Методи нумеричні - Лабораторна робота 3
Уклади лінійних рівнянь

Увага:Інструкції що стосуються перебігу лабораторних занять складені в частині 2 цього документа.

1 Теоритична частина

- Стислі методи
  - зразки Cramera
  - вибіркові методи:
    - метод ліквідації Gaussa
    - метод Gaussa-Jordana
  - декомпозиційні методи:
    - розклад LU за допомогою ліквідації Gaussa
    - метод Gaussa-Doolittle'a
    - метод Gaussa-Crouta
    - метод Choleskiego (Choleskiego-Crouta/Banachiewicza/корінь квадратний) (для семетричної матриці)
- методи ітераційні (наближені)
  - метод Jacobiego (iteracji prostej)
  - метод Gaussa-Seidla
  - інше.

На заняттях ми зважуватимемо проблему знаходження рішення нижче згаданого укладу лінійних рівнянь (URL)

$$\begin{array}{ccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n = b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n = b_2 \\ & & & & \vdots & & \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \dots & + & a_{nn}x_n = b_n \end{array} \tag{1}$$

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

$$\mathbf{A}\ \mathbf{x} = \mathbf{b} \tag{3}$$

1.1 Безпосередні методи

1.1.1 Метода ліквідації Gaussa

З метою представлення ліквідації Gaussa запишемо уклад(2) у вигляді

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \dots & a_{1n}^{(0)} \\ a_{21}^{(0)} & a_{22}^{(0)} & \dots & a_{2n}^{(0)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(0)} & a_{n2}^{(0)} & \dots & a_{nn}^{(0)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,n+1}^{(0)} \\ a_{2,n+1}^{(0)} \\ \vdots \\ a_{n,n+1}^{(0)} \end{bmatrix} \tag{4}$$

де вартості $a_{i,n+1}^{(0)}$ то вільні вирази, які відповідають вартостям b_i з рівняння (2), натомість . Якщо $a_{11} \neq 0$ відняти перше рівняння, помножене через a_{i1}/a_{11} від інших рівнянь ($i = 2, 3, \dots, n$). Отримається тоді редукований уклад, в якому всі елементи першої колони нижче першого рядка будуть нулями:

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \dots & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(1)} & \dots & a_{nn}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,n+1}^{(0)} \\ a_{2,n+1}^{(1)} \\ \vdots \\ a_{n,n+1}^{(1)} \end{bmatrix} \tag{5}$$

$a_{ij}^{(1)} = a_{ij}^{(0)} - \frac{a_{i1}^{(0)}}{a_{11}^{(0)}} a_{1j}^{(0)}$, $i = 2, 3, \dots, n$, $j = 2, 3, \dots, n$.

У наступному кроці (оскільки $a_{22}^{(1)} \neq 0$),віднімаємо друге рівняння укладу (5) помножене на $a_{i2}^{(1)}/a_{22}^{(1)}$ від рівняння $3, 4, \dots, n$. Отримується тоді другий редукований уклад:

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} & \dots & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \dots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \dots & a_{2n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{n3}^{(2)} & \dots & a_{nn}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,n+1}^{(0)} \\ a_{2,n+1}^{(1)} \\ a_{3,n+1}^{(2)} \\ \vdots \\ a_{n,n+1}^{(2)} \end{bmatrix} \tag{6}$$

де $a_{ij}^{(2)} = a_{ij}^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} a_{2j}^{(1)}$, $i = 3, \dots, n$, $j = 3, \dots, n$.

Виконуючи аналогічні перетворення для нижче розташованих рядів, по $n-1$ кроках отримається уклад з верхньою трикутною матрицею:

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \dots & a_{1,n-1}^{(0)} & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & \dots & a_{2,n-1}^{(1)} & a_{2n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{n-1,n-1}^{(n-2)} & a_{2n}^{(n-2)} \\ 0 & 0 & \dots & 0 & a_{nn}^{(n-1)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_3 \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,n+1}^{(0)} \\ a_{2,n+1}^{(1)} \\ \vdots \\ a_{3,n+1}^{(n-2)} \\ a_{n,n+1}^{(n-1)} \end{bmatrix} \tag{7}$$

Узагальнюючи, уклад (7) отримуємо створену так звану розширену матрицю $\mathbf{A_r}$ укладу (2) що виникла через сполучення матриці $\mathbf{A}$ і вектора вільних виразів $\mathbf{b}$ ($\mathbf{A_r} = [\mathbf{A}\mathbf{b}]_{n_x n+1}$). Потім на тій матриці робиться серія перетворень (заміщання рядів через їх лінійні комбінації), в такий спосіб, щоб обнуляти наступні елементи матриці нижче головної діагоналі, тобто згідно із зразком:

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)} a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}}$$
$$k = 1, 2, \dots, n-1$$
$$j = k+1, \dots, n+1$$
$$i = k+1, \dots, n$$

Після отримання укладу (7) розв'язок можна отримати вчиняючи зворотну редукцію. Вартість x_n безпосередньо з останнього рівняння укладу (7): $x_n = c_n/t_{nn}$. Тоді в інших рівняннях n- колону можемо перенести на праву сторону рівняння, зменшити розмір URL до $n-1$ x $n-1$ і знову в легкий спосіб вирахувати невідому x_{n-1}. Таку схему повторюємо, поки не отримаємо всі шукані вартості.

Dane: A, b, N
Wyniki: Ur
  $Ur \leftarrow [Ab]$;
  for $k \leftarrow 1$ to $N-1$ **do**
    if $A(k,k) = 0$ **then**
      return error;
    end if
    for $j \leftarrow k+1$ to $N+1$ **do**
      for $i \leftarrow k+1$ to N **do**
        $Ur(i,j) = Ur(i,j) - \frac{Ur(i,k)}{Ur(k,k)} \cdot Ur(k,j)$;
      end for
    end for
  end for

Algorytm 1: Метод ліквідації Gaussa

Dane: $\mathbf{U}r$, N
Wyniki: $\mathbf{x}$
  $\mathbf{x} \leftarrow \mathbf{0}$;
  for $i \leftarrow N$ to 1 **do**
    $S \leftarrow 0$
    for $j \leftarrow i+1$ to N **do**
      $S \leftarrow S + \mathbf{U}r(i,j) \cdot x(j)$;
    end for
    $x(i) \leftarrow \frac{\mathbf{U}r(i,n+1) - S}{\mathbf{U}r(i,i)}$;
  end for

Algorytm 2: Зворотна редукція

Вище згадані дії можна записати як:

$$
\begin{aligned}
x_n &= a_{n,n+1}^{(n-1)} / a_{nn}^{(n-1)} \\
x_i &= \frac{1}{a_{ii}^{(n-1)}} \left(a_{i,n+1}^{(i-1)} - \sum_{j=i+1}^{n} a_{ij}^{(i-1)} x_j \right) \quad dla \quad i = n-1, n-2, ..., 1
\end{aligned}
\tag{8}
$$

1.1.2 Метод Gaussa-Jordana

В ліквідації Gaussa обнуляємо тільки ніжчі елементи (або вищі) діагоналі головної матриці. Модифікація вище згаданого алгоритму, полягає у обнуленні однаково нижніх і вищих елементів діагоналі, веде до методу Gaussa-Jordana. Метод має перевагу над звичайною ліквідацією Gaussa, не потрібна зворотна редукція з метою отримання рішення. Матриця укладу перетворювана до одиничної матриці, і рішення отримується в деякій мірі 'автоматично' в останній колоні матриці $\mathbf{A_r}$ (вектор вільних виразів).

У кожному k- кроку виконання алгоритму Gaussa-Jordana, k-рядок матриці $\mathbf{A_r}$ поділенний на елементи $a_{kk}^{(k-1)}$. Наступний рядок цього помножений на $a_{i1}^{(0)}$ є віднятий від кожного i- того рядка ($i \neq k$), так щоб обнулили всі елементи k колони поза елементом, лежачим наголовній діагоналі.

Так отже, перетворення укладу рівнянь (4) по першому кроці цього методу дає уклад постаті:

$$
\begin{bmatrix}
1 & a_{12}^{(1)} & \dots & a_{1n}^{(1)} \\
0 & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\
\vdots & \vdots & \ddots & \vdots \\
0 & a_{n2}^{(1)} & \dots & a_{nn}^{(1)}
\end{bmatrix}
\cdot
\begin{bmatrix}
x_1 \\ x_2 \\ \vdots \\ x_n
\end{bmatrix}
=
\begin{bmatrix}
a_{1,n+1}^{(1)} \\ a_{2,n+1}^{(1)} \\ \vdots \\ a_{n,n+1}^{(1)}
\end{bmatrix}
\tag{9}
$$

У наступному кроці друге рівняння ми ділимо двохсторонньо на $a_{22}^{(1)}$ і від i- рядка ($i = 1,\ 3, \dots, n$) віднімаємо другий рядок помножений на $a_{i2}^{(1)}$. У такий спосіб отримуємо уклад:

$$
\begin{bmatrix}
1 & 0 & \dots & a_{1n}^{(2)} \\
0 & 1 & \dots & a_{2n}^{(2)} \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \dots & a_{nn}^{(2)}
\end{bmatrix}
\cdot
\begin{bmatrix}
x_1 \\ x_2 \\ \vdots \\ x_n
\end{bmatrix}
=
\begin{bmatrix}
a_{1,n+1}^{(2)} \\ a_{2,n+1}^{(2)} \\ \vdots \\ a_{n,n+1}^{(2)}
\end{bmatrix}
\tag{10}
$$

Наприкінці, по $n-1$ кроках ліквідації уклад перетворюваний до постаті

$$
\begin{bmatrix}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \dots & 1
\end{bmatrix}
\cdot
\begin{bmatrix}
x_1 \\ x_2 \\ \vdots \\ x_n
\end{bmatrix}
=
\begin{bmatrix}
a_{1,n+1}^{(n)} \\ a_{2,n+1}^{(n)} \\ \vdots \\ a_{n,n+1}^{(n)}
\end{bmatrix}
\tag{11}
$$

де права сторона укладу є рішенням.

Алгоритм методу Gaussa-Jordana можна описати наступним чином:

- Утвори розширену матрицю $\mathbf{A_r}$ для укладу $\mathbf{Ax} = \mathbf{b}$
- Для кожного k-такого рядка ($k = 1, 2, \ldots, n$) матриці $\mathbf{A_r}$:
  - Для кожного j-такого елементу k-того рядка, поділи j-той елемент на елемент $\mathbf{A_r}(\mathbf{k}, \mathbf{k})$

$$\begin{aligned} a_{kj}^{(k)} = &\quad a_{kj}^{(k)} / a_{kk}^{(k)} \\ j = &\quad 1, 2, \ldots, n+1 \end{aligned} \tag{12}$$

  - Для кожного i-такого ($i = 1, 2, \ldots, n,\ i \neq k$) рядка
    Для $j = 1, 2, \ldots, n+1$

$$\begin{aligned} \lambda_i &= \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} \\ a_{ij}^{(k)} &= a_{ij}^{(k-1)} - \lambda a_{kj}^{(k-1)} \end{aligned} \tag{13}$$

1.1.3 Розклад LU, метод Gaussa-Doolittle'a

Іншим способом рішення URL є декомпозиційні методи. Опираються вони на перетворення матриці **A** на добуток двох трикутних матриць: нижньої **L** (ang. lower) верхньої **U** (ang. upper):

$$A = LU \tag{14}$$

Щоб розклад був однозначний засновується, що всі елементи діагоналі матриці L або U рівні 1 (метод Gaussa-Doolittle'a або Gaussa-Crouta).

Зважаючи на (14) і (3) отримуємо

$$LUx = b \tag{15}$$

Таким чином розв'язок URL можна поділити на 2 етапи. В першу чергу розв'язується

$$Ly = b \Rightarrow y = L^{-1}b \tag{16}$$

а потім

$$Ux = y \Rightarrow x = U^{-1}y \tag{17}$$

Розклад LU за допомогою методу Gaussa-Doolittle'a описується нижче виразом:

$$\begin{aligned} &u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \qquad dla \quad j = i, i+1, \ldots, n \\ &l_{ji} = \frac{1}{u_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} l_{jk} u_{ki} \right) \quad dla \quad j = i+1, i+2, \ldots, n \\ &i = 1, 2, \ldots, n \end{aligned}$$

Dane: n, (a_{ij})
Wyniki: l_{ij}, u_{ij}
  for $k \leftarrow 1$ to n **do**
    $l_{kk} \leftarrow 1$;
    for $j \leftarrow k$ to n **do**
      $u_{kj} \rightarrow a_{kj} - \sum_{s=1}^{k-1} l_{ks} U_{sj}$
    end for
    for $i \rightarrow k+1$ to N **do**
      $l_{ik} \rightarrow a_{ik} - \sum_{s=1}^{k-1} l_{is} U_{sk} / u_{kk}$
    end for
  end for

Algorytm 3: Розклад LU за допомогою методу Gaussa-Doolittle'a

1.1.4 Вибір основних елементів

У всіх вищезгаданих методах існує можливість виникнення нульових, або дуже малих елементів на діагоналі матриці **A**, що є приводом нестабільності нумеричного алгоритму. Тому також на практиці застосовується їх змодифікована версія. Засновується, що в кожній ітерації до ліквідації елементів вибирається елемент з найбільшою щодо модуля вартістю (ang. pivoting). Вирізнити можна два способи вибору

- частковий вибір основного елементу - полягає у вишукуванні перед кожним k-тим кроком мінливих ліквідацій найбільшого щодо модуля елементу з числа елементів, що знаходяться нижче k-тої колони. Після установлення номера r рівняння, в якому цей елемент виступає, рядок при номері r замінюваний є місцем з рядком при номері k. Чинності переставлення рядків мусить супроводжувати зміна знаків в довільному з числа рівнянь, що переставляються, якщо визначник матриці напівчинник укладу, що виник, в результаті переставлення рядків має бути рівний визначникові матриції факторів вихідного укладу.

- повний вибір основного елементу – в k- кроці знайдений є найбільший щодо модуля елемент в цілій матриці [...]. Після установлення номера r рядка і номера s колони наступає перестановка рядка при номер і r з вершем при номері k, а потім колони при номері s з колонами при номері k. Число перестановок рядків і колон є запам'ятана і враховується при обчисленні визначника фікторів матриці.

1.2 Методи ітераційні

1.2.1 Метод Jacobiego (метод простої ітерації)

Зміняємо уклад (1) до постаті:

$$\begin{array}{lllllllll} x_1 & = & c_1 & + & & & d_{12}x_2 & + \ \ldots \ + & d_{1n}x_n \\ x_2 & = & c_2 & + & d_{21}x_1 & + & & \ldots \ + & d_{2n}x_n \\ & & & & & & \vdots & & \\ x_n & = & c_n & + & d_{n1}x_1 & + & d_{n2}x_2 & + \ \ldots & \end{array} \tag{18}$$

де

$$\begin{aligned} c_i &= b_i/a_{ii} \quad dla \quad i = 1, 2, ..., n \\ d_{ij} &= -a_{ij}/a_{ii} \quad dla \quad i = 1, 2, ..., n; j = 1, 2, ..., n; i \neq j \end{aligned} \tag{19}$$

Приймаючи

$$C = \begin{bmatrix} c_1 \\ c_2 \\ \ldots \\ c_3 \end{bmatrix} \text{ oraz } D = \begin{bmatrix} 0 & d_{12} & \ldots & d_{1n} \\ d_{21} & 0 & \ldots & d_{2n} \\ \ldots & & & \\ d_{n1} & d_{n2} & \ldots & 0 \end{bmatrix} \tag{20}$$

уклад (18) можна записати матрично

$$x = C + Dx \tag{21}$$

На підставі останнього рівняння сконструйований ряд наближень

$$x^{(k+1)} = C + Dx^{(k)} \tag{22}$$

де за початковий вектор x^0 довільний, найчастіше приймається $x^0 = C$.

Кажучи інакше, $(k+1)$ приближення i-тієї невідомої укладу рівнянь можна вирахувати за допомогою

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1, j\neq i}^{n} a_{ij} * x_j^{(k)}}{a_{ii}} \tag{23}$$

Як умову кінцевого виконання ітерації можна подати:

- $\sum_{i=1}^{n} |x_i^{(k+1)} - x_i^{(k)}| < tol$, або
- $\sum_{i=1}^{n} |\tilde{b}_i^{(k)} - b_i| < tol$, де $\tilde{b}^{(k)} = Ax^{(k)}$

де tol – задана докладність.

Необхідною умовою збіжності ряду наступних наближень є:

$$\wedge_{i\in\{1,2,\ldots,n\}} |a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \tag{24}$$

або

$$\wedge_{j\in\{1,2,\ldots,n\}} |a_{jj}| > \sum_{i=1, i\neq j}^{n} |a_{ij}| \tag{25}$$

Таблица 1: Функції імплонтуючі операції на матрицях

Функція	Опис
det(A)	визначник матриці
cond(A)	число обумовленості матриці
norm(A,)	норма матриці
inv(A) A^-1	обернена матриця
pinv(A)	псевдообернена матриця
lu(A)	розклад LU матриці
chol(A)	розклад Choleskiego
eig(A)	власні вартості
qr(A)	розклад QR матриці

1.3 Octave/Matlab і URL

В програмах типу Matlab існує показна бібліотека функції можливих розвинених операцій на матрицях і векторах, завдяки чому, у багатьох випадках зайве є письмо власних процедур. Їх частина разом з коротким описом заключає Таблиця (1).

Згідно з теорією, щодо рішення URL, записаного у вигляді $Ax = b$ в програмі Matlab/Octave можна скористатися порадою

```
>> x = inv(A) * b
```

або

```
>> x = A^-1 * b
```

На практиці рекомендовано є вживання лівобічного оператора ділення

```
>> x = A \ b
```

оскільки вище згадана дія набагато швидша, ніж розв'язування URL через явне відвертання матриці.

1.4 Завдання

В програмі Octave познайомся з дією і способами виклику нижче поданих функцій:

- size
- sum
- triu
- diag

2 Вправи

Під час занять ми шукатимемо рішення укладу рівнянь лінійних постатей :

$$\begin{array}{rcrcrcr} 3x_1 & + & x_2 & - & x_3 & = & 6 \\ -x_1 & + & 5x_2 & - & x_3 & = & 10 \\ 2x_1 & + & 4x_2 & + & 8x_3 & = & 2 \end{array} \tag{26}$$

1. Створи документ `main.m`, а в ньому сформолюй змінні `A` і `b` що є відповідно матрицею факторів і вектором вільних виразів укладу (26).

2. Розв'яжи вище згаданий уклад рівнянь за допомогою оператора лівобічного ділення (рішенням є цілі числа).

3. На підставі алгоритмів 1 і 2 напиши (у окремій папці) функцію розв'язку URL за допомогою методу ліквідації Gaussa без вибору основного елементу. Постать її виклику то:

   ```
   [x, Ur] = f_gauss(A,b)
   ```

   де `A` – квадратна матриця факторів, `b` – вектор вільних виразів, `x` – шуканих (рішення URL), `Ur` – розширена матриця укладу, перетворена до трикутної верхньої матриці.

   Увага: Матриця `Ur` утворена згідно з алгоритмом 1 не буде матрицею з нулями нижче (тих елементів невикористовується на подальшому етапі обчислень)діагоналі.

   Увага 2: : Перед початком головного циклу алгоритму зворотньої редукції вектор **x належить** відповідно започаткувати (**x** то не скаляр, тільки вектор нульовий при 1 колоні і n рядах, де n то ступінь матриці **A**).

   Uwaga 3: У Octave оператор `==` то не то саме `=`.

   Увага 4 (Для тих, які ужили функцію `size` то визначення ступеня матриці): Прошу звернути увагу на те що повертає ця функція залежно від поданих вхідних і вихідних параметрів (→ `help size`).

4. У середні skrypu `main.m` виклич написану функцію `f_gauss` на користь змінних `A`, `b`. Коректність функції можна перевірити порівнюючи її результат зрішенням, отриманим в завдан. 2.

5. Зроби розклад матриці `A` на матриці трикутну нижню і вищу за допомогоб функції програми Matlab/Octave `lu`. Постаті виклику цієї функції можна перевірити вписуючи `help lu` в лінії команд.

6. На підставі алгоритму 3 zaimplementuj розклад LU за допомогою методу Gaussa-Doolittle'a без вибору основного елементу. Виклик функції повинен виглядати наступним чином:

   ```
   [L, U] = f_gauss_doolittle(A)
   ```

7. Перевір коректність заімплентованої через функцію, порівнюючи повернений через неї результат з результатом виклику функції `[L,U]=lu(A)`.

8. Напишу функцію, яка розв'язує URL за допомогою методу простої ітерації (розділ1.2.1 → зразки (20),(22)). Її декларація повинна виглядати:

   ```
   [xx, kk] = f_jacobi(A,b, k_max, tol)
   ```

   де `A` – квадратна матриця векторів, `b` – вектор вільних виразів, `k_max` - максимальна кількість ітерацій, `tol` – докладність розв'язку `xx` – вартість шуканих (розв'язок URL), `kk` – номер ітерації на якій закінчено обчислення.

9. Розв'яжи уклад рівнянь із завдання 1за допомогою функції `f_jacobi`. Чи цей уклад рівнянь виконує умови збіжності методу простої ітерації? Подай приклад такого URL, для якого метод Jacobiego розходитиметься. Перевір дію свого методу для цього прикладу.

10. Напиши функцію `f_gauss_jordan` на підставі алгоритму,описаного в розділі 1.1.2 (зразки (12),(13)). Постать її виклику то:

    ```
    [x] = f_gauss_jordan(A,b)
    ```

    де A - квадратна матриця факторів, b - вектор виразів вільних, x - вектор шуканих (рішення URL).

11. (Для охочих) Перепиши функції `f_gauss`, `f_gauss_jordani` `f_jacobi`, так щоб обчислення виконувались на цілих рядках (оператор :), не на кожному елементі рядка окремо.

12. ((Для охочих) Спробуй розв'язати за допомогою написаної собою функції `f_gauss` уклад лінійних рівнянь $Ax = b$, де

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

Його розв'язком є вектор $x = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}$. Що є приводом для помилкової дії функції?? Доповни алгоритм про частину, імплементуючу вибір часткового основного елементу, щоб функція давала правильний результат.