterations, J. Assoc. Comput. Mach., 21 (1974), pp. 643-651.



Sisteme de ecuatii neliniare

Puncte fixe pentru functii de mai multe var.

Un sistem neliniar de ecuatii se prezinta astfel

$$\begin{aligned} f_1(x_1, x_2, \dots, x_n) &= 0, \\ f_2(x_1, x_2, \dots, x_n) &= 0, \\ &\vdots && \vdots \\ f_n(x_1, x_2, \dots, x_n) &= 0, \end{aligned}$$

sau $\mathbf{F}(x)=0$, unde

$$\mathbf{F}(x_1, x_2, \dots, x_n) = (f_1(x_1, x_2, \dots, x_n), f_2(x_1, x_2, \dots, x_n), \dots, f_n(x_1, x_2, \dots, x_n))^T$$



Sisteme de ecuatii neliniare

Puncte fixe pentru functii de mai multe var.

Fie $f : D \subset \mathbb{R}^n \rightarrow \mathbb{R}$ si $x_0 \in D$. Daca toate derivatele partiale ale lui f exista si mai exista si constantele $\delta > 0$ si $K > 0$ daca pentru oricare x cu $|x - x_0| < \delta$, $x \in D$, avem

$|\partial f(x)/\partial x_j| \leq K$, $j = 1, 2, \dots, n$. Atunci f este continua in x_0 .

O functie $G: \mathbb{R}^n \rightarrow \mathbb{R}^n$ are un punct fix p daca $G(p)=p$;



Sisteme de ecuatii neliniare

Puncte fixe pentru functii de mai multe var.

Fie $D = \{(x_1, x_2, \dots, x_n)^T | a_i \leq x_i \leq b_i, i = 1, 2, \dots, n\}$ pentru o colectie de constante a_1, a_2, \dots, a_n si b_1, b_2, \dots, b_n . Daca $G: D \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$ este o functie continua, cu proprietatea ca $G(x) \in D$ oricare ar fi $x \in D$. Atunci G are un punct fix in D .



Sisteme de ecuatii neliniare

Puncte fixe pentru functii de mai multe var.

Mai mult, daca toate functiile componente ale lui G au derivate partiale continue si exista o constanta $K < 1$ cu

$$\left| \frac{\partial g_i(x)}{\partial x_j} \right| \leq \frac{K}{n}$$

atunci sirul x_k definit prin $x_k = G(x_{k-1})$ ce porneste de la un x_0 initial din D , este convergent catre un unic fix punct din D , p , iar

$$\|x^{(k)} - p\|_\infty \leq \frac{K^k}{1-K} \|x^{(1)} - x^{(0)}\|_\infty$$



Sisteme de ecuatii neliniare

Metoda Newton

Se rescrie sistemul sub forma $\mathbf{G}(\mathbf{x}) = \mathbf{x} - J(\mathbf{x})^{-1}\mathbf{F}(\mathbf{x})$
unde $J(\mathbf{x})$ este matricea **Jacobiana** (a aparut prima oara in 1815 fiind folosita de Augustin-Louis Cauchy(1789-1857) mat. francez, dar in 1841 Jacobi a scris despre ea si despre proprietatile sale)



MN Primavara 2015 CS-UPB
© Craiovea Mihai Iulian / Flickr / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs

Sisteme de ecuatii neliniare

Metoda Newton

De obicei se evita calculul inversei lui J si atunci se gaseste un vector y astfel incat

$\mathbf{J}y = -\mathbf{F}$ si apoi solutia va fi data de suma dintre y si x la pasul anterior.

Cu alte cuvinte in fiecare pas avem de solutionat un SEL in y , unde matricea coeficientilor este J , iar vectorul termenilor liberi, $-\mathbf{F}$



MN Primavara 2015 CS-UPB
© Craiovea Mihai Iulian / Flickr / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs

Sisteme de ecuatii neliniare

Metoda Newton

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{bmatrix}$$

de unde si recurrenta metodei

$$\mathbf{x}^{(k)} = \mathbf{G}(\mathbf{x}^{(k-1)}) = \mathbf{x}^{(k-1)} - J(\mathbf{x}^{(k-1)})^{-1}\mathbf{F}(\mathbf{x}^{(k-1)})$$



MN Primavara 2015 CS-UPB
© Craiovea Mihai Iulian / Flickr / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs

Bibliografie Ecuatii neliniare

- Richard L. Burden, J. Douglas Faires, Numerical Analysis, 9th edition, Brooks/Cole, Cengage Learning, 2011;



MN Primavara 2015 CS-UPB
© Craiovea Mihai Iulian / Flickr / CC-BY-NC-ND / Attribution-NonCommercial-NoDerivs

**Soluția ecuației neliniare $f(x) = 0$.
Rădăcinile polinoamelor. Lucrul cu
polinoame în Octave. Rezolvarea
sistemelor de ecuații neliniare**

Notiuni teoretice

Soluția ecuației neliniare $f(x) = 0$

Vom studia două tipuri de metode:

- a) Metode bazate pe interval

Se pornește de la observația că o funcție își schimbă semnul în vecinătatea unei rădăcini. Inițial, se consideră două valori de o parte și de cealaltă a rădăcinii, apoi se micșorează acest interval care încadrează rădăcina până ce se ajunge la rădăcină, cu o anumită precizie. Metodele de acest tip sunt întotdeauna convergente către soluție deoarece aplicarea repetată a algoritmului duce la o estimare mai precisă a rădăcinii.

În continuare vom detalia două metode bazate pe interval: metoda bisecției și metoda secantei.

Metoda bisecției

Dacă o funcție își schimbă semnul pe un interval, adică $f(a) \cdot f(b) < 0$, atunci se evaluează valoarea funcției în punctul de mijloc al intervalului, $c = \frac{a+b}{2}$. Localizarea rădăcinii se estimează ca fiind la mijlocul subintervalului în care funcția își schimbă semnul, noul subinterval având la unul dintre capete pe c . Procedura se repetă până ce se obțin estimări mai precise ale rădăcinii.

Se poate defini eroarea relativă procentuală folosind relația:

$$\epsilon = \left| \frac{x_r^{new} - x_r^{old}}{x_r^{new}} \right|$$

Când $\epsilon < prag$, algoritmul iterativ se termină, iar x_r^{new} este considerată valoarea calculată a rădăcinii.

Un alt avantaj al acestei metode este faptul că pentru o eroare acceptată tol , se poate calcula numărul de iterații ce trebuie parcursă pentru a se ajunge la aproximarea dorită a rădăcinii, conform formulei:

$$n = \log_2\left(\frac{b-a}{tol}\right).$$

Metoda secantei

În această metodă, aproximarea rădăcinii, în intervalul $[a_i, b_i]$, se va considera a fi intersecția dreptei care trece prin punctele $(a_i, f(a_i))$ și $(b_i, f(b_i))$ cu axa Ox , adică:

$$x_{i+1} = \frac{a_i f(b_i) - b_i f(a_i)}{f(b_i) - f(a_i)}.$$

Apoi, se va considera subintervalul care are la unul dintre capete pe x_{i+1} , în manieră similară cu metoda bisecției. Același criteriu de oprire: eroarea relativă procentuală ϵ , poate fi folosit.

b) Metode care nu se bazează pe un interval, în care este suficientă cunoașterea unei valori inițiale x_i care este folosită mai departe pentru estimarea valorii următoare x_{i+1} . Aceste metode, spre deosebire de primele, pot fi convergente sau pot fi divergente. Atunci când ele converg, convergența este mult mai rapidă decât în cazul metodelor bazate pe interval. Pentru această categorie, menționăm metoda tangentei (Newton-Raphson) și metoda aproximățiilor succesive (contractie).

Metoda tangentei

După cum s-a menționat anterior, se pornește cu o valoare de început x_i , apoi se deduce o estimare îmbunătățită x_{i+1} . În cazul metodei tangentei, se ducă o tangentă la curba din punctul de coordonate $[x_i, f(x_i)]$. Punctul de intersecție a tangentei cu Ox se va considera x_{i+1} .

Relația de recurență este:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Se poate arăta că eroarea la iterația curentă este proporțională cu pătratul erorii la iterația precedentă, ceea ce, aproximativ, înseamnă că la fiecare iterație numărul de zecimale corect calculate din rădăcină se dublează.

Metoda aproximățiilor successive

În cazul acestei metode, ecuația $f(x) = 0$ se scrie ca $x = g(x)$, ceea ce are avantajul de a furniza o formulă prin care se poate calcula o nouă valoare a lui x ca o funcție g aplicată vechii valori a lui x , adică:

$$x_{i+1} = g(x_i)$$

Această metodă converge liniar (eroarea la pasul i este proporțională cu eroarea la pasul $i - 1$ inmulțită cu un factor subunitar) dacă $|g'(x)| < 1, \forall x \in (a, b)$. Altfel, metoda este divergentă.

Rădăcinile polinoamelor

Fie p un polinom de grad n , $p(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \dots + a_1 x + a_0$, cu $a_n \neq 0$. Conform cu *teorema fundamentală a algebrei*, polinomul p are n rădăcini reale sau complexe (numărând și multiplicitatele). În cazul în care coeficienții a_i sunt toți reali, rădăcinile complexe apar conjugate (de forma $c + id$ și $c - id$).

Folosind regula semnelor a lui Descartes, putem număra câte rădăcini reale pozitive are polinomul p . Fie v numărul variațiilor de semn ale coeficienților

$$a_n, a_{n-1}, a_{n-2}, \dots, a_1, a_0,$$

ignorând coeficienții care sunt nuli. Fie n_p numărul de rădăcini pozitive. Avem următoarele două relații:

- 1) $n_p \leq v$;
- 2) $v - n_p$ este un număr par.

Analog, numărul de rădăcini reale negative ale lui $p(x)$ se obține folosind numărul de schimbări de semn ale coeficienților polinomului $p(-x)$. Pentru determinarea rădăcinilor se pot aplica metodele descrise anterior la punctul b), dacă nu se cunoaște localizarea rădăcinilor pe interval.

Lucrul cu polinoame în Octave

În Octave, un polinom este reprezentat prin coeficienți (în ordine descrescătoare). Vectorul

```
> p = [-2, -1, 0, 1, 2]
```

reprezintă polinomul $-2x^4 - x^3 + x + 2$.

Funcția *polyout* generează o reprezentare a funcției de o variabilă dată (de exemplu x)

```
> polyout(p, 'x')
```

are ca rezultat $-2 * x^4 - 1 * x^3 + 0 * x^2 + 1 * x^1 + 2$.

Evaluarea unui polinom:

```
> y = polyval(p, x)
```

returnează $p(x)$. Dacă x este vector sau matrice, polinomul este evaluat în fiecare dintre elementele lui x .

Înmulțirea a două polinoame:

```
> r = conv(p, q)
```

returnează un vector r care conține coeficienții produsului dintre p și q .

Împărțirea a două polinoame:

```
> [b, r] = deconv(y, a)
```

returnează coeficienții polinoamelor b și r a.î. $y = ab + r$, unde b este câtul și r este restul împărțirii.

Rădăcinile unui polinom:

```
> roots(p)
```

returnează un vector ce conține toate rădăcinile polinomului p .

Derivata unui polinom:

```
> q = polyder(p)
```

returnează un vector ce conține coeficienții derivatei polinomului p .

Integrarea unui polinom:

```
> q = polyint(p)
```

returnează un vector ce conține coeficienții rezultați din integrarea polinomului p .

Polinomul de interpolare de gradul n care aproximează setul de date (x, y) :

```
> p = polyfit(x, y, n)
```

Exemplu: Adunarea polinoamelor

Presupunem că dorim să adunăm polinoamele $p(x) = x^2 - 1$ și $q(x) = x + 1$. Următoarea încercare va da eroare:

```
> p = [1, 0, -1];
> q = [1, 1];
> "p + q error: operator +: nonconformant arguments
  (op1 is 1x3, op2 is 1x2)
error: evaluating binary operator '+' near line 22, column 3"
```

Operația de adunare a doi vectori de dimensiuni diferite în Octave dă eroare. Pentru a evita aceasta, trebuie adăugate zerouri la q .

```
> q = [0, 1, 1];
> p + q
ans = 1 1 0
> polyout(ans, 'x')
1*x^2 + 1*x^1 + 0
```

Sisteme de ecuații neliniare

Un sistem de ecuații neliniare are forma:

$$\begin{cases} f_1(x_1, x_2, x_3, \dots, x_{n-1}, x_n) = 0 \\ f_2(x_1, x_2, x_3, \dots, x_{n-1}, x_n) = 0 \\ \dots \\ f_n(x_1, x_2, x_3, \dots, x_{n-1}, x_n) = 0 \end{cases}$$

unde f_i reprezintă funcții cunoscute de n variabile x_1, x_2, \dots, x_n , presupuse continue, împreună cu derivatele lor parțiale până la un ordin convenabil (de obicei, până la ordinul doi). Se va urmări găsirea soluțiilor reale ale sistemului într-un anumit domeniu de interes, domeniu în care se consideră valabile proprietățile de continuitate impuse funcțiilor f_i și derivatelor lor. Rezolvarea sistemului este un proces iterativ în care se pornește de la o aproximare inițială pe care algoritmul o va îmbunătăți până ce se va îndeplini o condiție de convergență. În cazul de fată, localizarea apriori a soluției nu mai este posibilă (nu există o metodă analoagă metodei înjumătățirii intervalor).

Metoda Newton

Pentru simplificarea notației, considerăm $F = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \dots \\ f_n \end{pmatrix}$ și $x = (x_1, x_2, \dots, x_n)$.

Sistemul îl putem scrie ca $F(x) = 0$. Notăm cu $x^{(k)}$ estimarea la pasul k a soluției x^* , deci $F(x^*) = 0$.

Se poate deduce relația:

$$x^{(k+1)} = x^{(k)} - J^{-1}F(x^{(k)}), k = 0, 1, 2, \dots$$

unde J este matricea *Jacobiană*

$$J = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \dots & \frac{\partial F_1}{\partial x_n} \\ \dots & \dots & \dots \\ \frac{\partial F_n}{\partial x_1} & \dots & \frac{\partial F_n}{\partial x_n} \end{pmatrix}$$

Dacă matricea J este neinversabilă, atunci pasul este nedefinit. Vom presupune că $J(x^*)$ este inversabilă, iar continuitatea lui J va asigura că $J(x^{(k)})$ este inversabilă pentru orice $x^{(k)}$ suficient de apropiat de x^* . Secvența definită iterativ converge spre soluția x^* . Condiția de oprire la iterația k , $\|x^* - x^{(k)}\| < tol$, unde tol este o toleranță dată, se poate arăta că revine la $\|x^{(k)} - x^{(k-1)}\| < tol$.

Probleme propuse

Problema 1

Scripteți o funcție Octave care să rezolve o ecuație neliniară folosind metoda bisecției. Parametrii pe care îi primește funcția sunt: a, b (capetele intervalului în care se află soluția reală), tol (toleranța), respectiv un handler la funcția care definește ecuația neliniară de rezolvat.

Problema 2

Scripteți o funcție Octave care să rezolve o ecuație neliniară folosind metoda secantei. Parametrii pe care îi primește funcția sunt: a, b (capetele intervalului în care se află soluția reală), tol (toleranța), respectiv un handler la funcția care definește ecuația de rezolvat.

Problema 3

Scripteți o funcție Octave care să rezolve o ecuație neliniară prin metoda tangentei. Parametrii pe care îi primește funcția sunt: x_0 (aproximația inițială a soluției), tol (toleranța), respectiv un handler la funcția care definește ecuația de rezolvat și un handler la derivata acesteia.

Problema 4

Scripteți o funcție care să rezolve o ecuație neliniară prin metoda aproximățiilor succesive. Parametrii pe care îi primește funcția sunt: x_0 (aproximația inițială a soluției), tol (toleranța), respectiv un handler la funcția care definește ecuația de rezolvat.

Problema 5

Să se determine tipul (dacă sunt reale pozitive sau negative, complexe) rădăcinilor polinomului $p(x) = 3x^6 + x^4 - 2x^3 - 5$.

Problema 6

Să se scrie în Octave un program care rezolvă un sistem de ecuații neliniare prin metoda Newton. Ca intrare, se consideră un vector coloană care reprezintă $x^{(0)}$, un pointer (handler) la o funcție care evaluează F într-un vector generic x , un pointer la o funcție care calculează Jacobiana într-un vector generic x , o toleranță dată ϵ . Metoda se oprește atunci când $\|x^{(k)} - x^{(k-1)}\| < \epsilon$ și returnează vectorul soluție x^* și numărul de iterații n care au fost necesare pentru producerea soluției.