RREF(Reduced Row Echelon Form)계산기

학번: 2118029

이름: 이상엽

Github address:

https://github.com/sangyeob6949/sangyub.git

1. 계산기의 목적

a. 현재 수강하고 있는 공학 수학에서 RREF 를 배우고 있는데 RREF 를 계산하면 시간도 많이 들고 계산 실수도 많이 나와 이 문제를 간단하게 해결하고 싶어 RREF 를 선택하게 되었다.

b. 기약 행사다리꼴 형태(RREF)는 행렬을 특정 형태로 변환하는 과정을 나타내며 행렬의 간소화와 선형 방정식 시스템의 해를 찾는 데 유용하다.

이는 선형 대수학에서 매우 중요한 개념이며 RREF 를 통해 행렬의 간소화, 선형 방정식의 시스템 해, 선형 변환의 기저, 선형 독립성 판단, null space 와 particular solution 등등 다양한 문제를 찾는 데 매우 유용하여, 이 문제를 해결하기 위한 사람들이 사용하면 쉽게 문제를 해결할 수 있을 것이다.

피벗 변수와 자유 변수를 통해 null space 와 particular solution 을 구할 수 있으므로 추가하였다.

2. 계산기의 네이밍의 의미

a. RREF 를 구하기 위한 계산기로 RREF 계산기라는 이름을 지어줬다.

3. 계산기 개발 계획

a. 입력 변수

```
rows, cols = map(int, input("행렬의 행과 열의 크기를 입력하세요: ").split())
```

사용자로부터 행렬의 행과 열의 크기를 차례로 입력 받는다.

EX) 3 행 2 열 이면

3 2 차례로 입력

```
print("행렬의 요소를 한 줄씩 입력하세요:")
matrix = []
for i in range(rows):
    row = list(map(float, input().split()))
    matrix.append(row)
```

사용자로부터 행과 열의 크기에 맞게 요소를 입력 받는다.

EX) 3 행 3 열 일 경우

1 2 3

4 5 6

7 8 9 와 같이 원하는 숫자 입력

b. 개발한 함수의 역할

```
def swap_rows(matrix, i, j):
    matrix[i], matrix[j] = matrix[j], matrix[i]
```

행렬에서 두 개의 행을 교환

```
def scale_row(matrix, i, scale):
    matrix[i] = [element * scale for element in matrix[i]]
```

행렬에서 특정 행의 모든 요소를 스케일링(곱) 한다.

```
def add_row(matrix, i, j, scale):
    matrix[i] = [a + b * scale for a, b in zip(matrix[i], matrix[j])]
```

행렬에서 특정 행에 다른 행을 추가

```
def rref(matrix):
    num_rows = len(matrix)
    num_cols = len(matrix[0])
```

행렬을 rref 로 변환(이때 피벗 변수와 자유 변수를 반환)

```
def print_matrix(matrix):
    for row in matrix:
        print(" | ".join(map(lambda x: f"{x:.2f}", row)))
```

주어진 행렬을 출력

```python
def main():
    # 행렬의 크기 입력 받음
    rows, cols = map(int, input("행렬의 행과 열의 크기를 입력하세요: ").split())

    # 행렬 요소 입력 받음
    print("행렬의 요소를 한 줄씩 입력하세요:")
    matrix = []
    for i in range(rows):
        row = list(map(float, input().split()))
        matrix.append(row)
    # rref 계산
    pivot_variables, _ = rref(matrix)
    # 변수 이름 설정
    variables = [f"X{i + 1}" for i in range(cols)]
    # 결과 출력
    print("Reduced Row Echelon Form (rref):")
    print_matrix(matrix)
    if not pivot_variables:
        print("이 행렬에는 피벗 변수도 자유 변수도 없습니다.")
    else:
        print("피벗 변수 및 자유 변수:")
        for i, variable in enumerate(variables):
            if i in pivot_variables:
                print(f"{variable} (피벗 변수)")
            else:
                print(f"{variable} (자유 변수)")
```

메인 함수로 행렬의 크기와 요소를 입력 받고, rref 함수를 호출하여 결과를 출력

c. 연산 과정

1. 사용자로부터 행렬의 크기를 입력 받는다.
2. 사용자로부터 행렬의 요소를 입력 받는다.
2. 입력된 행렬을 기약 행사다리꼴 형태로 변환하는 함수 rref() 호출
3. rref() 함수에서는 주어진 행렬을 기약 행사다리꼴 형태로 변환하기 위해 가우스 소거법을 사용
4. 추가로 변환된 행렬에서 피벗 변수 또는 자유 변수를 출력

d. 각 조건문 역할

```python
for r in range(num_rows):
```

각 행에 대해 가우스 소거법을 수행하기 위한 반복문

```
if lead >= num_cols:
    break
```
가우스 소거법을 수행하면서 피벗이 열의 개수를 초과할 때 처리
>>> 더 이상 피벗 열을 찾을 수 없으므로 가우스 소거법을 종료

```
while matrix[i][lead] == 0:
```
현재 행에서 피벗 열에 해당하는 요소가 0 인 경우를 처리
>>> 행렬의 행을 교환하여 0 이 아닌 요소를 찾아야 한다.

```
if i == num_rows:
```
모든 행이 이미 검사되었음에도 불구하고 피벗 열에 0 이 아닌 요소가 없는 경우를 처리
>>> 가우스 소거법을 종료하고 다음 열로 넘어간다.

```
if num_cols == lead:
    break
```
피벗을 찾을 열이 더 이상 없는 경우를 확인하여 반복문 종료

```
if lead >= num_cols:
    break
```
피벗 열이 행렬의 열 수보다 큰 경우를 확인하여 반복문을 종료

```
for i in range(num_rows):
```
현재 피벗 열 아래의 각 행에 대해 반복 각 행에서 가우스 소거를 수행해야 하므로, 행의 수에 해당하는 num_rows 만큼 반복

```
if i != r:
    add_row(matrix, i, r, -matrix[i][lead])
```
현재 반복 중인 행 i 가 피벗 행 r 이 아닌 경우에만 실행

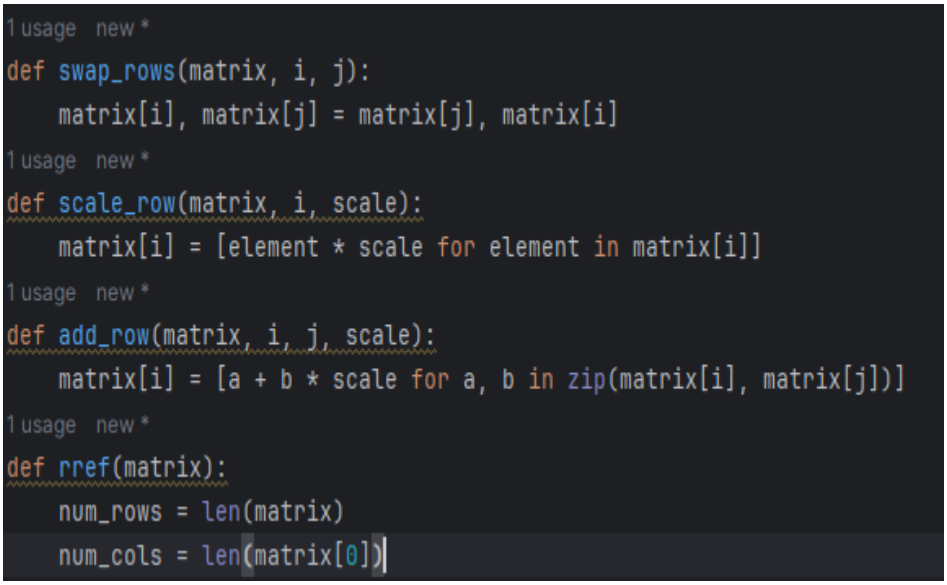
```
for i in range(num_cols):
```
모든 열에 대해 반복하며, 피벗 변수가 아닌 열을 찾는다.

```
if i not in pivot_variables:
```

피벗 변수와 자유 변수를 식별하고 처리

4. 계산기 개발 과정

a. 개발 과정 및 각 함수의 동작



```
1 usage  new *
def swap_rows(matrix, i, j):
    matrix[i], matrix[j] = matrix[j], matrix[i]
1 usage  new *
def scale_row(matrix, i, scale):
    matrix[i] = [element * scale for element in matrix[i]]
1 usage  new *
def add_row(matrix, i, j, scale):
    matrix[i] = [a + b * scale for a, b in zip(matrix[i], matrix[j])]
1 usage  new *
def rref(matrix):
    num_rows = len(matrix)
    num_cols = len(matrix[0])
```

swap_rows(matrix, i, j)

= 행렬에서 두 행의 위치를 바꾸는 역할

scale_rwo(matrix, i, scale)

= 행렬의 주어진 특정 행에 스칼라 값을 곱하는 역할

add_row(matrix, i, j, scale)

= 한 행에 다른 행의 스케일링된 값을 더하는 역할

rref(matrix)

= 행렬을 기약 행사다리꼴 형태로 변환하는 역할

```
lead = 0
pivot_variables = []
free_variables = []

for r in range(num_rows):
    if lead >= num_cols:
        break
    i = r
    while matrix[i][lead] == 0:
        i += 1
        if i == num_rows:
            i = r
            lead += 1
            if num_cols == lead:
                break
    if lead >= num_cols:
        break
```

lead = 0

= 현재 가우스 소거법을 수행할 열의 인덱스를 나타내는 변수를 초기화

pivot_variables = [ ]

= 피벗 변수의 열 인덱스를 저장할 빈 리스트를 초기화

free_variables = [ ]

= 자유 변수의 열 인덱스를 저장할 빈 리스트를 초기화

for r in range(num_rows)

= 행렬의 각 행에 대해 반복, r 은 현재 행을 나타내는 인덱스

If lead >= num_cols: break

= 현재 열 인덱스가 열의 개수를 초과하면 반복문을 종료

while matrix[i][lead] == 0

= 현재 열에서 피벗 열을 찾을 때까지 반복

if i == num_rows:

    i = r

= 모든 행을 검사한 후에도 피벗 열을 찾지 못한 경우 다시 현재 행부터 검색 시작

```
swap_rows(matrix, i, r)
scale_row(matrix, r, 1 / matrix[r][lead])

pivot_variables.append(lead)
for i in range(num_rows):
    if i != r:
        add_row(matrix, i, r, -matrix[i][lead])
```

sawp_rows(matrix, i, r)

= 피벗 열의 값을 찾았을 때, 현재 행(i)과 피벗 행(r)을 교환

scale_row(matrix, r, i/ matrix[r][lead])

= 피벗 열의 값을 1 로 만든다.

pivot_variables.append(lead)

= 피벗 열의 인덱스를 피벗 변수 리스트에 추가>>>피벗이 발견될 때마다 실행

if i !=r :

  add_row(matrix, i, x, -matrix[i][lead])

= 현재 행을 피벗 행으로 만들기 위해 각 행에 적절한 연산 수행

```
    for i in range(num_cols):
        if i not in pivot_variables:
            free_variables.append(i)

    return pivot_variables, free_variables

1 usage  new *
def print_matrix(matrix):
    for row in matrix:
        print(" | ".join(map(lambda x: f"{x:.2f}", row)))
```

for i in range(num_cols)

= 열 인덱스를 반복하면서 모든 열에 대해 검사

if i not in pivot_variables

  free_variables.append(i)

= 해당 열이 피벗 열이 아니라면 자유 변수 리스트에 현재 열의 인덱스 추가

return pivot_variables, free_variables

= 피벗 변수 리스트와 자유 변수 리스트를 반환

print_matrix(matrix)
= 각 행의 요소들을 소수점 둘째 자리까지 출력하고 요소들을 구분하기 위해 파이프(" | ") 기호를 사용

```
def main():
    # 행렬의 크기 입력 받음
    rows, cols = map(int, input("행렬의 행과 열의 크기를 입력하세요: ").split())

    # 행렬 요소 입력 받음
    print("행렬의 요소를 한 줄씩 입력하세요:")
    matrix = []
    for i in range(rows):
        row = list(map(float, input().split()))
        matrix.append(row)
    # rref 계산
    pivot_variables, _ = rref(matrix)
    # 변수 이름 설정
    variables = [f"X{i + 1}" for i in range(cols)]
```

rows, cols = map(int, input("행렬의 행과 열의 크기를 입력하세요")
= 사용자로부터 행과 열의 크기를 입력 받는다.

for i in range(rows)
= 행의 개수만큼 반복

row = list(map(float, input(), split()))
= 사용자로부터 행렬의 각 행을 입력 받고 각 요소를 실수형으로 변환하여 리스트로 만든다.

matrix.append(row)
= 변환된 행을 행렬 리스트에 추가

pivot_variables,_=rref(matrix)
= 입력된 행렬을 기약 행사다리꼴 형태로 변환하는 rref() 함수 호출 두 번째 반환 값은 사용하지 않는다.

variables= [f"X{i+1}" for i in range(cols)]
= 변수 이름을 설정

```
# 결과 출력
print("Reduced Row Echelon Form (rref):")
print_matrix(matrix)
if not pivot_variables:
    print("이 행렬에는 피벗 변수도 자유 변수도 없습니다.")
else:
    print("피벗 변수 및 자유 변수:")
    for i, variable in enumerate(variables):
        if i in pivot_variables:
            print(f"{variable} (피벗 변수)")
        else:
            print(f"{variable} (자유 변수)")
```

print("Reduced Row Echelon From (rref):")
= 기약 행사다리꼴 형태로 변환된 행렬을 출력하기 전 메시지 출력

print_matrix(matrix)
= 함수를 사용하여 기약 행사다리꼴 형태로 변환된 행렬 출력

if not pivot_variables
= 행렬이 기약 행사다리꼴로 변환되지 않는 경우를 확인

else:
= 행렬이 기약 행사다리꼴로 변환된 경우를 확인

for i, variable in enumerate(variables)
= 변수 이름 리스트를 반복하면서 각 열에 대한 변수 이름을 출력

if i in pivot_variables
= 현재 열이 피벗 변수의 열에 해당하는 경우를 확인

else:
= 현재 열이 자유 변수의 열에 해당하는 경우를 확인

b. 에러 발생 지점

1 사용자가 잘못된 행, 열 크기를 입력하는 경우

```
행렬의 행과 열의 크기를 입력하세요: 3 5 7
Traceback (most recent call last):
  File "C:\Users\82105\OneDrive\문서\desktop-tutorial\계산기.py", line 83, in <module>
  File "C:\Users\82105\OneDrive\문서\desktop-tutorial\계산기.py", line 53, in main
    # 행렬 요소 입력 받음
    ^^^^^^^^^
ValueError: too many values to unpack (expected 2)
```

2 사용자가 행렬의 요소를 입력할 때 숫자가 아닌 문자를 입력한 경우

```
행렬의 행과 열의 크기를 입력하세요: 3 3
행렬의 요소를 한 줄씩 입력하세요:
1 2 3
a 5 6
Traceback (most recent call last):
  File "C:\Users\82105\OneDrive\문서\desktop-tutorial\계산기.py", line 78, in <module>
    main()
  File "C:\Users\82105\OneDrive\문서\desktop-tutorial\계산기.py", line 57, in main
```

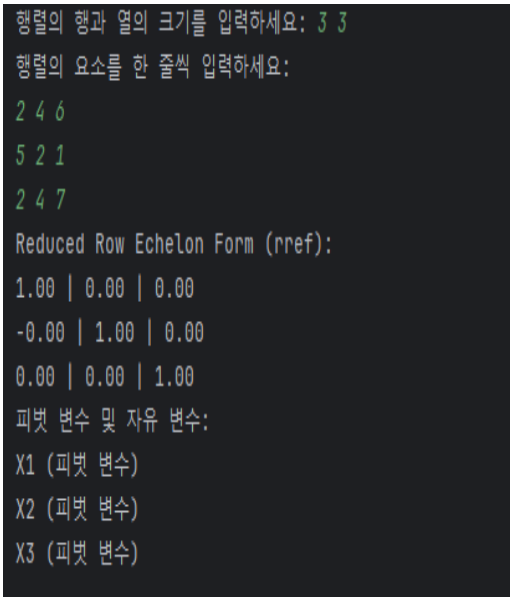
c. 에러 발생에 대한 해결책과 적용시 변화

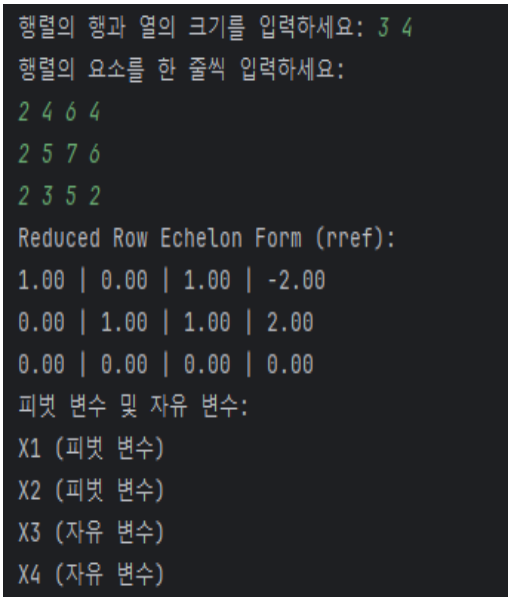
```
행렬의 행과 열의 크기를 입력하세요: 3 3
행렬의 요소를 한 줄씩 입력하세요:
1 2 3
4 5 6
7 8 9
```

```
행렬의 행과 열의 크기를 입력하세요: 4 3
행렬의 요소를 한 줄씩 입력하세요:
3 1 2
1 2 6
2 1 7
5 3 2
```

에러가 발생하지 않게 하려면 행과 열의 크기를 알맞게 입력하고 크기에 따른 각각의 요소에는 문자를 포함하지 않고 숫자만 입력한다.

d. 동작 결과 캡쳐





이처럼 올바른 값을 입력하면 RREF 수행 시 나타나는 행렬과 추가로 피벗 변수와 자유 변수를 확인할 수 있다.
(1 열은 X1, 2 열은 X2, n 열 Xn 으로 표현된다.)

5. 계산기 개발 후기

계산기를 개발하기 전 가장 큰 난관이 어떤 계산기를 만들 것인가였다. 계속 해서 고민하던 중 수강하는 과목 중에서 계산기를 만들자 해서 공학 수학의 행렬 부분을 택하였다. 그 중 지금 배우고 있는 RREF 가 떠올랐고 RREF 계산기를 인터넷에 쳐보니 있었지만 그래도 RREF 가 하고 싶어 이것을 택하였다.
계산기를 만드는 과정은 거의 GPT 가 해주었지만, 계산기 개발 계획과 계산기 개발 과정을 작성하면서 하나하나 코드를 뜯어 그 코드가 무엇을 하는 코드인지 알게 되었고 어느 정도 이 코드가 어떤 역할이고 그 코드들이 모여 전체적으로 어떻게 돌아가는지 알게 되었다. 프로그래밍 수업을 듣기 전에는 코딩에는 관심도 없었고 할 생각도 없었지만, 이 수업을 듣고 계산기를 만들고 나니 코딩이나 GPT 같은 것들을 잘 다룬다면 앞으로의 인생에 있어 이로운 점이 많을 것 같다는 생각이 들었다. 무섭게 발전하고 있는 시대에 맞춰 나도 발전해야겠다고 다짐했다.