# Temă la disciplina
# Analiza Algoritmilor
# Rezolvarea sistemelor de ecuații liniare
# Etapa 1

Buzea Alexandru-Mihai-Iulian, grupa 321CA
alexandru.buzea2007@stud.acs.upb.ro

Facultatea de Automatică și Calculatoare
Calculatoare și Tehnologia Informației

**Abstract.** Prezentarea unor metode folosite pentru soluționarea sistemelor de ecuații liniare, precum și a unor avantaje/dezavantaje ale acestora.

**Keywords:** GPP · GPPS · Householder · Conjugate Gradient

## 1 Introducere

### 1.1 Descrierea problemei rezolvate

Sistemele de ecuații liniare reprezintă una dintre cele mai importante teme din analiza numerică, cu aplicații extrem de răspândite în Computer Science. Majoritatea proceselor din natură ce se doresc a fi modelate cu ajutorul aplicațiilor software pot fi descrise foarte precis prin rezolvarea sistemelor de ecuații liniare. De asemenea, multe modele neliniare sau dinamice (descrise prin ecuații diferențiale) pot fi descrise local sau global cu ajutorul sistemelor de ecuații liniare.
La modul general, un sistem de ecuații liniare are $m$ ecuații și $n$ necunoscute. În forma sa desfășurată, acesta se scrie sub următoarea formă:

$$\begin{cases} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \cdots + a_{1n} \cdot x_n = b_1 \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + \cdots + a_{2n} \cdot x_n = b_2 \\ a_{31} \cdot x_1 + a_{32} \cdot x_2 + \cdots + a_{3n} \cdot x_n = b_3 \\ \cdots \\ a_{m1} \cdot x_1 + a_{m2} \cdot x_2 + \cdots + a_{mn} \cdot x_n = b_m \end{cases} \tag{1}$$

În mod similar, un sistem de ecuații liniare se poate scrie și în forma sa matriceală (mai compactă):

$$A \cdot x = b \tag{2}$$

unde $A \in \mathbb{R}^{m \times n}$ este matricea coeficienților (matricea sistemului), $x \in \mathbb{R}^{n \times 1}$ este vectorul necunoscutelor, iar $b \in \mathbb{R}^{m \times 1}$ este vectorul termenilor liberi.

În domeniul metodelor numerice, se întâlnesc în majoritatea situațiilor probleme bine formulate: sisteme de ecuații liniare în care matricea $A$ este pătratică (sistem bine dimensionat) și nesingulară, caz în care vectorul necunoscutelor se poate calcula prin metoda trivială:

$$x = A^{-1} \cdot b \tag{3}$$

Principala problemă întâmpinată în acest caz este instabilitatea numerică și efortul computațional substanțial adus pentru a calcula inversa unei matrice ($O(n^3)$ pentru metoda Gauss-Jordan, cea mai folosită pentru inversarea de matrice). De aceea, este preferată folosirea unor metode exacte de rezolvare a sistemelor de ecuații (cum ar fi eliminarea gaussiană și factorizarea QR cu ajutorul transformărilor ortogonale Householder) sau a unor metode iterative (cum ar fi metoda gradientului, Gradient Descent sau metoda gradientului conjugat), care ne furnizează o soluție cât mai apropiată de cea reală.

### 1.2 Exemple de aplicații practice pentru problema aleasă

#### Aplicație practică 1: Rezolvarea circuitelor electrice

Primul exemplu de aplicație practică a rezolvării sistemelor de ecuații liniare îl reprezintă rezolvarea de circuite electrice și electronice reale în cadrul unor simulatoare, cum ar fi **LTSpice XVII**. Acest simulator permite folosirea unor elemente de circuit cum ar fi: surse de tensiune/curent, rezistori, bobine, condensatoare, diode (cu o mare varietate de modele matematice pentru fiecare element în parte). În cazul simplu al unui circuit de curent continuu, cu elemente liniare de circuit, circuitul se poate reprezenta ca un graf orientat, în care nodurile circuitului corespund nodurilor din graf, iar fiecare latură este caracterizată de tensiunea sursei de tensiune (dacă aceasta există), curentul sursei de curent (dacă aceasta există), rezistența echivalentă etc. Apoi, folosind metoda potențialelor nodurilor, obținem un sistem de ecuații liniare cu $n - 1$ ecuații, unde $n$ este numărul de noduri (un nod este considerat cu potențial 0). Acest sistem poate fi rezolvat (mai eficient) prin orice metodă exactă sau iterativă cunoscută.

#### Aplicație practică 2: Predicție bazată pe regresie liniară

Una dintre cele mai folosite metode de predicție în Machine Learning pentru output care poate lua orice valoare reală este regresia liniară. Spre exemplu, dorim să estimăm costul unui articol de îmbrăcăminte. Asociem cu un articol de îmbrăcăminte generic mai multe **calități/features**, cum ar fi mărimea, tipul de articol etc. Fiecare feature va contribui în mod semnificativ (sau nu) la costul articolului nostru, de aceea prețul unui articol va fi dat de o combinație liniară de valorile acestor calități/features luate în considerare, plus un termen constant (**bias**). Dacă notăm cu $X \in \mathbb{R}^{m \times (n+1)}$ [1] matricea tuturor calităților ale tuturor

[1] Apare n + 1 datorită faptului că avem n features pentru fiecare exemplu, plus un feature constant (bias). Astfel, matricea X va avea pe prima coloană numai 1.

exemplelor de antrenament ale modelului nostru de predicție, cu $\theta \in \mathbb{R}^{n+1}$ vectorul greutăților (weights) și cu $y \in \mathbb{R}^m$ prețurile reale pentru exemplele de antrenament, noi ne dorim să minimizăm următoarea funcție de cost:

$$J(\theta) = \frac{1}{2} \cdot \|X \cdot \theta - y\|^2 \tag{4}$$

Se poate demonstra analitic că funcția de cost dată de formula de mai sus are un minim global, dat de următoarea ecuație matriceală:

$$X^T X \cdot \theta = X^T y \tag{5}$$

Relația de mai sus reprezintă un sistem de ecuații liniar, în care matricea sistemului este $A = X^T X$, iar vectorul termenilor liberi este $X^T y$. În situația cea mai probabilă ca cele $n+1$ calități/features să fie liniar independente, matricea $X^T X$ este inversabilă, deci sistemul nostru are soluție unică.

Un mare avantaj al acestui sistem este însă faptul că matricea sistemului $A$ este **simetrică și pozitiv-definită**. Astfel, sistemul nostru de ecuații poate fi rezolvat cu ajutorul metodei **gradientului conjugat**, ajungându-se la soluția exactă în exact $n$ pași.

**Aplicație practică 3: Rezolvarea sistemelor de ecuații neliniare**

Sistemele neliniare de ecuații pot fi descrise generic de o funcție de forma $F : \mathbb{R}^n \to \mathbb{R}^n$. Rezolvarea sistemului de ecuații neliniare presupune a găsi un vector $x \in \mathbb{R}^n$ astfel încât:

$$F(x) = 0 \tag{6}$$

Pentru rezolvarea unui astfel de sistem de ecuații (în anumite condiții), se poate folosi metoda iterativă Newton, astfel: se pleacă de la o aproximație inițială $x_0$ a soluției sistemului și se folosește următoarea recurență (similară cu metoda lui Newton pentru rezolvarea ecuațiilor neliniare):

$$x_{k+1} = x_k - J(x_k)^{-1} \cdot F(x_k) \tag{7}$$

unde $J(x_k)$ este matricea jacobiană corespunzătoare transformării $F$ în punctul $x_k$. Relația de mai sus poate fi adusă la forma:

$$J(x_k) \cdot (x_k - x_{k+1}) = F(x_k) \tag{8}$$

Relația de mai sus reprezintă un sistem de ecuații liniare, rezolvabil pentru a putea astfel determina o aproximare mai bună a soluției sistemului.

### 1.3 Specificarea soluțiilor alese

Pentru analiză, am ales patru algoritmi de rezolvare a sistemelor de ecuații liniare: eliminarea gaussiană (în cele două variante principale - eliminarea gaussiană cu pivotare parțială (GPP) și eliminarea gaussiană cu pivotare parțială cu pivot scalat (GPPS)), factorizarea QR folosind transformări ortogonale Householder și metoda gradientului conjugat/Conjugate Gradient.

### 1.4 Criteriile de evaluare pentru soluția propusă

Ne propunem să avem un număr de aproximativ 50 de teste generate cu ajutorul GNU Octave (limbaj care va fi folosit și pentru implementarea algoritmilor), distribuite astfel:

- 20 de teste pe sisteme pozitiv definite și simetrice de ecuații, de dimensiuni mari (de ordinul sutelor), tocmai pentru a arăta avantajul metodei gradientului comjugat în fața celorlalte metode. Aceste teste sunt pe sisteme bine definite (matricea sistemului este pătratică și se garantează că sistemul nostru are soluție).
- 30 de teste pe matrice normale. Vom avea grijă ca sistemele generate să aibă dimensiuni din ce în ce mai mari, astfel că putem testa eficacitatea sistemelor noastre și pe sisteme de ecuații mai mari, de dimensiuni mai relevante pentru aplicațiile practice studiate.

Vom rula metodele exacte pe toate testele create, iar metoda gradientului conjugat va fi rulată doar pe cele 20 de teste (menționate anterior) din totalul de 50.

Verificarea de corectitudine și de precizie va fi făcută folosind operatorul existent în sintaxa limbajului Octave și care permite rezolvarea sistemelor de ecuații folosind cele mai eficiente metode, astfel:

$$A \cdot x = b \Longrightarrow x = A \backslash b \tag{9}$$

Astfel, putem compara soluția dată de metoda folosită cu soluția returnată prin intermediul acestui operator.

Principalele metrici pentru compararea algoritmilor de rezolvare a sistemelor de ecuații vor fi timpul de execuție (măsurabil în environment-ul din Octave) și corectitudinea răspunsului (eroarea în raport cu cea mai bună soluție; se poate folosi una din cele variante de calcul ale erorii de mai jos: )

$$\epsilon_1(x) = \|b - A \cdot x\| \tag{10}$$

$$\epsilon_2(x) = \|x - x_0\| \tag{11}$$

În formula de mai sus, $x_0$ reprezintă soluția sistemului returnată de operatorul \. Prima variantă de eroare este mai bună, deoarece reprezintă o metrică obiectivă ce indică corectitudinea soluției (spre deosebire de varianta 2, unde rezultatul nu este comparat cu cel real, ci cu cel aproximat de operator).

# 2 Prezentarea soluțiilor

## 2.1 Prezentarea modului în care funcționează algoritmii aleși

### Eliminarea gaussiană

Eliminarea gaussiană este cel mai simplu și intuitiv algoritm de determinare a soluției unui sistem de ecuații liniare. În cadrul acestui algoritm, se pot realiza următoarele operații pe ecuațiile unui sistem (respectiv, pe liniile matricei sistemului $A$), astfel:

- Adunarea la o ecuație (linie) a oricărei alte ecuații (linii).
- Înmulțirea unei ecuații (linii) cu orice număr real nenul.
- Adunarea la o ecuație (linie) a oricărei alte ecuații (linii), înmulțită cu un număr real (operație derivată din primele 2 operații).
- Permutarea ecuațiilor (liniilor) în orice ordine (caz particular: interschimbarea a două ecuații (linii)).

Principalele etape ale algoritmului de eliminare gaussiană simplă sunt:

- Cât timp mai există linii/coloane de prelucrat, ne alegem un **pivot** (situat pe diagonala principală).
- Scădem din toate liniile care urmează după linia pivotului linia pe care se află pivotul, scalată (astfel încât coeficientul corespunzător necunoscutei pivotului în aceste ecuații să fie 0, adică sub pivot să avem numai elemente egale cu 0).
- Trecem la următorul element de pe diagonala principală (pivot).
- La finalul algoritmului, obținem un sistem superior triunghiular echivalent cu cel inițial, pe care îl putem analiza cu scopul de a-i stabili caracterul (are/nu are soluție unică), precum și a-i determina soluția (dacă aceasta există).
- După terminarea algoritmului, problema originală s-a redus la rezolvare aunui sistem superior triunghiular, care poate fi rezolvat foarte simplu prin substituție înapoi.

O problemă semnificativă este dată de posibilitatea apariției a unor erori foarte mari pe anumite cazuri particulare datorită unui pivot cu valoare absolută mică (astfel că la calcularea factorului de scalare putem avea erori foarte mari de aproximare și instabilitate numerică). De aceea, se folosesc strategii de pivotare, dintre care putem menționa eliminarea gaussiană cu **pivotare parțială** și eliminarea gaussiană cu **pivotare parțială cu pivot scalat**.

În cadrul eliminării gaussiene cu pivotare parțială (**GPP**), înainte de a realiza eliminarea propriu-zisă, se determină cel mai mare element de pe coloana pivotului, începând cu acesta (cu alte cuvinte, cel mai mare element de sub pivot). Apoi, se interschimbă linia pivotului cu linia elementului descoperit, astfel aducând pe poziția pivotului cel mai mare element (micșorând eroarea generată de scalarea la un pivot mic).

În cadrul eliminării gaussiene cu pivotare parțială cu pivot scalat (**GPPS**), se dorește găsirea unui nou pivot (în mod asemănător cu GPP). De această dată, se va alege cel mai mare element de sub pivot raportat (scalat) la maximul (în valoare absolută) de pe linia respectivului element. Astfel, îmbunătățim stabilitatea numerică a algoritmului nostru cu costul unui efort computațional suplimentar depus.

### Factorizarea QR folosind Householder

Una dintre cele mai folosite metode pentru soluționarea sistemelor de ecuații liniare este factorizarea QR. Această metodă de rezolvare are marele avantaj de a fi eficientă mai ales asupra sistemelor de ecuații cu matrice rare, unde folosirea unor metode tradiționale precum factorizarea LU nu este de dorit (nu exploatăm calitatea matricei sistemului de a fi matrice rară). Am ales factorizarea QR folosind transformări ortogonale Householder deoarece, spre deosebire de celelalte variante existente (aplicarea algoritmului Gram-Schmidt și factorizarea folosind matrice de rotație - Givens) timpul de execuție este mai redus și stabiitatea numerică este mai mare, deși complexitatea temporală este aceeași ($O(n^3)$ pentru matrice pătratice).

Ideea algoritmului este de a construi, pentru fiecare vector coloană din matricea ce se dorește a fi factorizată, un reflector (matrice ortogonală, simetrică, notată cu $H$, cu proprietatea că $H^2 = I_n$) denumit și reflector elementar Householder. Acest reflector, aplicat unui vector coloană $z$ are efectul de a reflecta acest vector pe un subspațiu vectorial (mai exact, de a păstra primele k componente din cele n ale vectorului nostru nenule, restul fiind nule). Pentru un vector coloană $z$, se parcurg următorii pași pentru a factoriza matricea noastră:

- Pe coloana numărul $k$ din matrice, se alege elementul de pe poziția $k$ (de pe diagonala principală) drept pivot.
- Se extrage $z$, vectorul coloană din matrice ce are pivotul drept prim element.
- Se completează vectorul $z$ cu zerouri (deasupra pivotului) pentru a obține un vector de dimensiune $n$, care va reprezenta vectorul director al reflectorului elementar (hiperplan față de care se reflectă toți vectorii din matricea A).
- Se calculează reflectorul elementar Householder conform următoarei formule:

$$H = I_n - 2\frac{vv^T}{v^Tv} \tag{12}$$

- Se actualizează matricea ortogonală $Q$ și matricea superior-triunghiulară $R$:

$$Q \leftarrow H \cdot Q \tag{13}$$

$$A \leftarrow H \cdot A \tag{14}$$

- Se continuă cu următoarea coloană.

Alogritmul continuă cât timp există coloane și linii disponibile pentru a efectua factorizarea. Odată realizată factorizarea QR, la fel ca la eliminarea gaussiană, rezolvarea presupune rezolvarea sistemului superior triunghiular astfel obținut prin substituție înapoi, astfel:

$$A \cdot x = b \tag{15}$$

$$Q \cdot R \cdot x = b \tag{16}$$

Matricea $Q$ este ortogonală, deci:

$$Q \cdot Q^T = Q^T \cdot Q = I_n \tag{17}$$

Astfel, inversa se calculează mult mai ușor, sistemul de rezolvat fiind dat de:

$$R \cdot x = Q^T \cdot b \tag{18}$$

Prin analiza corespunzătoare a matricei $R$, se poate astfel da un răspuns exact cu privire la compatibilitatea și determinarea sistemului.

**Metoda gradientului conjugat**

Spre deosebire de metodele exacte prezentate până acum, metoda gradientului conjugat iese în evidență prin câteva dezavantaje pe care le are:

- Pentru a putea folosi metoda gradientului conjugat, matricea sistemului trebuie să fie simetrică și pozitiv-definită (o condiție destul de restrictivă în general, dar care nu reprezintă o problemă în domenii unde astfel de sisteme sunt des întâlnite, cum ar fi Machine Learning).
- Spre deosebire de eliminarea gaussiană și Householder, metoda gradientului conjugat nu oferă un mecanism de determinare a compatibilității/determinării unui sistem (se aplică exclusiv pe sisteme bine definite, compatibil determinate).

Cu toate acestea, metoda este extrem de eficientă pentru sisteme pozitiv-definite (complexitate $O(n^2)$, unde n este dimensiunea matricei sistemului), putând fi optimizată pentru matrice rare (se poate demonstra că în acest caz complexitatea este de $O(m\sqrt{k})$, unde $m$ este numărul de elemente nenule, iar $k$ este numărul de condiționare). De asemenea, metoda poate fi extinsă în anumite situații pentru matrice nesimetrice (gradienți biconjugați).

Metoda noastră se bazează pe ideea de **vectori conjugați** în raport cu matricea $A$. Doi vectori $x$ și $y$ se numesc conjugați dacă $x^T Ay = 0$. Principala idee a algoritmului este de a genera direcții conjugate, care pentru o matrice $A$ inversabilă sunt liniar independente. În acest mod, soluția noastră este căutată într-un subspațiu de tip Krylov, adică:

$$x_{sol} \in span(b, A \cdot b, A^2 \cdot b \cdots A^{n-1} \cdot b) \tag{19}$$

Deoarece direcțiile de căutare conjugate acoperă tot spațiul $\mathbb{R}^n$, algoritmul nostru va determina soluția exactă în $n$ pași. Cu toate acestea, algoritmul poate fi oprit mai devreme dacă aproximația soluției sistemului este suficient de bună.

### 2.2 Analiza complexității soluțiilor

În general, pentru metodele exacte complexitatea rezolvării unui sistem liniar de ecuații pătratic (acestea sunt cele mai întâlnite sisteme de ecuații - sistemele de ecuații subminesionate sau supradimensionate au dezavantajul de a fi incompatibile sau compatibil nedeterminate, caz care este lipsit de relevanță practică în cele mai multe situații) face parte din clasa de complexitate $\Theta(n^3)$.

Pentru metoda gradientului conjugat, complexitatea algoritmului este $\Theta(n^2)$ pentru fiecare pas. Deși algoritmul nostru poate executa maximum $n$ pași (acest lucru se întâmplă în cazul cel mai nefavorabil, atunci când sunt parcurse toate direcțiile conjugate din subspațiul Krylov), în practică eroarea se micșorează considerabil după numai câțiva pași (în general, s-a observat că eroarea devine de ordin sub $10^{-10}$ după 4-5 pași). Astfel, deși pentru rigurozitate putem spune că complexitatea întregului algoritm este de $O(n^3)$, din punct de vedere practic complexitatea este $\Theta(n^2)$.

Complexitatea rezolvării unui sistem superior triunghiular este $\Theta(n^2)$. Această afirmație se poate justifica astfel: la pasul $k$, se efectuează $n-k$ operații, deci complexitatea este dată de:

$$S = \sum_{k=1}^{n}(n-k) = n^2 - \frac{n(n+1)}{2} = \frac{n(n-1)}{2} \to \Theta(n^2) \tag{20}$$

**Eliminarea gaussiană**

Pentru eliminarea gaussiană clasică (fără pivotare și fără scalare), studiem numărul de pași efectuați la pasul $k$ (unde $k$ va lua valori între 1 și $n$). La pasul $k$, se calculează $n-k$ factori de scalare (factorii cu care se înmulțește prima linie pentru a obține, prin scăderi repetate, valori nule sub pivot) și $2(n-k)(n-k+1)$ operații aritmetice (jumătate fiind înmulțiri cu factorii de scalare, iar cealaltă jumătate fiind scăderi): linia $k$, care are

$$x_k = 2(n-k)(n-k+1) \tag{21}$$

Sumând pentru toate valorile posibile ale lui $k$ (de la 1 la $n$, obținem:

$$S = \sum_{k=1}^{n} x_k + \sum_{k=1}^{n}(n-k) = \frac{n(n-1)}{2} + n(n-1) + 2\sum_{k=1}^{n} k^2 \tag{22}$$

$$S = \frac{3n(n-1)}{2} + \frac{n(n+1)(2n+1)}{6} \to \Theta(n^3) \tag{23}$$

Menționăm că este important faptul că, pentru n mare, $T(n) \approx \frac{2n^3}{3}$). Mai este necesară și rezolvarea sistemului superior triunghiular obținut, însă aceasta nu afectează constanta de complexitate (rezolvarea unui SST are complexitate $\Theta(n^2)$). Observăm că constanta corespunzătoare clasei de complexitate obținute

este 2/3, fapt foarte important pentru a compara eliminrea gaussiană generală cu celelalte metode.

În practică însă, eliminarea gaussiană clasică nu este folosită, datorită instabilității numerice (putem avea pivot 0 sau foarte apropiat de 0). De aceea, cei doi algoritmi studiați (GPP și GPPS) vor face parte din aceeași clasă de complexitate, însă vor avea alte constante asociate, deoarece presupun un număr în plus de operații aritmetice.

Pentru eliminarea gaussiană cu pivotare parțială, avem în plus operațiile de determinare a maximului la pasul $k$ și de înlocuire a liniilor, operații care presupun un cost suplimentar ($n-k+1$ pentru determinarea maximului, respectiv $3(n-k+1)$ pentru interschimbarea celor 2 linii):

$$\delta x_{k,GPP} = n-k+1+3(n-k+1) = 4(n-k+1) \tag{24}$$

Costul total, cumulat pe toți cei $n$ pași este:

$$\Delta x_{GPP} = \sum_{k=1}^{n} \delta x_{k,GPP} = 4\sum_{k=1}^{n}(n-k+1) = 4\sum_{k=1}^{n} k = 2n(n+1) \to \Theta(n^2) \tag{25}$$

Din considerente de complexitate de ordin mai mic al operațiilor adiționale, algoritmul de eliminare gaussiană cu pivotare parțială (GPP) va face parte din aceeași clasă de complexitate și va avea aceeași constantă de complexitate (anume, 2/3).

Acum, studiem operațiile adiționale care au loc pentru algoritmul de eliminare gaussiană cu pivotare parțială cu pivot scalat (GPPS). Costul adițional este dat de următoarea formulă (termenul $(n-k+1)^2$ provine de la calculul maximului pentru fiecare linie, $n-k+1$ pentru scalare și încă $n-k+1$ pentru maximul vectorului în care calculăm scalele):

$$\delta x_{k,GPPS} = (n-k+1)^2 + 2(n-k+1) \tag{26}$$

Calculăm costul total:

$$\Delta x_{GPPS} = \sum_{k=1}^{n} \delta x_{k,GPPS} = \sum_{k=1}^{n}(n-k+1)^2 + 2(n-k+1) \tag{27}$$

$$\Delta x_{GPPS} = \sum_{k=1}^{n} k^2 + k = \frac{n(n+1)}{2} + \frac{n(n+1)(2n+1)}{6} \to \Theta(n^3) \tag{28}$$

Obținem că algoritmul de eliminare gaussiană cu pivotare parțială cu pivot scalat (GPPS) este tot în clasa de complexitate $\Theta(n^3)$, însă are altă constantă de complexitate ($\frac{2}{3}+\frac{1}{3}=1$). Concluziile în ceea ce privește complexitatea sunt:

$$T_G(n) \approx \frac{2n^3}{3} \in \Theta(n^3) \tag{29}$$

$$T_{GPP}(n) > T_G(n), T_{GPP}(n) \approx \frac{2n^3}{3} \in \Theta(n^3) \tag{30}$$

$$T_{GPPS}(n) > T_{GPP}(n), T_{GPPS}(n) \approx n^3 \in \Theta(n^3) \tag{31}$$

**Householder**

Pentru rezolvarea unui sistem de ecuații folosind factorizarea QR cu transformări ortogonale Householder, sunt necesari $n-1$ pași. La fel ca în cazul eliminării gaussiene, vom demonstra că algoritmul Householder face parte din clasa de complexitate $\Theta(n^3)$.

Cel puțin aparent, algoritmul Householder ar trebui să aibă complexitate $\Theta(n^3)$ pentru fiecare pas, întrucât la fiecare pas avem o înmulțire de matrice pătratice, de ordin $n$ (se știe că înmulțirea a două matrice oarecare se efectuează în complexitate $\Theta(n^3)$). Cu toate acestea, putem realiza următoarea optimizare: scriem reflectorul explicit și, în loc să calculăm reflectorul direct, vom calcula direct noua matrice astfel:

$$A \leftarrow HA = (I_n - \frac{2vv^T}{v^Tv})A = A - \frac{2vv^T}{v^Tv}A \tag{32}$$

Dacă realizăm operațiile de forma matrice - vector coloană sau vector linie - matrice, obținem că această operație poate fi realizată în complexitate $\Theta(n^2)$. Mai precis, numărul de operații efectuate este aproximativ $2n^2$ în timpul unui pas de Householder.

Costul pentru un pas de Householder este:

$$\delta x_{k,HH} \approx 2(n-k+1)^2 \tag{33}$$

Costul pentru rularea întregului algoritm este:

$$\Delta x_{HH} = \sum_{k=1}^{n-1} \delta x_{k,HH} \approx \sum_{k=1}^{n} \delta x_{k,HH} \tag{34}$$

$$\Delta x_{HH} \approx \sum_{k=1}^{n} 2(n-k+1)^2 = \sum_{k=1}^{n} 2k^2 = \frac{2n(n+1)(2n+1)}{6} \approx \frac{2n^3}{3} \in \Theta(n^3) \tag{35}$$

Obținem că algoritmul Householder este similar cu eliminarea gaussiană (clasică sau cu pivotare parțială) din punctul de vedere al timpului de rulare (aceeași clasă de complexitate și aceeași constantă de complexitate).

**Metoda gradientului conjugat**

Pentru metoda gradientului conjugat, singura operație de complexitate mare este cea de determinare a noii direcții de căutare (înmulțire de matrice cu vector), care are complexitate $\Theta(n^2)$. Conform precizărilor anterioare, deducem că, din punct de vedere practic putem afirma cu certitudine că complexitatea metodei gradientului conjugat este $\Theta(n^2)$.

### 2.3 Avantajele și dezavantajele metodelor propuse

Toate metodele propuse sunt folosite în cazuri bine determinate și în anumite contexte pe care le vom preciza în ceea ce urmează:

#### Eliminarea gaussiană

Eliminarea gaussiană (în cele două variante ale sale, GPP și GPPS) are următoarele avantaje:

- Este un algoritm ușor de înțeles și intuitiv.
- Spre deosebire de factorizările QR, eliminarea gaussiană are marele avantaj de a nu folosi memorie suplimentară. Acest fapt o face foarte potrivită pentru rezolvarea sistemelor de ecuații de către sistemele de calcul ce dispun de puține resurse (mai ales în IoT există situații în care se dorește determinarea rapidă a soluției, însă cu efort computațional redus, chiar dacă metoda folosită nu este optimă).

De asemenea, aceasta are și unele dezavantaje, printre care se numără:

- Numărul mare de operații o face nepotrivită pentru sistemele de ecuații cu numere reale de dimensiuni foarte mari, în care numărul de condiționare este mare.
- În particular, eliminarea gaussiană cu pivotare parțială cu pivot scalat este folosită mai rar, întrucât numărul său suplimentar de operații în raport cu celelalte metode exacte nu o face un candidat fezabil pentru rezolvarea sistemelor de ecuații. Aceasta poate da rezultate mai bune decât GPP, însă mai slabe decât Householder, nejustificându-se astfel efortul computațional.

#### Householder

Printre avantajele algoritmului Householder se numără:

- Rezultate bune și foarte bune pentru sistemele de ecuații liniare, în general.
- Factorizarea Householder este folosită în mulți alți algoritmi computaționali, astfel justificându-se eficiența sa (algoritmul QR, tehnici de deflație etc).
- Algoritm foarte rapid, care prezintă o stabilitate numerică superioară altor metode exacte.
- Eliminarea tuturor elementelor de sub pivot are loc într-un singur pas, folosind reflectorul elementar Householder.

Unele dintre dezavantajele acestui algoritm sunt:

- Folosește memorie suplimentară (pentru matricea ortogonală finală, pentru reflectori și pentru vectorii cu elementele de sub pivot - vectorii de reflectat), ceea ce nu îl face fezabil pentru sistemele cu puține resurse sau sistemele de numere întregi - cazul tehnologiilor IoT (în acest caz, eliminarea gaussiană are un avantaj clar).

- Pentru sisteme foarte rău condiționate (cum ar fi sistemele subdimensionate - întâlnite în Machine Learning), Householder poate eșua în a oferi un rezultat bun. În acest caz, sunt folosite alte metode (de obicei iterative), cum ar fi Gradient Descent (în general, se preferă o rezolvare de eroare minimă în sensul celor mai mici pătrate, nu o rezolvare a sistemului în sensul folosit până acum).

**Metoda gradientului conjugat**

Printre avantajele metodei gradientului conjugat se numără:

- Rezultate bune și foarte bune pentru sistemele de ecuații liniare în care matricea sistemului este pozitiv definită.
- Pentru matrice rare, această metodă are un plus de avantaj deoarece poate fi optimizată să calculeze acele direcții de căutare și să ajungă la soluția exactă în complexitate $\Theta(m\sqrt{k})$, unde $m$ reprezintă numărul de elemente nenule din matricea rară, iar $k$ este numărul de condiționare.
- Prin ideea sa, poate fi folosită pentru rezolvarea unor sisteme de ecuații mult prea mari (cu dimensiunea peste $10^3$) pentru ca o metodă exactă să le poată rezolva în timp fezabil (datorită clasei diferite de complexitate).

Principalul dezavantaj al metodei gradientului conjugat este că nu poate oferi o rezolvare pentru sistemele de ecuații liniare generale. O încercare de acest gen poate duce la o rezolvare greșită (funcția de potențial asociată matricei $A$, $\Phi(x) = \frac{1}{2}x^T Ax - b^T x$ nu are un minim global).

## 3 Evaluarea soluțiilor propuse

### 3.1 Descrierea modalității de construire a setului de teste folosit pentru validare

Pentru implementarea soluțiilor propuse, am folosit environmentul GNU Octave (versiunea 5.2.0). Am realizat un script în cadrul acestui environment care îmi generează matrice cu valori random (matricea $A$) și respectiv un vector de aceeași dimensiune cu matricea noastră (vectorul $b$), iar rezultatele le-am salvat în cadrul folderului in/. Utilizând alt script scris tot cu ajutorul Octave,

am preluat datele de intrare (matricea $A$ și vectorul $b$) și am rezolvat sistemele de ecuații folosind operatorul \ existent în sintaxa de Matlab/Octave. Pentru ultimele 20 de teste, matricea $A$ a fost generată de forma $L \cdot L^T$, pentru a ne asigura că aceasta este pozitiv definită și simetrică, așa cum ar trebui să fie matricea sistemului pentru metoda gradientului conjugat.

### 3.2 Specificațiile sistemului de calcul

Pentru rularea algoritmilor am folosit laptop-ul personal, care are următoarele caracteristici:

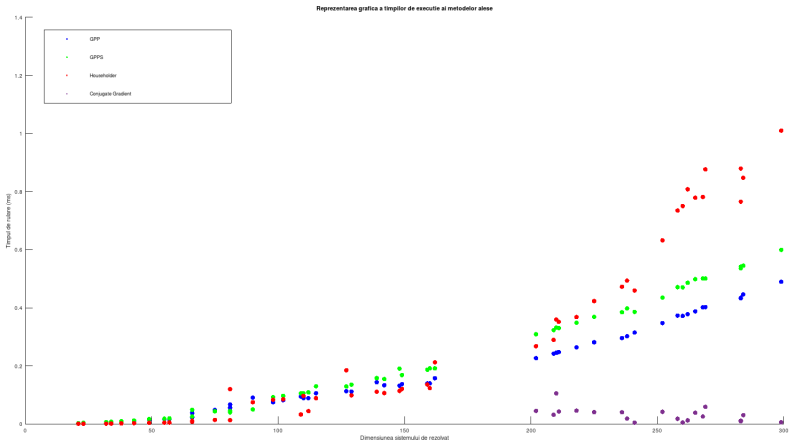
| Procesor | Intel Core i7-1165G7 |
|---|---|
| Frecvență (MHz) | 2800 |
| Număr core-uri | 8 |
| Număr thread-uri / core | 2 |
| Memorie RAM | 16 GB |

### 3.3 Ilustrarea rezultatelor evaluării soluțiilor

Am creat un nou script (diferit de checker-ul pus la dispoziție în cadrul etapei 2) care îmi rulează toate testele, înregistrând simultan și cele două tipuri de erori prezentate anterior (am folosit pentru calculul erorii ca scalar norma euclidiană).

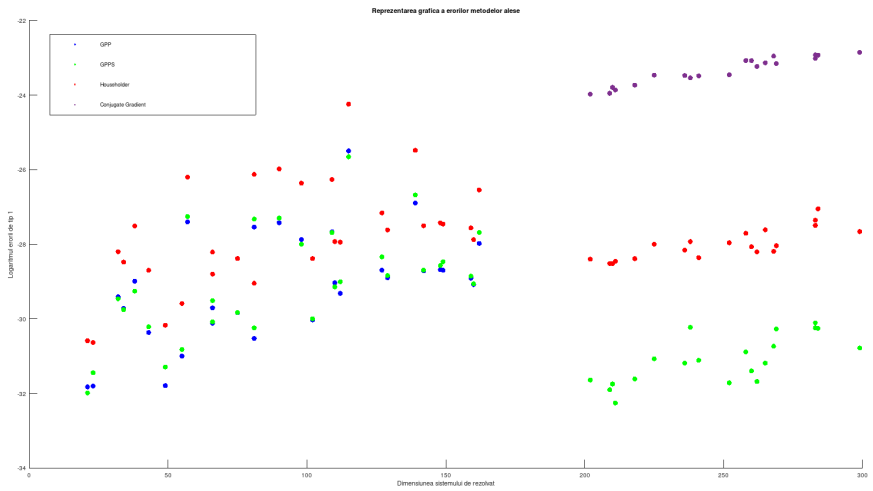
Am măsurat timpul de rulare a algoritmului (excluzând partea de I/O).

Rezultatele soluțiilor propuse pentru primele 30 de teste (în ceea ce privește timpul de rulare) au fost înregistrate în următorul tabel:




Reprezentarea grafică a timpului de execuție în funcție de dimensiunea sistemului

| NR CRT | Dimensiune | GPP(ms) | GPPS(ms) | Householder(ms) |
|---|---|---|---|---|
| 1 | 23 | 3.221989 | 3.865957 | 1.168966 |
| 2 | 21 | 2.990961 | 3.224134 | 0.954151 |
| 3 | 32 | 6.342888 | 6.803989 | 1.478910 |
| 4 | 34 | 6.595135 | 7.583857 | 1.625061 |
| 5 | 38 | 8.798122 | 10.190010 | 2.038956 |
| 6 | 43 | 9.913206 | 12.367964 | 3.197908 |
| 7 | 49 | 14.855862 | 16.833067 | 3.969908 |
| 8 | 55 | 17.102957 | 18.251896 | 4.660130 |
| 9 | 57 | 16.656160 | 19.535065 | 5.184889 |
| 10 | 66 | 22.315979 | 25.816917 | 7.429838 |
| 11 | 66 | 37.883043 | 48.960209 | 11.134148 |
| 12 | 75 | 48.141003 | 43.897867 | 14.214993 |
| 13 | 81 | 67.413092 | 39.968967 | 13.303995 |
| 14 | 81 | 55.982113 | 44.672012 | 120.224953 |
| 15 | 90 | 90.826035 | 49.957991 | 74.954033 |
| 16 | 98 | 74.987888 | 92.416048 | 83.616972 |
| 17 | 102 | 82.275867 | 96.977949 | 84.949017 |
| 18 | 109 | 94.759941 | 105.638981 | 32.914162 |
| 19 | 112 | 89.035988 | 109.035015 | 44.350147 |
| 20 | 110 | 89.154959 | 106.414080 | 96.901894 |
| 21 | 115 | 105.967999 | 130.110025 | 89.031935 |
| 22 | 129 | 111.716032 | 135.333061 | 98.667145 |
| 23 | 127 | 112.848043 | 129.257917 | 185.083151 |
| 24 | 139 | 143.871069 | 158.180952 | 111.303091 |
| 25 | 142 | 133.553982 | 154.754877 | 106.466055 |
| 26 | 149 | 137.689829 | 168.459177 | 120.877028 |
| 27 | 148 | 131.919146 | 190.723896 | 113.977909 |
| 28 | 160 | 139.952898 | 191.621780 | 123.728991 |
| 29 | 159 | 139.668941 | 186.810017 | 136.007071 |
| 30 | 162 | 157.749176 | 191.505909 | 212.241888 |



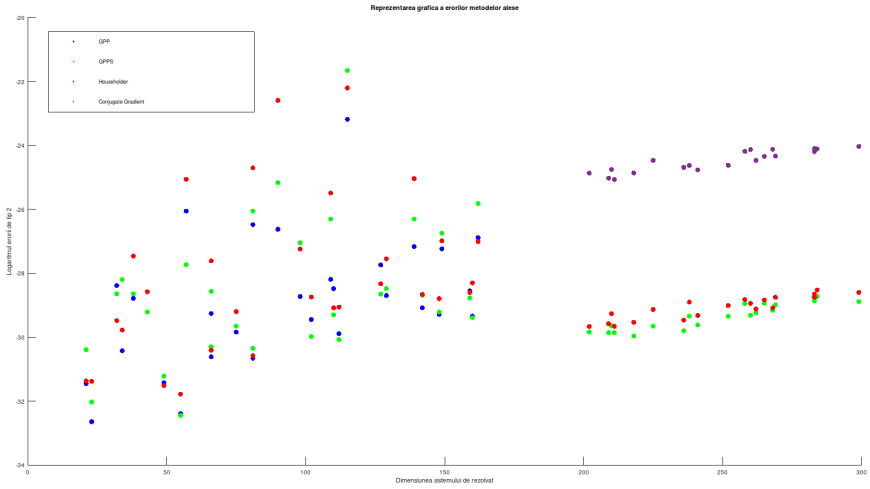

Reprezentarea grafică a erorii de tip 1 în funcție de dimensiunea sistemului

Pentru următoarele 20 de teste (testele 31-50), rezultatele înregistrate au fost următoarele:

| NR CRT | Dimensiune | GPP(ms) | GPPS(ms) | Householder(ms) | Conjugate Gradient(ms) |
|---|---|---|---|---|---|
| 31 | 283 | 434.362888 | 541.569948 | 765.260935 | 9.194136 |
| 32 | 284 | 446.337938 | 545.037985 | 847.602129 | 30.786037 |
| 33 | 241 | 314.723015 | 386.065960 | 459.676981 | 4.880905 |
| 34 | 238 | 302.231789 | 397.830009 | 493.402004 | 18.620014 |
| 35 | 236 | 295.907974 | 384.886026 | 472.659826 | 40.587187 |
| 36 | 283 | 432.999849 | 535.861969 | 879.420042 | 11.452198 |
| 37 | 299 | 489.674807 | 599.709034 | 1010.138035 | 6.170034 |
| 38 | 252 | 347.576141 | 435.213089 | 632.377148 | 42.170048 |
| 39 | 260 | 371.865988 | 470.326900 | 750.220060 | 5.213976 |
| 40 | 262 | 378.350973 | 486.150980 | 808.108091 | 12.591124 |
| 41 | 209 | 242.424011 | 323.169947 | 289.690018 | 31.876802 |
| 42 | 218 | 264.127970 | 348.519087 | 368.232965 | 46.047926 |
| 43 | 258 | 373.377085 | 471.135139 | 734.872103 | 18.159151 |
| 44 | 202 | 226.552010 | 309.182882 | 267.970800 | 45.437098 |
| 45 | 211 | 247.859955 | 330.493927 | 351.737022 | 42.909145 |
| 46 | 268 | 401.748896 | 501.165152 | 781.590939 | 25.716066 |
| 47 | 225 | 281.519175 | 369.092941 | 423.119068 | 40.709019 |
| 48 | 265 | 388.000011 | 498.669863 | 778.848886 | 38.897991 |
| 49 | 269 | 402.606010 | 500.981092 | 876.548052 | 59.240818 |
| 50 | 210 | 245.099783 | 332.324982 | 359.490871 | 105.255842 |

| NR CRT | Dimensiune | GPP | GPPS | Householder | Conjugate Gradient |
|---|---|---|---|---|---|
| 1 | 23 | 1.548e-14 | 2.2099e-14 | 4.9708e-14 | - |
| 2 | 21 | 1.5106e-14 | 1.2867e-14 | 5.2106e-14 | - |
| 3 | 32 | 1.6925e-13 | 1.6078e-13 | 5.6812e-13 | - |
| 4 | 34 | 1.2385e-13 | 1.1981e-13 | 4.2893e-13 | - |
| 5 | 38 | 2.5768e-13 | 1.9731e-13 | 1.1277e-12 | - |
| 6 | 43 | 6.5021e-14 | 7.5675e-14 | 3.4388e-13 | - |
| 7 | 49 | 1.5632e-14 | 2.576e-14 | 7.8962e-14 | - |
| 8 | 55 | 3.4556e-14 | 4.1181e-14 | 1.4163e-13 | - |
| 9 | 57 | 1.2588e-12 | 1.4603e-12 | 4.1924e-12 | - |
| 10 | 66 | 8.3081e-14 | 8.6426e-14 | 3.1117e-13 | - |
| 11 | 66 | 1.258e-13 | 1.5287e-13 | 5.6171e-13 | - |
| 12 | 75 | 1.1051e-13 | 1.109e-13 | 4.7242e-13 | - |
| 13 | 81 | 1.099e-12 | 1.3611e-12 | 4.5105e-12 | - |
| 14 | 81 | 5.5424e-14 | 7.36e-14 | 2.4355e-13 | - |
| 15 | 90 | 1.2336e-12 | 1.3988e-12 | 5.2114e-12 | - |
| 16 | 98 | 7.872e-13 | 6.9502e-13 | 3.5621e-12 | - |
| 17 | 102 | 9.108e-14 | 9.3958e-14 | 4.7191e-13 | - |
| 18 | 109 | 9.6868e-13 | 9.4746e-13 | 3.9366e-12 | - |
| 19 | 112 | 1.8561e-13 | 2.5334e-13 | 7.3335e-13 | - |
| 20 | 110 | 2.4666e-13 | 2.209e-13 | 7.4489e-13 | - |
| 21 | 115 | 8.4628e-12 | 7.2281e-12 | 2.9687e-11 | - |
| 22 | 129 | 2.8211e-13 | 2.99e-13 | 1.017e-12 | - |
| 23 | 127 | 3.457e-13 | 4.9411e-13 | 1.6063e-12 | - |
| 24 | 139 | 2.0983e-12 | 2.5969e-12 | 8.6025e-12 | - |
| 25 | 142 | 3.3914e-13 | 3.4637e-13 | 1.1381e-12 | - |
| 26 | 149 | 3.4541e-13 | 4.3309e-13 | 1.1891e-12 | - |
| 27 | 148 | 3.5001e-13 | 3.9182e-13 | 1.2291e-12 | - |
| 28 | 160 | 2.3512e-13 | 2.4028e-13 | 7.8374e-13 | - |
| 29 | 159 | 2.7905e-13 | 2.9506e-13 | 1.0726e-12 | - |
| 30 | 162 | 7.0849e-13 | 9.5213e-13 | 2.9741e-12 | - |
| 31 | 283 | 8.4158e-14 | 8.4158e-14 | 1.3195e-12 | 1.0111e-10 |
| 32 | 284 | 7.2667e-14 | 7.2667e-14 | 1.7915e-12 | 1.1052e-10 |
| 33 | 241 | 3.0879e-14 | 3.0879e-14 | 4.8332e-13 | 6.3318e-11 |
| 34 | 238 | 7.4786e-14 | 7.4786e-14 | 7.4402e-13 | 5.9968e-11 |
| 35 | 236 | 2.8681e-14 | 2.8681e-14 | 5.935e-13 | 6.3899e-11 |
| 36 | 283 | 7.3454e-14 | 7.3454e-14 | 1.148e-12 | 1.1076e-10 |
| 37 | 299 | 4.2851e-14 | 4.2851e-14 | 9.7422e-13 | 1.1896e-10 |
| 38 | 252 | 1.6878e-14 | 1.6878e-14 | 7.2192e-13 | 6.5183e-11 |
| 39 | 260 | 2.3251e-14 | 2.3251e-14 | 6.4836e-13 | 9.5362e-11 |
| 40 | 262 | 1.7456e-14 | 1.7456e-14 | 5.6395e-13 | 8.1703e-11 |
| 41 | 209 | 1.4027e-14 | 1.4027e-14 | 4.1298e-13 | 3.9702e-11 |
| 42 | 218 | 1.8745e-14 | 1.8745e-14 | 4.7071e-13 | 4.9461e-11 |
| 43 | 258 | 3.862e-14 | 3.862e-14 | 9.2986e-13 | 9.5633e-11 |
| 44 | 202 | 1.8195e-14 | 1.8195e-14 | 4.6461e-13 | 3.8762e-11 |
| 45 | 211 | 9.823e-15 | 9.823e-15 | 4.3916e-13 | 4.3442e-11 |
| 46 | 268 | 4.4907e-14 | 4.4907e-14 | 5.7214e-13 | 1.0744e-10 |
| 47 | 225 | 3.2129e-14 | 3.2129e-14 | 6.9194e-13 | 6.4559e-11 |
| 48 | 265 | 2.86e-14 | 2.86e-14 | 1.0205e-12 | 9.009e-11 |
| 49 | 269 | 7.1409e-14 | 7.1409e-14 | 6.661e-13 | 8.8063e-11 |
| 50 | 210 | 1.6324e-14 | 1.6324e-14 | 4.1129e-13 | 4.6528e-11 |

| NR CRT | Dimensiune | GPP | GPPS | Householder | Conjugate Gradient |
|---|---|---|---|---|---|
| 1 | 23 | 6.6722e-15 | 1.2327e-14 | 2.3626e-14 | - |
| 2 | 21 | 2.1951e-14 | 6.3621e-14 | 2.3825e-14 | - |
| 3 | 32 | 4.7152e-13 | 3.6544e-13 | 1.575e-13 | - |
| 4 | 34 | 6.1452e-14 | 5.6966e-13 | 1.1756e-13 | - |
| 5 | 38 | 3.1578e-13 | 3.6384e-13 | 1.1842e-12 | - |
| 6 | 43 | 3.8968e-13 | 2.0637e-13 | 3.8901e-13 | - |
| 7 | 49 | 2.2555e-14 | 2.7693e-14 | 2.0616e-14 | - |
| 8 | 55 | 8.5726e-15 | 8.1068e-15 | 1.5789e-14 | - |
| 9 | 57 | 4.8598e-12 | 9.0919e-13 | 1.3132e-11 | - |
| 10 | 66 | 5.0697e-14 | 6.9708e-14 | 6.2395e-14 | - |
| 11 | 66 | 1.972e-13 | 3.9235e-13 | 1.0213e-12 | - |
| 12 | 75 | 1.0983e-13 | 1.3322e-13 | 2.0907e-13 | - |
| 13 | 81 | 3.1809e-12 | 4.8458e-12 | 1.8701e-11 | - |
| 14 | 81 | 4.8709e-14 | 6.635e-14 | 5.2534e-14 | - |
| 15 | 90 | 2.7471e-12 | 1.1799e-11 | 1.5504e-10 | - |
| 16 | 98 | 3.3537e-13 | 1.7993e-12 | 1.4765e-12 | - |
| 17 | 102 | 1.6338e-13 | 9.585e-14 | 3.3002e-13 | - |
| 18 | 109 | 5.7289e-13 | 3.7727e-12 | 8.5423e-12 | - |
| 19 | 112 | 1.049e-13 | 8.6876e-14 | 2.4103e-13 | - |
| 20 | 110 | 4.2851e-13 | 1.8901e-13 | 2.3571e-13 | - |
| 21 | 115 | 8.5853e-11 | 3.9441e-10 | 2.2833e-10 | - |
| 22 | 129 | 3.4594e-13 | 4.304e-13 | 1.0911e-12 | - |
| 23 | 127 | 8.9712e-13 | 3.6204e-13 | 4.9857e-13 | - |
| 24 | 139 | 1.6008e-12 | 3.7737e-12 | 1.3394e-11 | - |
| 25 | 142 | 2.356e-13 | 3.4783e-13 | 3.5795e-13 | - |
| 26 | 149 | 1.4953e-12 | 2.421e-12 | 1.9091e-12 | - |
| 27 | 148 | 1.9213e-13 | 2.0621e-13 | 3.134e-13 | - |
| 28 | 160 | 1.8069e-13 | 1.7348e-13 | 5.1287e-13 | - |
| 29 | 159 | 4.0065e-13 | 3.21e-13 | 3.8022e-13 | - |
| 30 | 162 | 2.1171e-12 | 6.141e-12 | 1.8692e-12 | - |
| 31 | 283 | 2.9263e-13 | 2.9263e-13 | 3.2848e-13 | 3.1111e-11 |
| 32 | 284 | 3.3544e-13 | 3.3544e-13 | 4.1209e-13 | 3.3946e-11 |
| 33 | 241 | 1.3731e-13 | 1.3731e-13 | 1.8517e-13 | 1.7633e-11 |
| 34 | 238 | 1.8162e-13 | 1.8162e-13 | 2.8093e-13 | 2.0244e-11 |
| 35 | 236 | 1.1456e-13 | 1.1456e-13 | 1.597e-13 | 1.9021e-11 |
| 36 | 283 | 3.358e-13 | 3.358e-13 | 3.6215e-13 | 3.4302e-11 |
| 37 | 299 | 2.853e-13 | 2.853e-13 | 3.8105e-13 | 3.6727e-11 |
| 38 | 252 | 1.8063e-13 | 1.8063e-13 | 2.5241e-13 | 2.0333e-11 |
| 39 | 260 | 1.8653e-13 | 1.8653e-13 | 2.7032e-13 | 3.3271e-11 |
| 40 | 262 | 2.0216e-13 | 2.0216e-13 | 2.2649e-13 | 2.3716e-11 |
| 41 | 209 | 1.0824e-13 | 1.0824e-13 | 1.4287e-13 | 1.3576e-11 |
| 42 | 218 | 9.7094e-14 | 9.7094e-14 | 1.4961e-13 | 1.601e-11 |
| 43 | 258 | 2.681e-13 | 2.681e-13 | 3.0399e-13 | 3.1394e-11 |
| 44 | 202 | 1.0999e-13 | 1.0999e-13 | 1.3124e-13 | 1.5868e-11 |
| 45 | 211 | 1.0798e-13 | 1.0798e-13 | 1.317e-13 | 1.3038e-11 |
| 46 | 268 | 2.1874e-13 | 2.1874e-13 | 2.3428e-13 | 3.3568e-11 |
| 47 | 225 | 1.3229e-13 | 1.3229e-13 | 2.2389e-13 | 2.371e-11 |
| 48 | 265 | 2.7139e-13 | 2.7139e-13 | 3.0064e-13 | 2.6833e-11 |
| 49 | 269 | 2.5928e-13 | 2.5928e-13 | 3.2718e-13 | 2.7042e-11 |
| 50 | 210 | 1.3428e-13 | 1.3428e-13 | 1.9533e-13 | 1.7808e-11 |




Reprezentarea grafică a erorii de tip 2 în funcție de dimensiunea sistemului

### 3.4 Prezentarea valorilor obținute pe teste și interpretarea graficelor

Având în vedere graficele și tabelele anexate mai sus, putem observa următoarele:

- Pentru primele 30 de teste, am atașat numai timpii de rulare ai primilor 3 algoritmi (ai metodelor exacte), întrucât aceștia sunt cei mai relevanți (metoda gradientului conjugat nu oferă soluția corectă pentru sisteme generale, așa că nu am atașat și timpii corespunzători acestei metode).

- Pentru testele 31-50, am prezentat timpii de rulare ai tuturor algoritmilor, pentru a scoate în evidență avantajul de timp al metodei gradientului conjugat.

- Se observă că timpii de rulare pentru GPP, GPPS și Householder sunt comparabili (sunt teste unde algoritmul Householder este mai eficient, însă sunt și teste unde eliminarea gaussiană este mai eficientă), astfel că cei doi algoritmi principali pot fi comparați doar din punctul de vedere al preciziei oferite (complexitatea temporală fiind aceeași).

- Se observă că timpul de rulare pentru sisteme pozitiv definite sunt mai mici în cazul metodei gradientului conjugat (de aici se justifică și complexitatea mai redusă în comparație cu metodele exacte).

- Eroarea algoritmului Householder este comparabilă cu eroarea dată de eliminarea gaussiană. Cu toate acestea, pentru matrice mai mari eliminarea gaus-

siană își dovedește instabilitatea numerică (sisteme de ecuații cel puțin de ordinul miilor).
– Eroarea gradientului conjugat este mai mare deoarece algoritmul se oprește după ce eroarea scade sub o limită aleasă de noi. În schimb, pentru majoritatea aplicațiilor practice o eroare sub $10^{-7}$ este suficientă.
– Se observă că primul tip de eroare este mai relevant (eroarea în raport cu restul soluției: $r = b - Ax$), întrucât evidențiază mia bine diferențele dintre metodele exacte studiate.

Deși sistemele alese au dimensiuni rezonabile pentru a putea fi rulate pe un calculator personal, se observă faptul că, totuși, erorile date de cele 2 metode exacte sunt de același ordin de mărime (uneori, eroarea dată de algoritmul Householder chiar o depășește pe cea a eliminării gaussiene), fapt care nu evidențiază întru totul instabilitatea numerică a eliminării gaussiene (această instabilitate se poate evidenția cu teste mai mari, dar care sunt imposibil de rulat folosind un calculator personal).

## 4 Concluzii

În concluzie, fiecare dintre metodele studiate are avantajele sale, aspecte evidențiate în cadrul testelor. Pentru sisteme de ecuații de dimensiuni reduse, eliminarea gaussiană este o soluție foarte bună și de dorit (mai ales în cazul în care sistemul de calcul care rezolvă acea problemă dispune de puține resurse). Pentru sisteme de ecuații mari, în care sistemul de calcul dispune de resursele necesare, algoritmul Householder reprezintă o soluție mai bună. În cazul în care avem sisteme de ecuații în care matricea sistemului este pozitiv-definită, metoda gradientului conjugat reprezintă cea mai bună metodă disponibilă. De asemenea, am studiat comparativ 2 tipuri de eroare, din care am demonstrat faptul că eroarea relativă la restul dat de soluția sistemului este cea mai relevantă.

Analiza complexității (realizată în întregime de mine) a arătat că metodele exacte sunt în aceeași clasă de complexitate, însă se disting prin abordare și prin performanță în cazuri particulare. În paralel, metoda gradientului conjugat se distinge prin abordarea teoretică de celelalte metode din clasa metodelor iterative (de tipul Jacobi, Gauss-Seidel, SOR) prin faptul că se adaptează mereu la o nouă direcție de căutare, care nu se repetă (ajunge la soluția exactă în $n$ pași, însă se poate opri și după doar câțiva pași dacă eroarea noastră este suficient de bună, tipic metodelor iterative).

## 5 Referințe


- Multivariate linear regression in Machine Learning
- A simple approach to Conjugate Gradient
- Newton method for solving nonlinear equation systems
- Typical Gaussian elimination
- An interesting approach to Householder transformation
- Householder algorithm complexity
- Gaussian elimination complexity
- Conjugate gradient complexity
- Conjugate gradient analysis
- Gaussian elimination with scale pivoting