محدثه حکیمی - ۴۰۱۲۴۳۰۴۵

گزارش تمرین کامپیوتری ۲

سؤال اول:

توی این سؤال باید دستگاه معادلات داده شده رو به دو روش کرامر و ماتریس معکوس حل کنیم.
اول از همه ورودی گرفتن و ذخیره‌ی ضرایب(coefficients) و ثابت‌ها(constants):

```
n = int(input("Enter the number of variables: "))
coefficients = []
constants = []

for i in range(2 * n):
    if i % 2 == 0:
        nums = map(int, input("Enter the coefficients of equation " + str(i // 2 + 1) + ": ").split())
        coefficients.append(list(nums))
    else:
        constants.append(int(input("Enter the constant of equation " + str(i // 2 + 1) + ": ")))
```

که به صورت یکی در میون ضرایب و عدد ثابت هر معادله رو از یوزر می‌گیریم.

- روش اول - ماتریس معکوس یا وارون:

حل معادلات با وارون

اگر در معادله $AX=B$، $m=n$ و A ناتکین، آنگاه

$$\underset{n\times n}{A}\,\underset{n\times 1}{X}=\underset{n\times 1}{B} \Rightarrow A^{-1}(AX)=A^{-1}B$$

$$\Rightarrow \underset{n\times 1}{X}=\underset{n\times n}{A^{-1}}\,\underset{n\times 1}{B}$$

پس درواقع باید ماتریس معکوس ضرایب رو به دست بیاریم و در ماتریس ثابت‌ها ضرب کنیم.
اما برای به دست آوردن ماتریس معکوس، داریم:

یافتن وارون

روش اول: با کمک ماتریس الحاقی (Adjoint Matrix)

$$C_{ij}=(-1)^{i+j}M_{ij}$$

M_{ij} همان ماتریسی است که از حذف سطر i ام و ستون j ام بدست می‌آید.

$$\text{ماتریس الحاقی } A = \operatorname{adj} A = \begin{bmatrix} C_{11} & \cdots & C_{1n} \\ \vdots & & \vdots \\ C_{n1} & \cdots & C_{nn} \end{bmatrix}^T = \begin{bmatrix} C_{11} & \cdots & C_{n1} \\ \vdots & & \vdots \\ C_{1n} & \cdots & C_{nn} \end{bmatrix}$$

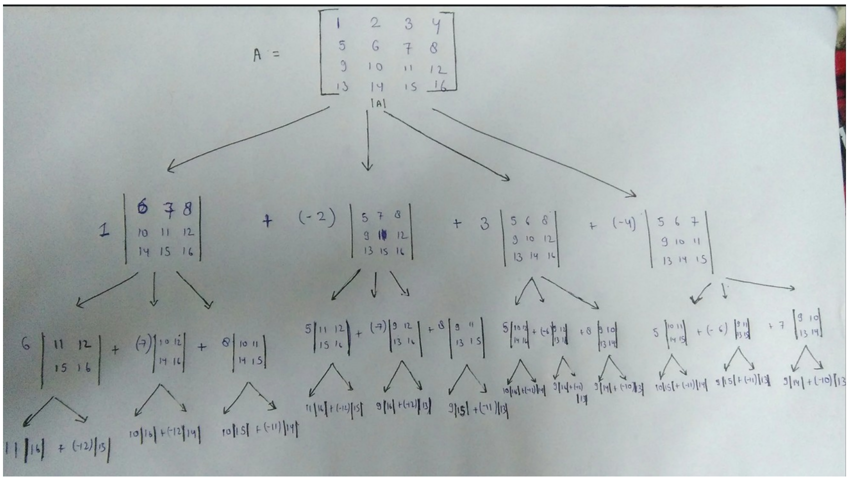
قضیه: اگر A ماتریس $n\times n$ باشد و $\det A \neq 0$ آنگاه

$$A^{-1}=\frac{1}{\det A}\operatorname{adj} A$$

یعنی در واقع، به سه تابع دترمینان، ترانهاده و وارون‌کننده نیاز داریم که هر کدوم رو اینجا میارم:

```python
def recursive_determinant(matrix, n):
    # use first row to calculate determinant
    if n == 1:
        return matrix[0][0]
    elif n == 2:
        return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]
    else:
        det = 0
        sign = 1
        for c in range(n):
            det += sign * matrix[0][c] * recursive_determinant([row[:c] + row[c + 1:] for row in matrix[1:]], n - 1)
            sign *= -1
        return det
```

این تابع دترمینانه که به روش بازگشتی دترمینان رو محاسبه می‌کنه. منطقش هم که ساده‌ست و عملاً از روی فرمول دترمینان جلو می‌ره، بدین شکل:



درواقع شروط پایه ماتریس‌های ۱*۱ و ۲*۲ هستن و ماتریس‌های بزرگ‌تر از محاسبه‌ی همین‌ها به دست میان.

فرمول استفاده شده هم که اینه:

نکته: می‌توان این تعریف را برای ماتریس $n \times n$ بسط داد:

$$\det A = a_{i1}C_{i1} + a_{i2}C_{i2} + \cdots + a_{in}C_{in}$$

$$1 \leq i \leq n$$

تابع دوم، ترانهاده‌ست که خلاصه و جمع و جوره:

```python
def transpose(matrix):
    t = [[0 for i in range(len(matrix))] for j in range(len(matrix[0]))]
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            t[j][i] = matrix[i][j]
    return t
```

و فقط درایه‌های ماتریس رو نسبت به قطر اصلی بازتاب می‌ده.

تابع سوم که بهش نیاز داریم هم معکوس‌کننده‌ست که ماتریس رو وارون می‌کنه. طبق فرمولی که بالاتر گذاشتم، باید اول ترانهاده‌ی ماتریس الحاقی رو به دست بیاریم و تقسیم بر دترمینان بکنیم:

```
def find_inverse_matrix(matrix):
    # use determinant of the matrix to find the inverse matrix
    det = recursive_determinant(matrix, len(matrix))
    if det == 0:
        return None
    else:
        adj_matrix = []
        for i in range(len(matrix)):
            adj_matrix.append([])
            for j in range(len(matrix)):
                element = (-1) ** (i + j) * recursive_determinant([row[:j] + row[j + 1:] for row in matrix[:i] + matrix[i + 1:]],
                                                                  len(matrix) - 1)
                adj_matrix[i].append(element)
    # transpose the adjoint matrix
    adj_matrix = transpose(adj_matrix)
    # multiply the adjoint matrix by 1/determinant
    for i in range(len(matrix)):
        for j in range(len(matrix)):
            adj_matrix[i][j] /= det
    return adj_matrix
```

حالا با کمک این ۳تا تابع، تابع اصلی یعنی حل کردن دستگاه معادلات به روش ماتریس معکوس رو می‌نویسیم که بالاتر فرمولش رو آوردم($X=A^{-1}B$):

```
def solve_equation_using_inverse_matrix(coefficients, constants):
    inverse_matrix = find_inverse_matrix(coefficients)
    if inverse_matrix is None:
        return None
    else:
        solution = []
        for i in range(len(inverse_matrix)):
            x = 0
            for j in range(len(inverse_matrix[0])):
                x += inverse_matrix[i][j] * constants[j]
            solution.append(x)
        return solution
```

• روش دوم – قانون کرامر:

قانون کرامر (Cramer's Rule)

برای دستگاه معادلات با n معادله و n مجهول

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n = b_1 \\ \vdots \qquad\qquad \vdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n = b_n \end{cases}$$

کروشه کنم:

$$A_K = \begin{bmatrix} a_{11} & \cdots & a_{1k-1} & \boxed{b_1} & a_{1k+1} & \cdots & a_n \\ & & & \vdots & & & \\ a_{n1} & \cdots & a_{nk-1} & \boxed{b_n} & a_{nk+1} & \cdots & a_n \end{bmatrix}$$

قضیه: اگر A ماتریس ضرایب معادله فوق باشد، و $\det A \neq 0$ آنگاه

جواب دستگاه معادلات به صورت ذیل است:

$$x_1 = \frac{\det A_1}{\det A}, \quad x_2 = \frac{\det A_2}{\det A}, \ldots, x_n = \frac{\det A_n}{\det A}$$

پس برای به دست‌آوردن جواب‌ها کافیه دترمینان ماتریس اصلی و دترمینان A_k ها رو به دست بیاریم که به کمک تابع دترمینان انجامش می‌دیم:

```python
def solve_equation_using_crammer_rule(coefficients, constants):
    det = recursive_determinant(coefficients, len(coefficients))
    if det == 0:
        return None
    else:
        for i in range(len(coefficients[0])):
            A_k = [row[:] for row in coefficients]
            # replace k-th column with constants
            for j in range(len(coefficients)):
                A_k[j][i] = constants[j]
            det_k = recursive_determinant(A_k, len(coefficients))
            x = det_k / det
            print(f'x{i + 1} = {x:.1f}')
```

در آخر هم اجرای توابع:

```python
# solution 1
print("Solution using Crammer's Rule:")
solve_equation_using_crammer_rule(coefficients, constants)

# solution 2
print("Solution using Inverse Matrix:")
solution = solve_equation_using_inverse_matrix(coefficients, constants)
if solution is None:
    print("No solution")
else:
    for i in range(len(solution)):
        print(f'x{i + 1} = {solution[i]:.1f}')
```

تست ۱:

```
Solution using Crammer's Rule:
x1 = 13.0
x2 = -12.3
x3 = -7.4
Solution using Inverse Matrix:
x1 = 13.0
x2 = -12.3
x3 = -7.4
```

مثال:

$$3x_1+2x_2+x_3=7$$
$$x_1-x_2+3x_3=3$$
$$5x_1+4x_2+2x_3=1$$

$$\det A = 13, \quad \det A_1 = \begin{vmatrix} 7 & 2 & 1 \\ 3 & -1 & 3 \\ 1 & 4 & 2 \end{vmatrix} = -39$$

$$\det A_2 = 78, \quad \det A_3 = 52$$

$$\Rightarrow x_1 = -3, \quad x_2 = 6, \quad x_3 = 4$$

تست ۲:

```
Solution using Crammer's Rule:
x1 = 6.0
x2 = -0.3
Solution using Inverse Matrix:
x1 = 6.0
x2 = -0.3
```

مثال:

$$\begin{cases} 2x_1 - 9x_2 = 15 \\ 3x_1 + 6x_2 = 16 \end{cases}$$

$$\underbrace{\begin{bmatrix} 2 & -9 \\ 3 & 6 \end{bmatrix}}_{A} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 15 \\ 16 \end{bmatrix}$$

$$A^{-1} = \frac{1}{39} \begin{bmatrix} 6 & 9 \\ -3 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \frac{1}{39} \begin{bmatrix} 6 & 9 \\ -3 & 2 \end{bmatrix} \begin{bmatrix} 15 \\ 16 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ -\frac{1}{3} \end{bmatrix}$$

سؤال دوم:
برای این سؤال باید الگوریتمی رو پیاده کنیم که رنک ماتریس رو محاسبه می‌کنه.
من به کمک فرم سطری اچلون که سر کلاس هم گفته شده بود رنک رو محاسبه کردم.

گرفتن ورودی:

```python
import numpy as np

m = int(input("Enter the number of rows(m): "))
n = int(input("Enter the number of columns(n): "))

print("Enter the elements of the matrix: ")
matrix = []
for i in range(m):
    a = []
    for j in range(n):
        a.append(int(input()))
    matrix.append(a)
```

- الگوریتم:

اول از همه چک می‌کنیم که آیا تمام درایه‌های ماتریس صفر هستن یا نه. اگر صفر بود که هیچ، وگرنه اولین ستون غیرصفر از سمت چپ رو پیدا می‌کنیم. بعد سطر اول رو تقسیم بر مقدار درایه‌ی سطر و ستون اول می‌کنیم تا مقدارش برابر با یک بشه. حالا باید درایه‌های پایین اون درایه همگی صفر بشن. اینجا می‌تونیم از اون درایه‌ای که یک کردیم استفاده کنیم تا درایه‌های پایین رو صفر کنیم. بدین صورت که اون سطر مربوطه رو منهای درایه *سطر اول می‌کنیم. این مراحل رو برای سطرهای بعدی هم انجام می‌دیم تا به فرم اچلون برسیم. درواقع هر بار تابع رو به طور بازگشتی صدا می‌زنیم. در آخر هم تعداد سطرهای غیرصفر رو می‌شماریم تا رنک رو به دست بیاریم.

- یه توضیحی هم لازمه بدم. الگوریتمی که من پیاده کردم درواقع row echelon form هست، نه reduced row echelon form. درواقع فرم سطری **کاهش‌یافته** نیست. برای همین هم ممکنه توی تست‌ها دیده بشه که ماتریس نهایی لزوماً درایه‌هاش صفر و یک نیستن ولی رنک درست محاسبه شده.

تابعی که ماتریس رو به فرم اچلون درمیاره و در آرایه‌ی ref_matrix ذخیره می‌کنه:

```python
def row_echelon_form(matrix, m, n, ref_matrix):
    # if matrix == 0, return 0
    if np.all(matrix == 0):
        ref_matrix = np.vstack((ref_matrix, matrix))
        return ref_matrix
    # find leftmost non-zero column
    pivot_col = 0
    for i in range(n):
        if matrix[0][i] != 0:
            pivot_col = i
            break
    # divide the row by the pivot element to achieve one in the pivot position
    matrix[0] = matrix[0] / matrix[0][pivot_col]
```

```python
    # eliminate(zero) the elements below the pivot element
    for i in range(1, m):
        matrix[i] = matrix[i] - matrix[i][pivot_col] * matrix[0]
    # add the first row to the ref_matrix, change -0 to 0
    try:
        matrix[0] = np.where(matrix[0] == -0, 0, matrix[0])
        ref_matrix = np.vstack((ref_matrix, matrix[0]))
    except ValueError:
        # to make sure the dimensions of the matrices are the same
        ref_matrix = matrix[0]
    # recursively call the function to the sub-matrix
    return row_echelon_form(matrix[1:], m-1, n, ref_matrix)

print("Row Echelon Form of the matrix is: ")
echelon_form = row_echelon_form(np.array(matrix), m, n, np.array([]))
for row in echelon_form:
    print(row)
```

تابع بعدی که سطرهای غیرصفر رو می‌شمراه:

```python
def find_rank(ref_matrix):
    rank = 0
    for row in ref_matrix:
        if not np.all(row == 0):
            rank += 1
    return rank
print("Rank of the matrix is: ", find_rank(row_echelon_form(np.array(matrix), m, n, np.array([]))))
```

این توابع تقریباً توضیح خاصی ندارن چون الگوریتم رو بالا توضیح دادم.

تست ۱:

مثال:

$$A = \begin{bmatrix} 1 & 1 & -1 & 3 \\ 2 & 2 & 6 & 8 \\ 3 & 5 & 7 & 8 \end{bmatrix} \xrightarrow{\cdots} \begin{bmatrix} 1 & 1 & -1 & 3 \\ 0 & 1 & -2 & -\frac{1}{2} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\text{rank}(A) = 2$$

```
Row Echelon Form of the matrix is:
[ 1.   1. -1.   3.]
[ 0.    1.   -2.   -0.5]
[0.  0.  0.  0.]
```

```
Rank of the matrix is:   2
```

تست ۲:

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

$$\text{rank}(A) = 3 \Rightarrow$$ مستقل خطی

```
Row Echelon Form of the matrix is:
[1.  0.5 0.5]
[0. 1. 0.]
[0. 0. 1.]
```

```
Rank of the matrix is:   3
```