**선형대수학 및 프로그래밍**

**과제 4-5 보고서**

소프트웨어학과 20003322 박세훈

**기획 및 구상**

1. 파이썬 라이브러리 중 numpy를 이용할 계획이다. 그 이유는 파이썬에서도 배열을 이용하여 행렬처럼 활용할 수 있지만 numpy를 이용한다면 이차원 배열을 아예 행렬로 변환하여 연산이나 특성, 또는 행렬에 관련된 여러 함수를 이용할 수 있기 때문이다.
2. 파일 입출력에 관련하여 os모듈을 활용하여 현재 위치에서 용이하게 입출력인 진행되게 한다
3. Input.txt 파일에서 행렬의 정보를 입력 받아 numpy를 이용하여 이차원배열에서 행렬로 변환한다.
4. 그후 해당 행렬을 함수에 넣어 내부에서 두가지 방법으로 (1) cofactor expansion 2) properties of determinant) 행렬 값을 계산해 과정을 ouput.txt에 저장한다.

**코드 구현**

1. 입출력 관련 부분
2. Input.txt와 파이썬 코드 파일을 하나의 디렉토리에 저장하기 때문에 절대경로로 지정하여 작업을 진행한다.
3. Input.txt에서 행렬의 정보를 입력 받아 먼저 이차원 배열에 저장한다
4. 그후 numpy로 계산에 용이하도록 행렬화한다(np.array(“리스트 이름”)이용)
5. 작업 도중 발생하는 모든 계산과정을 현재 파일 경로에 있는 output.txt(없으면 생성)을 쓰기모드로 불러와 작성한다.
6. 행렬식을 계산하는 함수 부분
7. calculate\_determinant  
   - cofactor\_exapansion함수와 determinant\_with\_steps함수를 이용하여 두가지 방식으로 풀이를 진행한다. 그 과정에서 생기는 계산과정은 포맷에 맞게 ouput.txt에 작성한다.  
   - calculate\_determinant함수의 파라미터는 행렬이며, 실행 시 ouput.txt를 쓰기모드로 연다.
8. cofactor\_exapansion  
   - matrix, row, depth, all\_steps, step\_value, detailed\_steps를 받아 각각 행렬, cofactor expansion을 진행할 행, 몇 번째 재귀인지, 계산과정을 기록하기 위한 구조이다.  
   - 해당 함수의 핵심 로직은 기본적으로 2x2의 행렬까지 재귀적으로 계산을 진행하고, 2x2행렬이 되면 numpy 제공 함수인 np.linalg.det을 이용하여 행렬식 값을 계산하고, 이는 종료조건이다. nxn 행렬의 경우 첫번째 헹(row를 기준으로 cofactor\_exapansion를 진행한다. 또한 이과정에서 생기는 과정을all\_steps에는 각 재귀 단계에서 cofactor expansion 과정 기록, step\_values에서 최상위 전개 단계에서 계산된 항(term) 저장, detailed\_steps에서 2x2 행렬 계산의 상세 과정을 기록한다.  
   - 세부과정에 대해서는 np.delete를 이용하여 현재 축에 대한 row와col을 제거하여 minor matrix를 만들고, cofactor를 row와 col에 따라 부호를 할당하여 현재 축에 맞는 값에 곱해 만든다. 이러한 과정을 2x2행렬이 나올 때까지 반복하면 추후에 있을 출력을 위해 값을 각 데이터 구조에 포맷에 맞게 저장한다.  
   - 함수의 반환값은 계산된 행렬식 값인 cofactor\_sum, 전개 과정을 깊이별로 기록한 딕셔너리인 all\_steps, 최상위 단계의 항(term) 값들인 step\_values, 2x2 행렬의 상세 계산 기록인 detailed\_steps이다.  
   - 출력부(output.txt에 작성하는 부분)에서는 먼저 깊이별 출력을 진행한다. all\_steps에서 깊이를 먼저 출력 후 그에 맞는 행렬식 전개를 작성한다. 그 후 2x2 행렬은 바로 계산이 되기 때문에 해당 계산과정을 행렬과 함께 출력한다. 마지막에는 다 더하는 과정을 출력하여 보여준다.
9. determinant\_with\_steps  
   - 가우스 소거법을 중심으로 행렬식을 계산하는 방식의 함수이다. 일반적으로 수기로 계산을 진행할 때는 계산의 편의성을 위해 행과 행을 바꾸는 작업을 진행(첫번째 행이 2이고, 두번째 행이1인 경우)하지만 컴퓨터로 계산할 때는 불필요하다고 판단하여 제외했다.  
   - 핵심로직의 경우 각 단계를 저장할 리스트하나, 행렬식을 저장할 변수, 행렬의 크기를 선언 후 작업에 이용한다. 이후 과정은 피봇팅 -> 가우스 소거 -> 대각선 용소 곱해 값 구하기 방식으로 진행된다  
   - 피봇팅의 경우 현재 피봇 요소가 0이라면 해당 행과 대각 부분이 0이 아닌 행을 교환하고, 그 내용을 all\_steps에 저장, 행렬식 값에 -1곱한다(행렬식의 특성) 그 과정에서 만약 모든 행이 0이라면 행렬식이 0이라고 판단한 후 반환한다.  
   - 가우스 소거 같은 경우는 해당 행에 대해 적당한 스칼라값(factor로 나누기로 구함)을 곱해 하위 행에 뺀다. 이 과정을 또한 all\_step에 저장한다. 이러한 과정으로 상대각행렬 형태로 만든다.  
   - 그 후 대각 원소들을 다 곱하여 행렬식 값을 구한다.

**결론 및 참조 내용**

1. 과제 1처럼 여러 행렬을 계산할 수 있게 만들었고, 과제2, 3처럼 사람이 손으로 쓴 것처럼은 아니지만, 계산 순서와 포맷에 맞게 출력 후 output.txt에 저장하게 했다.
2. 중간 확인용으로 디버깅 출력문을 추가하여 안정성을 부여했다.
3. 이 과제를 수행함에 있어서 생성형AI(chat GPT)에서 참조한 내용은 다음과 같다  
   - 정해진 포맷에 맞게 저장된 input.txt에서 행렬 정보를 파싱하는 부분과 내가 원했던 포맷에 맞게 입력하는 부분  
   - numpy에 대한 기본적인 설명과 간단한 예제를 통한 사용법
4. 이 과제를 수행함에 있어서 배운 점은 다음과 같다  
   - 사실 numpy가 필요한가 싶었다. 그냥 이차원배열로 구현해도 큰 문제가 안될 것 같았는데 구현 난이도나 복잡도가 줄었다.  
   - 선형대수학의 모든 수학적 해결방법을 구현할 수 있는 라이브러리에 대해 알 수 있어 다음에 이런 방식의 문제가 생기면 이를 쉽게 이용할 수 있을 것 같다.