תרגיל בקורס ״תוכנה 1״, פרופ׳ סיון טולדו, סמסטר ב׳ תשס״ו

תרגול מערכים (פירוק משולשי של מטריצות)

המטרה של תרגיל זה היא לתרגל שימוש במערכים בג׳אווה, במסגרת תכנות פרוצדורלי בלבד. את התרגול נבצע בעזרת מימוש אלגוריתם לפתרון מערכות משוואות לינאריות.

האלגוריתם

את התרגול נבצע בעזרת מימוש אלגוריתם לפתרון מערכות משוואות לינאריות על ידי פירוק משולשי של מטריצת המקדמים. הפירוק נקרא פירוק LU והוא בעצם גרסה של אלימינציה של גאוס. מערכת המשוואות כוללת n משוואות ב־n נעלמים $x_1, x_2, \ldots, x_n$,

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

ובניסוח של מטריצות,

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

מערכות משוואות כאלה קשה לפתור, אבל מערכות משוואות משולשיות קל לפתור, בין אם הן משולשיות עליונות או תחתונות. את מערכת המשוואות המשולשית התחתונה

$$\begin{bmatrix} \ell_{11} & & & \\ \ell_{21} & \ell_{22} & & \\ \vdots & & \ddots & \\ \ell_{n1} & \ell_{n2} & \cdots & \ell_{nn} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

(האיברים שמעל האלכסון הראשי, ושאינם מצויינים באופן מפורש, כולם אפס) קל לפתור בהצבה,

$c = b$
for $j = 1, 2, \ldots, n$
  $y_j = c_j / \ell_{jj}$
  for $i = j+1, \ldots n$
    $c_i = c_i - \ell_{ij} y_j$

הלולאה הפנימית מציבה את הערך של y_j שזה עתה חושב במשוואות שנותרו, אם נותרו. את הוקטור c הגדרנו כדי שהאלגוריתם לא ישנה את הערכים של וקטור הקבועים b שקיבלנו כקלט. באופן דומה ניתן לפתור מערכות משוואות שמטריצות המקדמים שלהן משולשיות עליונות (כדאי לכם לכתוב את האלגוריתם המקביל באופן מפורש לפני שאתם ניגשים לתכנות התרגיל).

דרך אחת לפתור מערכות משוואות שבהן מטריצת המקדמים אינה משולשית היא לפרק את מטריצת המקדמים למכפלה של מטריצה משולשית תחתונה L ומשולשית עליונה U כך שיתקיים $A = LU$. פירוק כזה לפעמים קיים ולפעמים אינו קיים. כאשר אינו קיים, קל למצוא פירוק שהוא טוב כמעט באותה מידה, אך לא נדון בזה בתרגיל; נתרכז רק במקרים שבהם הפירוק קיים. ניתן למצוא בקלות אלגוריתם פירוק על ידי כתיבת המשוואות של הפירוק כך שהשורה והעמודה הראשונות בכל מטריצה מפורשות, ושאר המטריצות מייוצגות על ידי אות אחת,

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & & & \\ \vdots & & A' & \\ a_{n1} & & & \end{bmatrix} = \begin{bmatrix} \ell_{11} & \\ \ell_{21} & \\ \vdots & L' \\ \ell_{n1} & \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & & U' & \end{bmatrix}$$

כאשר A', L', ו־U' הן מטריצות בגודל $n-1$ על $n-1$. אם נכפול את המטריצות שמימין נקבל את המשוואות

$$\begin{aligned}
a_{11} &= \ell_{11}u_{11} \\
\begin{bmatrix} a_{21} \\ \dots \\ a_{n1} \end{bmatrix} &= \begin{bmatrix} \ell_{21} \\ \dots \\ \ell_{n1} \end{bmatrix} u_{11} + L' \cdot 0 \\
\begin{bmatrix} a_{12} & \cdots & a_{1n} \end{bmatrix} &= \ell_{11} \begin{bmatrix} u_{12} & \cdots & u_{1n} \end{bmatrix} + 0 \cdot U' \\
A' &= \begin{bmatrix} \ell_{21} \\ \dots \\ \ell_{n1} \end{bmatrix} \begin{bmatrix} u_{12} & \cdots & u_{1n} \end{bmatrix} + L'U'
\end{aligned}$$

האלגוריתם פותר את שלושת המשוואות הראשונות ואז ממשיך ברקורסיה. את המשוואה הראשונה ניתן לפתור על ידי ההשמות $\ell_{11} = 1$ ו־$u_{11} = a_{11}$. את השניה על ידי חלוקה של האיברים $a_{21}, \dots a_{n1}$ ב־u_{11} על מנת לקבל את $\ell_{21}, \dots, \ell_{n1}$, ואת השלישית על ידי העתקת האיברים $a_{12}, \dots a_{1n}$ לתוך $u_{12}, \dots, u_{1n}$ (כי $\ell_{11} = 1$). כעת צריך לפרק ברקורסיה את

$$A'' = A' - \begin{bmatrix} \ell_{21} \\ \dots \\ \ell_{n1} \end{bmatrix} \begin{bmatrix} u_{12} & \cdots & u_{1n} \end{bmatrix}$$

לגורם משולשי L' וגורם משולשי U'. איברי A'' הם פשוט $a''_{ij} = a'_{ij} - \ell_{i+1,1}u_{1,j+1}$. אם נרכיב את כל החלקים הללו לאלגוריתם אחד, נקבל את האלגוריתם הבא:

```
for j = 1,2,...,n
  ℓjj = 1
  ujj = ajj
  for i = j+1,...n
    ℓij = aij/ujj
  for i = j+1,...n
    uji = aji
  for i = j+1,...n
    for k = j+1,...n
      aik = aik − ℓij ujk
```

במימוש הזה השתמשנו באיברי המטריצה A גם כדי לייצג את A'' וכן הלאה, כך שהאלגוריתם משנה את מטריצת הקלט. בתוכנית שתכתבו תצטרכו להימנע משינוי מטריצת הקלט, ולכן תצטרכו להעתיק אותה, כפי שהעתקנו את b ל־c קודם.

לא לכל מטריצה יש פירוק כזה. אם נריץ את האלגוריתם הזה על $A = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ הוא יחלק באפס, ולא קשה להראות שלמטריצה הזו אין פירוק LU כלל. על מנת לבדוק את התוכנית שלכם אתם יכולים להשתמש במטריצות שנתונות בתוכנית השלד שניתן לכם. על מנת לנסות את האלגוריתם על מטריצות נוספות, אתם יכולים להשתמש בכל מטריצה שהאלכסון שלה דומיננטי, כלומר שאיברי האלכסון חיוביים ומקיימים $a_{jj} > \sum_{i \neq j} |a_{ij}|$. למטריצות כאלה תמיד יש פירוק LU.

בהינתן פירוק $A = LU$, קל לפתור מערכות משוואות $Ax = LUx = b$ על ידי מציאת y שמקיים $Ly = b$ ואז מציאת x שמקיים $Ux = y$.

כיצד נייצג וקטורים ומטריצות

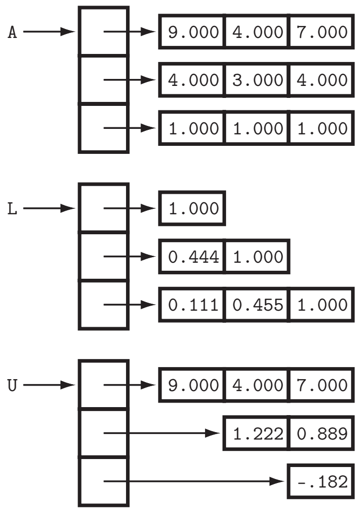
וקטורים ייוצגו על ידי מערכים של n ערכים מטיפוס `double`, כלומר על ידי משתנים מטיפוס `double[]`.

מטריצות נייצג על ידי מערך של n שורות של המטריצה, כאשר כל שורה תיוצג כמו וקטור, על ידי מערך מטיפוס `double[]`. אולם למרות שכל המטריצות שנטפל בהן יהיו ריבועיות, המערכים שמייצגים את השורות לא יהיו תמיד באותו אורך. במטריצת הקלט A כל שורה תיוצג על ידי מערך בגודל n. במילים אחרות, את המטריצה כולה נייצג על ידי n^2 מספרים. במטריצות L ו־U שורה

תיוצג על ידי מערך שגודלו תלוי במספר האיברים שחייבים להיות אפס בשורה. למשל ב-L, השורה הראשונה תיוצג על ידי מערך בגודל 1, השניה על ידי מערך בגודל 2, וכו׳. ב-U השורה הראשונה תיוצג על ידי מערך בגודל n, השניה על ידי מערך בגודל $n-1$, וכן הלאה. למשל, הייצוג של

$$A = \begin{bmatrix} 9 & 4 & 7 \\ 4 & 3 & 4 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1.000 & & \\ 0.444 & 1.000 & \\ 0.111 & 0.455 & 1.000 \end{bmatrix} \begin{bmatrix} 9.000 & 4.000 & 7.000 \\ & 1.222 & 0.889 \\ & & -0.182 \end{bmatrix} = LU$$

(המספרים מעוגלים) יהיה



רמז חשוב: בייצוג הזה, הביטוי `L[i][j]` ניגש לאיבר ℓ_{ij} והביטוי `A[i][j]` ניגש לאיבר a_{ij}, אבל הביטוי שניגש לאיבר u_{ij} קצת יותר מורכב. כדאי למצוא את הביטוי לפני שממשיכים בתרגיל.

התרגיל

עליכם לממש שלוש פונקציות בשלד קיים של תוכנית. את השלד ניתן יהיה להוריד מהאתר.

פונקציה אחת, `solveTriLower` מקבלת מטריצה משולשית תחתונה L (בייצוג שהוסבר, כלומר כארגומנט מטיפוס `[][]double` שהשורות שלו באורך משתנה) ווקטור b (ארגומנט מטיפוס `double[]`). היא צריכה להחזיר וקטור חדש y (שהיא צריכה ליצור ולמלא) שמקיים $Ly = b$. אסור לפונקציה לשנות את ערכי הוקטור b (אבל היא יכולה להעתיק אותו למערך שהיא מקצה, כמובן).

פונקציה שניה, `solveTriUpper`, פותרת מערכת משוואות מהצורה $Ux = y$ עבור x תוך שימוש בממשק דומה לממשק של הפונקציה `solveTriLower`.

הפונקציה השלישית, `decompose`, מקבלת את המטריצה A (ארגומנט מטיפוס `double[][]`) ומחזירה את המטריצות L ו-U של הפירוק שלה. מותר לפונקציה להניח שיש ל-A פירוק LU. מכיון שפונקציות בג׳אווה יכולות להחזיר ערך אחד בלבד (או לא להחזיר כלום, אם הערך המוחזר מוגדר להיות `void`), אנו צריכים למצוא דרך להחזיר גם את L וגם את U ביחד. למרבה המזל, מכיון ששתיהן מיוצגות על ידי מערך של מערכים של `double`-ים, זה קל: נחזיר מערך בגודל 2 שהאיבר הראשון שלו הוא L והאיבר השני שלו הוא U. כלומר נחזיר ערך מטיפוס `double[][][]`.

כדי לסייע לכם במימוש הפונקציות הללו (ובמציאת שגיאות במימושים שלכם/ן), תוכנית השלד כוללת כמה פונציות עזר שאתם יכולים להשתמש בהן. הפונקציה `copy` מעתיקה מטריצה בייצוג שתיארנו. הפונקציות `printSquareMatrix`, `printTriLower`, ו-`printTriUpper` מדפיסות מטריצות בייצוג הזה (ריבועיות, משולשיות תחתונות, ומשולשיות עליונות, בהתאמה). הפונקציה `toString` מחזירה ייצוג של וקטור כמחרוזת. הפונקציה `compare` בודקת האם שני וקטורים דומים בערכיהם (בגלל שגיאות העיגול שמתרחשות כאשר המחשב מבצע חישובים על ערכים מטיפוס `double`, אי אפשר לצפות שהפלט יהיה מדוייק לחלוטין גם אם מימוש האלגוריתם נכון; שגיאות עיגול אינן מלמדות בהכרח על מימוש לא נכון). כדאי לקרוא את המימושים של הפונקציות הללו ולנסות להבין אותן, אבל אין צורך לשנות אותן.

בהצלחה.