

Python 实验报告

姓名	郭凯杰
学号	18342022
实验名称	用 Python 做数学题

目录

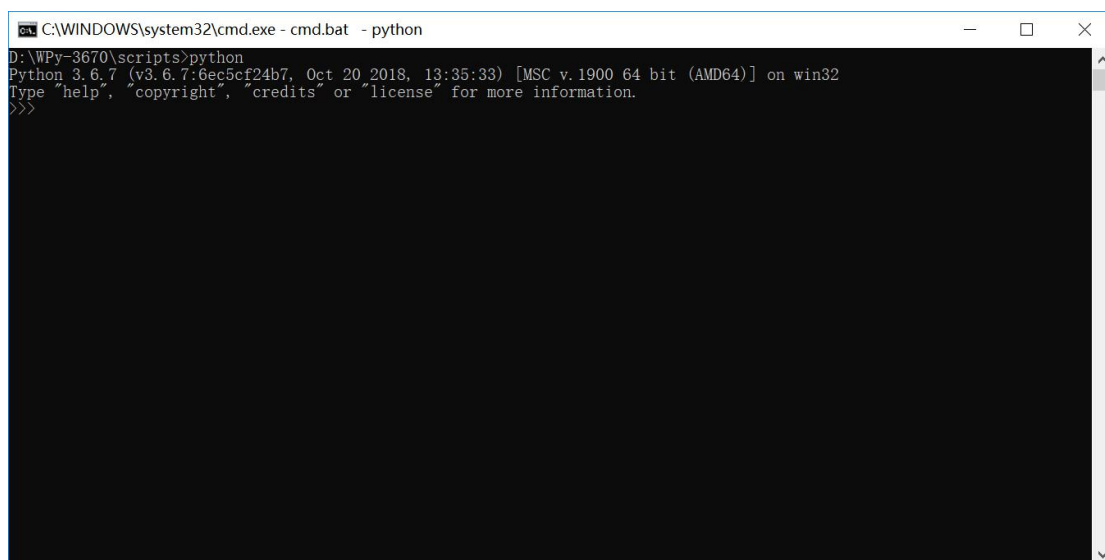
一、实验目的.....	1
二、实验环境.....	1
三、 使用 Python 做高数题目.....	2
1. 高等数学.....	2
1) 写出 e^x 的在 $x=0$ 处的带有佩亚诺余项的9阶麦克劳林公式:	2
2) 计算积分 $\int \sin^2(x) \cos^5(x) dx$:	2
3) 验证 2) 结果是否正确:	3
2. 线性代数:	4
1) 解方程 $Ax=b$:	4
2) 若存在, 求 1) 中矩阵 A 的逆:	5
四、实验总结.....	6

一、实验目的

1. 了解一种“解释型”语言 python，以及何为交互式编程
2. 使用 python 做一些简单的科学计算

二、实验环境

1. 编程工具：Python（winpython）
2. 操作系统：Windows



```
C:\WINDOWS\system32\cmd.exe - cmd.bat - python
D:\WPY-3670\scripts>python
Python 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20 2018, 13:35:33) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

三、使用 Python 做高数题目

1. 高等数学

1) 写出 e^x 的在 $x = 0$ 处的带有佩亚诺余项的 9 阶麦克劳林公式：

指令分析：

1. `from sympy import *` :

SymPy 是 *Python* 的数学符号计算库，该指令用于从 *sympy* 库导入所有内容

2. `x = Symbol("x", real = True)` :

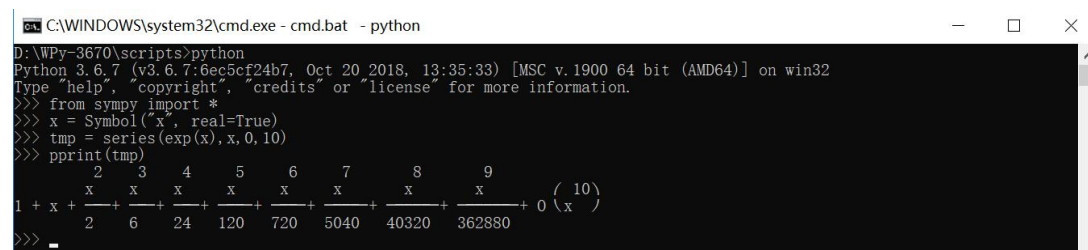
Symbol 定义符号 x ，并且指定其为实数（而不是复数）。

3. `tmp = series(exp(x), x, 0, 10)` :

series 是泰勒展开函数，其中 *exp(x)* 代表 e^x 。

4. `pprint(tmp)` :

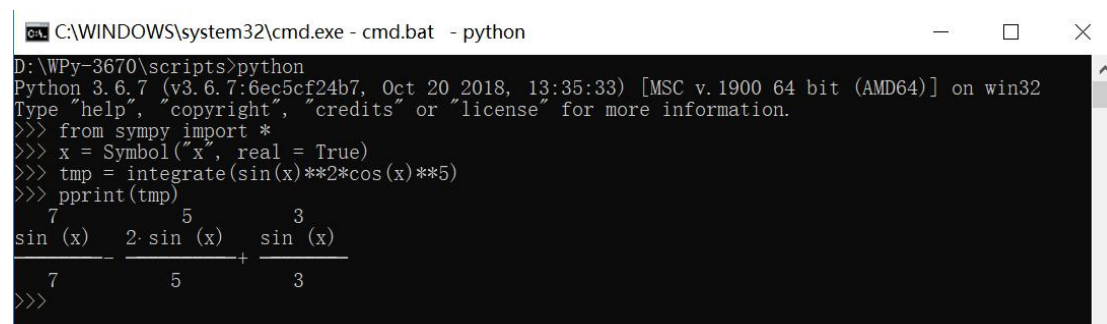
pprin 将公式用更好看的格式打印出来



```
C:\WINDOWS\system32\cmd.exe - cmd.bat - python
D:\WPY-3670\scripts>python
Python 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20, 2018, 13:35:33) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from sympy import *
>>> x = Symbol("x", real=True)
>>> tmp = series(exp(x), x, 0, 10)
>>> pprint(tmp)
      2      3      4      5      6      7      8      9
1 + x + - x + - x + - x + - x + - x + - x + - x + O ( x )
      2      6     24     120    720    5040   40320  362880
>>>
```

2) 计算积分 $\int \sin^2(x) \cos^5(x) dx$:

指令分析：*integrate* 为计算不定积分，其余同上。



```
C:\WINDOWS\system32\cmd.exe - cmd.bat - python
D:\WPY-3670\scripts>python
Python 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20, 2018, 13:35:33) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from sympy import *
>>> x = Symbol("x", real = True)
>>> tmp = integrate(sin(x)**2*cos(x)**5)
>>> pprint(tmp)
      7      5      3
sin (x) - - sin (x) + sin (x)
      7      5      3
>>>
```

3) 验证 2) 结果是否正确:

指令分析:

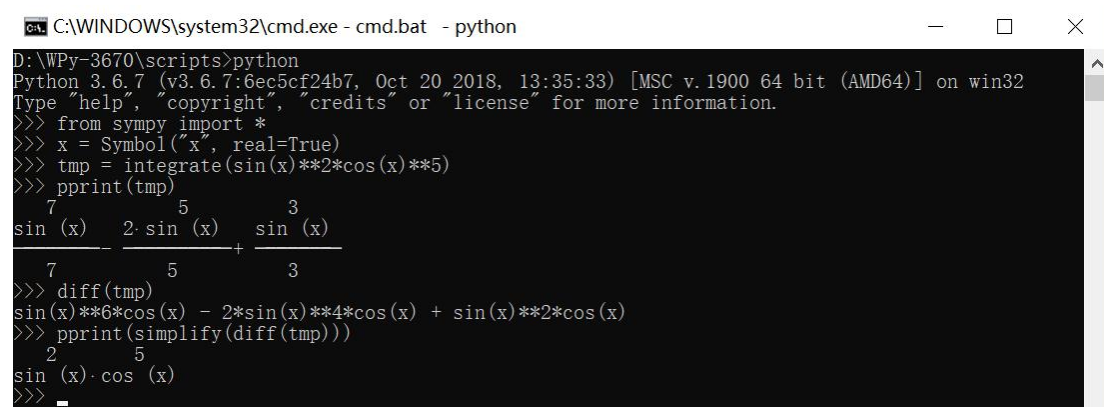
1. `diff(tmp)`:

diff 为求导。

但是结果稍微有点复杂，化简一下 *Diff(tmp)*。

2. `pprint(simplify(diff(tmp)))`:

simplify 为化简公式作用，此指令即为打印出 *tmp* 导数的简化形式。



```
C:\WINDOWS\system32\cmd.exe - cmd.bat - python
D:\WPY-3670\scripts>python
Python 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20 2018, 13:35:33) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from sympy import *
>>> x = Symbol("x", real=True)
>>> tmp = integrate(sin(x)**2*cos(x)**5)
>>> pprint(tmp)
      7      5      3
sin (x) - 2 sin (x) + sin (x)
-----
      7      5      3
>>> diff(tmp)
sin(x)**6*cos(x) - 2*sin(x)**4*cos(x) + sin(x)**2*cos(x)
>>> pprint(simplify(diff(tmp)))
      2      5
sin (x) cos (x)
>>> -
```

2. 线性代数：

1) 解方程 $Ax = b$:

$$\text{其中 } A = \begin{bmatrix} 1 & -2 & 1 \\ 0 & 2 & -8 \\ -4 & 5 & 9 \end{bmatrix} \quad , \quad b = \begin{bmatrix} 0 \\ 8 \\ -9 \end{bmatrix} .$$

指令分析：

1. `import numpy as np:`

导入 *Numpy* 模块，在下文中均采用 *np* 代替 *Numpy*

2. `A = np.mat ([[1,-2,1],[0,2,-8],[-4,5,9]]) :`

np.mat 创建矩阵 *A* , 同理, `b = np.mat ([[0],[8],[-9]])` 创建矩阵 *b*。

3. `x = np.linalg.solve(A, b) :`

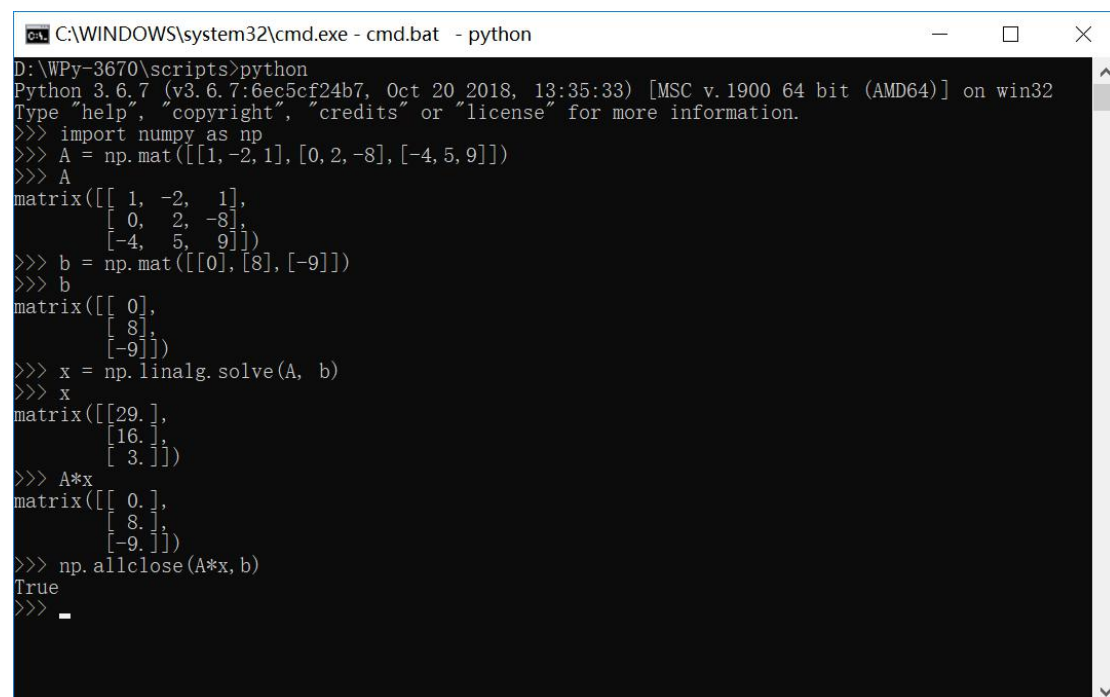
np.linalg.solve 是解线性方程组。

4. `A*x :`

*A*x* 计算矩阵乘积，此处用于检验 *x* 是否为方程的解。

5. `np.allclose(A*x, b) :`

np.allclose 用于检验两个矩阵是否相同，此处也用于检验 *x* 是否为方程的解。



```
C:\WINDOWS\system32\cmd.exe - cmd.bat - python
D:\WPY-3670\scripts>python
Python 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20 2018, 13:35:33) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import numpy as np
>>> A = np.mat([[1,-2,1],[0,2,-8],[-4,5,9]])
>>> A
matrix([[ 1, -2,  1],
        [ 0,  2, -8],
        [-4,  5,  9]])
>>> b = np.mat([[0],[8],[-9]])
>>> b
matrix([[ 0],
        [ 8],
        [-9]])
>>> x = np.linalg.solve(A, b)
>>> x
matrix([[29.],
        [16.],
        [ 3.]])
>>> A*x
matrix([[ 0.],
        [ 8.],
        [-9.]])
>>> np.allclose(A*x, b)
True
>>> _
```

2) 若存在, 求 1) 中矩阵 A 的逆:

指令分析:

1. `b = np.mat([[0],[0],[0]])`:

`np.mat` 创建矩阵 `b` , 使其为零向量。

2. `x = np.linalg.solve(A, b)` :

`np.linalg.solve` 是解线性方程组, 此处所得解为只有平凡解, 故根据 IMT 可知, 矩阵 A 可逆。

3. `A_inverse = np.linalg.inv(A)` :

`np.linalg.inv(A)` 用于求矩阵 A 的逆, 并赋值给 `A_inverse`。

```
C:\WINDOWS\system32\cmd.exe - cmd.bat - python
>>> A
matrix([[ 1, -2,  1],
        [ 0,  2, -8],
        [-4,  5,  9]])
>>> b
matrix([[0],
        [0],
        [0]])
>>> x = np.linalg.solve(A, b)
>>> x
matrix([[ -0. ],
        [  0. ],
        [  0. ]])
>>> A_inverse = np.linalg.inv(A)
>>> A_inverse
matrix([[29. , 11.5,  7. ],
        [16. ,  6.5,  4. ],
        [ 4. ,  1.5,  1. ]])
>>> B = np.linalg.inv(A_inverse)
>>> B
matrix([[ 1.00000000e+00, -2.00000000e+00,  1.00000000e+00],
        [ 2.46716228e-16,  2.00000000e+00, -8.00000000e+00],
        [-4.00000000e+00,  5.00000000e+00,  9.00000000e+00]])
>>> _
```

注意:

在此处由于精度误差, 不能够验证 $(A^{-1})^{-1} = A$, 有上述可以看出, A 与 B 仍有“差距”,

其中 $B = (A^{-1})^{-1}$ 。

实验总结

通过这一次实验，我了解到一门“解释型”语言 Python 与平时我常用的“编译型”语言 C 之间的不同，也被 Python 的交互式以及科学计算函数库给深深吸引住，比方说 Sympy。我认为，交互式编程就是：面对编写程序过程中的任何变化，程序直接产生反馈，让程序猿能够看到结果，或者说程序猿得到自己的创造的东西的实时的反馈。Python 便有这种特点，相较于 C 的编辑，保存，编译，运行和调试这种比较繁琐的方式，Python 更适合数据开发，上述实验就是一个很好的小例证。