שיטות לפתרון מערכת משוואות ליניאריות

<u>מטרה</u>: מציאת הפתרון למערכת $A\vec{x}=\vec{b}$, כאשר

$$A\vec{x}=\begin{pmatrix} a_{1,1} & a_{1,2} \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} \cdots & a_{2,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} \cdots & a_{m,n} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}=\vec{b}=\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

A – מטריצת המקדמים , מבטא את התנאים המקדימים (בביצוע ניסוי...)
b - וקטור התוצאה , מבטא את תוצאות הניסוי.
x - וקטור הנעלם – אותו אנחנו מחפשים!

פתרון של מטריצה נעשה ע"י הפעלה של אוסף של פעולות אלמנטריות על המטריצה:

1. החלפת שתי שורות
2. הכפלת שורה בסקלר שאינו אפס
3. הוספת שורה או כפולה ליניארית של שורה לשורה אחרת

שיטת האלימינציה של גאוס

מדרגים את המטריציה ע"י ביצוע פעולות אלמטריות על המטריצה עד לקבלת מטריצה משולשת עילית שממנה הפתרון ניתן לחילוץ באופן מיידי. כאשר בשיטת האלימינציה של גאוס קודם נאפס את האיברים בעמודה הראשונה ולאחר מכן נעבור לעמודה השנייה ולאחר מכן נמשיך הלאה.

שיטת האלימינציה של גאוס מורכבת משתי חלקים עיקריים:

- **Forward elimination** - מבצעים פעולות אלמנטריות עד שמגיעים למטריצה משולשת עילית. באמצעות שלב זה ניתן לדעת אם אין פתרונות, או פתרון ייחודי, או אינסוף פתרונות.
- **Back substitution**- חילוץ ערכי הנעלמים מהחלק התחתון של המטריצה כלפי מעלה.

pivot-בכל שורה במטריצה שאיננה שורת אפסים, הרכיב הראשון השונה מאפס ייקרא האיבר המוביל או איבר הציר **(pivot)** של השורה.

partial pivoting- מחליפים בין השורות המטריצה כך שהאיבר הגדול ביותר בעמודה (בערכו המוחלט) נמצא על האלכסון הראשי - כאיבר הציר.
תת-שלב זה מבוצע בחלק משלב ה- Forward elimination כדי להקנות יציבות חישובית לאלגוריתם.

1. נסרוק את איברי "עמודת הציר" מתחת לאיבר הציר כולל איבר הציר ונחפש מהו האיבר הגדול ביותר בערכו המוחלט.
2. נחליף בין השורה שבה נמצא האיבר הגדול ביותר לבין שורת הציר, וכך נקטין את ההשפעה של.. העיגול.. הנק' הצפה..

אי-יציבות "מושרית"
אי-יציבות המתקבלת כתוצאה משימוש בעקרון הנקודה הצפה ובשגיאות עיגול.
הסיבה לכך היא הצטברות שגיאות עיגול במהלך החישובים

פסאודו קוד

---

gaussianElimination algorithm

1: **gaussianElimination (A):**
2: **for** $i = 1$ **to** n **do**
3: **partial pivoting:** find p where $|A[p][i]|$ is the largest number with $i \leq p \leq n$
4: **if** $p \neq i$ **then** exchange row i with row p
5: **for** $j = i + 1$ **to** n **do steps 6,7**
6: **set** m **=** $A[j][i]$ / $A[i][i]$
7: Perform $A[j]$ **=** $A[j]$ **-** $m \cdot A[i]$
8: **If** $A[n][n] = 0$ **then** OUTPUT ('no unique solution exists'); STOP
9: **else then** start Back substitution**:**
10: **set** $x[n]$= $A[n][n+1]$ / $A[n][n]$
11: **for** $i = n - 1$ **to** 1 **do**
12: $x[i] = \left[A[i][n+1] - \sum_{j=i+1} A[i][j] \cdot x[j]\right] / A[i][i]$
13: OUTPUT x

שיטות איטרטיביות לפתרון משוואות ליניאריות

בשיטה איטרטיבית נבצע שימוש חוזר בנוסחה מתמטית לצורך קבלת תיקון לפתרון מקורב.
פתרון מערכת משוואות$AX = b$ בשיטה איטראטיבית מתחיל עם קירוב התחלתי של הפתרון X ומייצרים סדרת קירובים מתוקנים בתקווה שייתכנסו לפתרון האמיתי. במילים אחרות זוהי שיטה של "ניחוש מושכל" של התוצאה ובכל איטרציה נתקן את הניחוש עד שנתכנס לפתרון האמיתי (לא בהכרח נתכנס לפתרון). מספר האיטרציות שנקבע הוא בהתאם לרמת הדיוק שנרצה.

שיטות איטרטיביות:

1. שיטת יעקבי
2. שיטת גאוס-זיידל
3. שיטת SOR -Successive Over Relaxation

מערכת המשוואות תרשם כך:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \rightarrow \begin{cases} x_1 = \dfrac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}} \\ x_2 = \dfrac{b_2 - a_{21}x_1 - a_{23}x_3}{a_{22}} \\ x_3 = \dfrac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}} \end{cases}$$

נתחיל מניחוש התחלתי לפתרון:

$$\vec{x}^{(0)} = \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ x_3^{(0)} \end{pmatrix}$$

הערה: האינדקס העליון מסמן את מספר האיטרציה
כעת בכל שלב נבצע שיפור לפתרון הנוכחי ונחשב את $x^{(n+1)}$ שהוא פתרון קרוב ומדויק יותר מאשר $x^{(n)}$

1. **שיטת יעקבי**

עדכון הפתרון יבוצע ע"י האיטרציה הכללית:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)$$

לדוגמא:

$$x_1^{(k+1)} = \frac{b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)}}{a_{11}}$$
$$x_2^{(k+1)} = \frac{b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)}}{a_{22}}$$
$$\vdots$$

שלבי האלגוריתם:

0. סדר מערכת נתונה של משוואות לינאריות בצורה דומיננטית באלכסון

1. הגדר ניחושים ראשוניים עבור $X^{k=0} = (x_1{}^0 \quad , x_2{}^0, \ldots x_n{}^0)$

2. עבור $k = 1,2, \ldots . N$:

2.1 עבור $i = 1,2, \ldots n$ חשב את $X^k = (x_1{}^k, x_2{}^k, \ldots x_n{}^k)$ לפי הנוסחא:

$$x_i{}^k = \frac{1}{A_{ii}} \cdot \left[-\sum_{j \neq i}^{n} A_{ij} \cdot x_j{}^{(k-1)} + b_i \right]$$

2.2 אם $tolerance < ||X^k - X^{(k-1)}||$ החזר את X^k // הגענו להתכנסות הרצויה

2.3 אחרת: $X^{(k-1)} = X^k$

2.4 $k = k + 1$

3. הדפס: "הגענו למספר האיטרציות המקסימלי ללא התכנסות"

4. החזר את X^k

פסאודו קוד

Jacobi algorithm

Input: initial guess $x^{(0)}$ to the solution, (diagonal dominant) matrix *A*, right-hand side vector *b*, convergence criterion
Output: solution when convergence is reached
Comments: pseudocode based on the element-based formula above

JacobiMethod (A, b, X0,N, TOL):

```
k = 1
while k ≤ N do
    for i = 1 to n do
        x_i = 0
        for j = 1 to n do
            if j ≠ i then
                x_i = x_i + a_ij · x_j^(k)
        x_i^(k+1) = (b_i − x_i) / a_ii
   if ||x_i^(k+1) − x_i^(k) || < TOL then the output is x_i^(k+1)
   increment k
 end while
```

2. שיטת גאוס-זיידל

נשתמש בפתרון העדכני ביותר בחישובים מאוחרים באיטרציה:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)$$

הערה: האינדקס המסומן באדום מציין את ההבדל משיטת יעקבי

לדוגמא:

$$x_1^{(k+1)} = \frac{b_1 - a_{12}x_2^{(k+1)} - a_{13}x_3^{(k)}}{a_{11}}$$

$$x_2^{(k+1)} = \frac{b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)}}{a_{22}}$$

$$\vdots$$

שלבי האלגוריתם:

קלט: ערכי המטריצה A, ווקטור הנעלמים b. ניחוש התחלתי , N מספר מקסימלי של איטרציות

פלט: הפתרון המשוער $x1,\ldots,xn$

0. סדר מערכת נתונה של משוואות לינאריות בצורה דומיננטית באלכסון
1. הגדר ניחושים ראשוניים עבור $X^{k=0} = (x_1^0 \quad ,x_2^0,\ldots x_n^0)$
2. עבור $k = 1,2,\ldots.N$:

2.1 עבור $i = 1,2,\ldots n$ חשב את $X^k = (x_1^k, x_2^k,\ldots x_n^k)$ לפי הנוסחא:

$$x_i^k = \frac{1}{A_{ii}} \cdot \left[-\sum_{j=1}^{i-1} A_{ij} \cdot x_j^{(k)} - \sum_{j=i+1}^{n} A_{ij} \cdot x_j^{(k-1)} + b_i \right]$$

2.2 אם $||X^k - X^{(k-1)}|| < tolerance$ החזר את X^k // הגענו להתכנסות הרצויה

2.3 אחרת: $X^{(k-1)} = X^k$

2.4 $k = k + 1$

3. הדפס: "הגענו למספר האיטרציות המקסימלי ללא התכנסות"
4. החזר את X^k

פסאודו קוד

Gauss–Seidel algorithm

Input: initial guess $x^{(0)}$ to the solution, (diagonal dominant) matrix A, right-hand side vector b, convergence criterion
Output: solution when convergence is reached
Comments: pseudocode based on the element-based formula above

---

GaussSeidel_Method (A, b, X0,N, TOL):

```
k = 1
while k ≤ N do
    for i = 1 to n do
        x_i = 0
        for j = 1 to n do
            if j ≠ i then
                x_i = x_i + a_ij · x_j^(k+1)
        x_i^(k+1) = (b_i − x_i) / a_ii
   if ||x_i^(k+1) − x_i^(k) || < tolerance then the output is x_i^(k+1)
   increment k
  end while
```

3. שיטת SOR -Successive Over Relaxation

שיטת SOR הינה שיפור (הרחבה) של שיטת גאוס זיידללפתרון של מערכת משוואות לינאריות.

שיטה זו מתכנסת במהירות גבוהה יותר. הנוסחה הכללית לאיטרציה בשיטה זו:

$$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)} \right)$$

- ω הוא פרמטר לגודל הצעד בין האיטרציות. $\omega = 1$ עבור גאוס–זיידל.
- $0 < \omega < 1$ השיטה תקרא underrelaxation והיא מתכנסת לעתים גם כאשר גאוס–זיידל לא מתכנסת (הצעדים בשיטה זו הם קטנים יותר).
- $1 < \omega < 2$ השיטה תקרא overrelaxation והיא עשויה להאיץ התכנסות בהם גאוס–זיידל מתכנסת, במידה והמטריצה נשלטת אלכסונית.
- $\omega \geq 2$ או $\omega \leq 0$ שיטת SOR לא תתכנס.

קלט :

originMatrix - מטריצה המקדמים בגודל [size x size].

originVectorB - וקטור עמודה בגודל [size x 0].

Omega - פקטור ההרפיה (בעל ערך של 1...2).

Accuracy - ערך הדיוק לפתרון.

maxIteration- מספר מקסימלי של איטרציות

לפלט של הפסאודו קוד קיימות שתי אפשרויות :

1. פלט תקין המערכת מתכנסת -ולכן מוחזר לנו וקטור הפתרון.
2. פלט לא תקין המערכת לא מתכנסת - ולכן מוחזר לנו ערך שהוא null .

פסאודו קוד:

```
SOR(originMatrix, originVectorB, Omega, Accuracy)
	Build  prevIteration[size][0] and Initialize with zeros
	Build currentIteration[size][0] and initialize with zeros

	For k = 0…maxIteration
		For I = 0…size
			RowSum = 0
			For j = 0…size
				If I != j
					rowSum = rowSum + originMatrix[i][j] * currentIteration[j][0]

			currentIteration[i][0] = (1 - Omega) * prevIteration[i][0] + (originVectorB[i][0] – rowSum) / originMatrix[i][i]

		Flag = True
		For I = 0…size
			If currentIteration[i][0] – prevIteration[i][0] > Accuracy
				Flag = false

		If flag == true
			Break

		currentIteration = prevIteration

		If k == maxIteration - 1
			currentIteration = null

	Return currentIteration
```