提出日：2024/5/15

プログラミング演習　第５回演習レポート

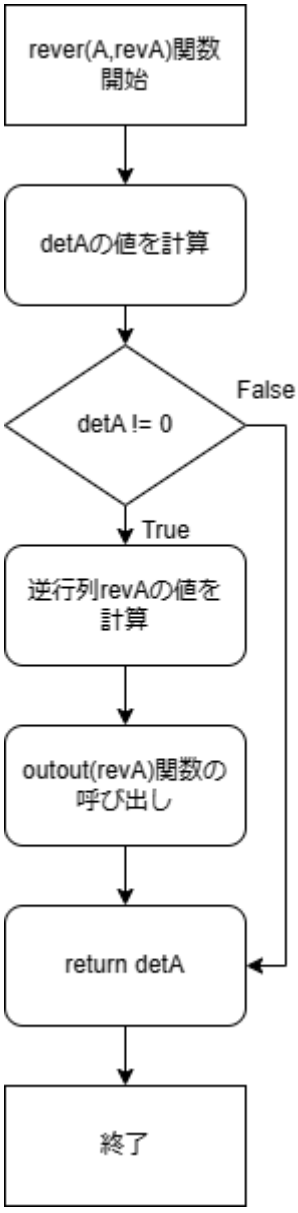
担当教員：杉本　千佳先生
所属：理工学部　数物・電子情報系学科
電子情報システムEP
学年・クラス：２年　Fe1
学籍番号：2364092
氏名：熊田　真歩

（1）課題番号：基本課題3

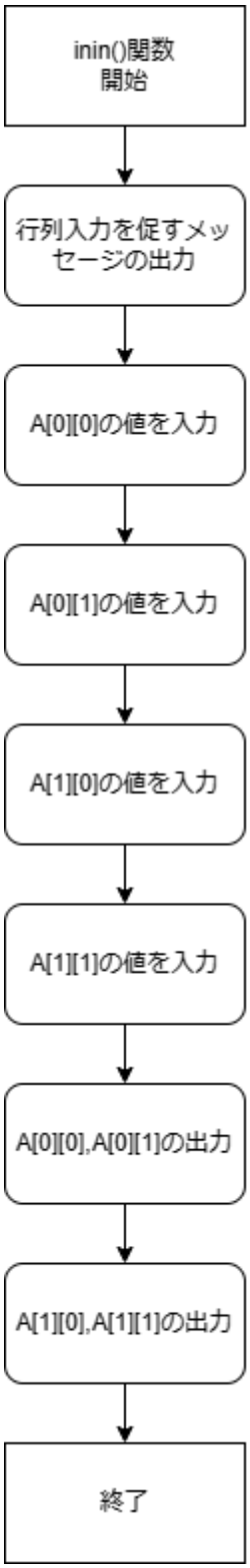
課題名：逆行列を求める関数の作成

（2）プログラムのフローチャート（関数ごと）

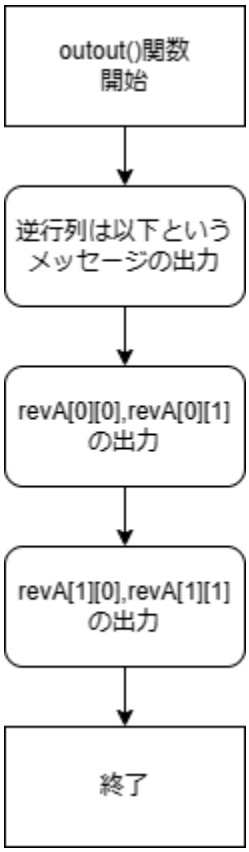
・逆行列を計算する rever(A,revA)関数



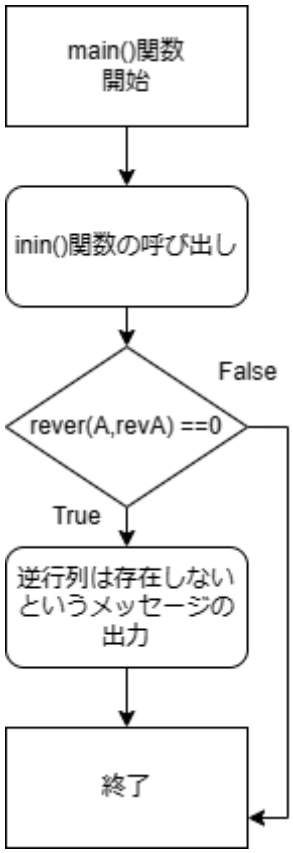
・行列の入力を行う inin()関数



・行列の出力を行う outout()関数



・行列式が 0 であるかを判定し逆行列存在の有無を判定する main 関数



（3）アルゴリズム「正しいこと」である説明

本プログラムでは、どのような要素を持つ２×２行列であってもその逆行列を出力するプログラムを作成した。ここで行列式の値が 0 となってしまう場合には逆行列の定義式からも明らかな用に、分母に 0 が来てしまうため逆行列を持たない。このつまり、本プログラムでは逆行列は存在しないというメッセージが出力される。以下に、行列式の値が正の時、負の時、零の時、更には各要素が少数値である時の出力結果を示す。

（ｉ）行列式の値が正の時

```
>>> main()
元の行列を入力してください
A[0][0]:9
A[0][1]:3
A[1][0]:4
A[1][1]:8
9.0 3.0
4.0 8.0
逆行列は以下の通り
0.13333333333333333 -0.05
-0.06666666666666667 0.15
>>>
```

（ⅱ）行列式の値が負の時

```
>>> main()
元の行列を入力してください
A[0][0]:1
A[0][1]:10
A[1][0]:7
A[1][1]:3
1.0 10.0
7.0 3.0
逆行列は以下の通り
-0.04477611940298507 0.14925373134328357
0.1044776119402985 -0.014925373134328358
>>>
```

（ⅲ）行列式の値が 0 の時

```
>>> main()
元の行列を入力してください
A[0][0]:5
A[0][1]:1
A[1][0]:5
A[1][1]:1
5.0 1.0
5.0 1.0
逆行列は存在しない
>>>
```

（ⅳ）各要素が少数値である時

```
>>> main()
元の行列を入力してください
A[0][0]:3.98
A[0][1]:3.14
A[1][0]:2.73
A[1][1]:1.732
3.98 3.14
2.73 1.732
逆行列は以下の通り
-1.0316647208787013 1.8703390436253593
1.6261227990755514 -2.3706845202639903
>>>
```

このように、いかなる２×２行列においても要素に数字を入れる限り、正しく逆行列を出力するという観点から本プログラムにおけるアルゴリズムは正しいと言える。

（４）ソース・プログラムの説明

・逆行列を計算する rever(A,revA)関数について

```
def rever(A,revA):                          ##関数の定義
  detA = float(A[0][0])*float(A[1][1])-float(A[0][1])*float(A[1][0])   ##行列式の計算
  if detA != 0:                             ##行列式の値が 0 でない時（逆行列が存在する時）
    revA=[[(1/detA)*A[1][1],-(1/detA)*A[0][1]],[-
                                            (1/detA)*A[1][0],(1/detA)*A[0][0]]]
                                            ##逆行列を計算
    outout(revA)                            ##上の式によって計算した逆行列を
                                            outout(revA)関数により出力
  return detA                               ##戻り値の値を行列式の値とする。
```

・行列に値を入力する inin()関数について

```
def inin():                                 ##関数 inin()の定義
 print("元の行列を入力してください")          ##行列の入力を促すメッセージの出力
 A[0][0] = float(input("A[0][0]:"))         ##１行１列の値の入力
 A[0][1] = float(input("A[0][1]:"))         ##1 行２列の値の入力
 A[1][0] = float(input("A[1][0]:"))         ##2 行１列の値の入力
 A[1][1] = float(input("A[1][1]:"))         ##2 行２列の値の入力
 print(A[0][0],A[0][1])                     ##1 行目の行列の値を出力
 print(A[1][0],A[1][1])                     ##２行目の行列の値を出力
```

・逆行列の値を出力する outout()関数について

```
def outout(revA):                    ##関数 outout()の定義
  print("逆行列は以下の通り")          ##逆行列は以下の通りというメッセージの出力
  print(revA[0][0],revA[0][1])       ##1 行目の逆行列を出力
  print(revA[1][0],revA[1][1])       ##2 行目の逆行列を出力
```

・行列式が 0 か否か判定し逆行列の出力を行う main（）関数について

```
def main():                          ##main（）関数の定義
  inin()                             ##行列の入力を行う inin()関数の呼び出し
  if   rever(A,revA) == 0:           ## rever(A,revA)の値を呼び出し、行列式の値が 0 の時
    print("逆行列は存在しない")          ##逆行列は存在しないというメッセージの出力
```

（５）考察

本プログラムではあらゆる２×２行列に対して逆行列の出力を行うというプログラムを作成した。今回は、逆行列の計算を行う関数、行列の入力を行う関数、逆行列の出力を行う関数、そして行列式が 0 であるかを判定し逆行列存在の有無を判定する関数と、4 つの関数を用いたプログラムとなった。関数はプログラムを書く上で非常に重要となるが今回のような場合においても複数の関数を適宜利用することによって人間が読んでも見やすく、理解しやすいプログラムとなった。その観点から、4 つの関数を採用したアルゴリズムを採用したことは良い選択であったと言える。

また、ここで、数はどのような四則演算においても式は同じであることが知られているが Python 等のプログラミング言語によるプログラムにおいて、計算する数が整数であるか少数であるかという事が非常に重要となる。これは Python の変数型の特性のためである。int 型である整数値と float 型である少数値を同じ式において混在させることはできない。すなわち式においてはすべての型を統一させる必要がある。今回のように計算式に割り算がある時は int 型のみの計算であってもその結果が float 型となることは大いに考えられるため、すべての値を float 型としてから行列式の計算を行った。このように、割り算を含む計算は float 型となっても対応できるよう、int 型変数を float 型とすることが望ましい。この選択は本アルゴリズムを正しいものとするために不可欠である。

今回は２×２行列を表現するために二次元のリストを採用した。行列を表現するうえで n 次正方行列に対応するには n 次元のリストを使うと簡単にその行列の計算が可能となることが本プログラムから理解できる。

（６）参考文献

・２次元リスト作成のために参照したサイト

GeekBlocks　「【Python】２次元リストの使い方(初期化/追加/参照など)を詳しく解説」

https://af-e.net/python-two-dimensional-list/　　2024/05/15 アクセス

・フローチャートを作成したサイト

draw.io　「Security-first diagramming for teams.」

https://www.drawio.com/　　2024/05/15 アクセス

（７）感想

　今回は２次元リスト以外、よく精通しているコードのみを用いてコードを書くことができた。そういった観点からは比較的短時間でプログラムを書くことができたが、加点対象である出力を関数とするということに少し時間を要した。関数を分けるとその分見やすく理解しやすいプログラムになる一方で、コードを書く際には関数の引数や戻り値をしっかり意識する必要があり、時間がかかってしまう。無論、これは私の勉強、実践不足であろう。更に多くの経験を積んでどんなコードをも短時間で正しくかける人材を目指して奮闘していきたい。