- `numpy` 解线性方程组
- `sympy` 符号运算
- `scipy` 求解方程数值解

例 1：

$$\begin{cases} 10x - y - 2z = 72 \\ -x + 10y - 2z = 83 \\ -x - y + 5z = 42 \end{cases}$$

拆解为矩阵形式：

$$A = \begin{pmatrix} 10 & -1 & -2 \\ -1 & 10 & -2 \\ -1 & -1 & 5 \end{pmatrix}, \quad b = \begin{pmatrix} 72 \\ 83 \\ 42 \end{pmatrix}$$

$$Ax = b$$

导入合适的工具包：

`numpy` 解线性方程组

```
from sympy import *
from scipy.integrate import odeint, solve_ivp
import numpy as np
import matplotlib.pyplot as plt
```

使用 `numpy` 求线性方程组的数值解

```
a = np.array([[10, -1, -2], [-1, 10, -2], [-1, -1, 5]])
b = np.array([[72], [83], [42]])
c = np.linalg.solve(a, b)  # lin: linear  alg: algebra
# 线性代数
print(c)
```

```
[[11.]
 [12.]
 [13.]]
```

```python
a = np.array([[10, -1, -2], [-1, 10, -2], [-1, -1, 5]])
b = np.array([[72], [83], [42]])
c = np.linalg.inv(a) @ b  # inv: inverse
# 逆矩阵
print(c)
```

上图就是：

$$x = A^{-1}b$$

sympy 符号运算

符号解以代数符号与运算的形式完整地写出解的代数式，强调 `解的代数性`

数值解则是在给定方程的一些初始条件下不按式子，而是要 `算出一个数值` ，这个数值可以不需要完完本本的精确，满足一定的要求就可以了

举例：与圆周率有关，那解析解也就是**符号解**必须表示为 $x = 0.32\pi$（或者 $\sqrt{5}$ 等等）

如果仅为求数值，可以在圆周率取 3.14 的精度限制下认为 $x = 1.0048$

- 创造符号对象
- `sympy` 的算术运算
- `sympy` 的微积分运算
- `sympy` 的方程与方程组求解

```python
init_session()  # 初始化 sympy 的方法
x, y, z, t = symbols("xyzt")  # 默认为符号的 symbols
a, b, c, d = symbols("abcd", real=True)  # real=True 表示实数域
k, m, n = symbols("kmn", integer=True)   # integer=True 表示积分
f, g, h = symbols("fgh", cls=function)   # cls=Function 表示这是函数
init_printing()
```

```
These commands were executed:
>>> from sympy import *
>>> x, y, z, t = symbols('x y z t')
>>> k, m, n = symbols('k m n', integer=True)
>>> f, g, h = symbols('f g h', cls=Function)
>>> init_printing()
```

唤醒算法后，在终端操作：

```
>>> 1 + 3
4
>>> sqrt(2)
√2
>>> float(sqrt(2))
1.4142135623730951
```

```
>>> radius = 10
>>> height = 100
>>> area = pi * radius**2
>>> volume = area * height
>>> volume
10000·π
```

积分有：

```
>>> integrate(sin(x),x)
-cos(x)
```

```
>>> integrate(1 / (1 + x**5), x)
log(x + 1)          ⎛     4       3       2                          ⎞
────────── + RootSum ⎝625·t  + 125·t  + 25·t  + 5·t + 1, t ↦ t·log(5·t + x)⎠
    5
```

```
>>> integrate(sin(x),(x,0,pi))
2
```

求定积分就将变量变成元组

脚本式编程：

```python
x = sp.symbols("x")
f = x**2 + 2 * x + 1
integral_f = sp.integrate(f, x)
print(sp.latex(integral_f)) # 以 latex 格式输出
```

有：

```python
x, y = sp.symbols("x y")
f = 1 / (x**2 + y)
integral_f = sp.integrate(f, x)
print(sp.latex(integral_f))
```

$$-\frac{\sqrt{-\frac{1}{y}}\log\left(x-y\sqrt{-\frac{1}{y}}\right)}{2}+\frac{\sqrt{-\frac{1}{y}}\log\left(x+y\sqrt{-\frac{1}{y}}\right)}{2}$$

指定一个数是正数：

```python
x = sp.symbols("x")
a = sp.symbols("a", positive=True) # 指定是正数
print(sp.latex(sp.integrate(1 / (x**2 + a), x)))
```

$$\frac{\operatorname{atan}\left(\frac{x}{\sqrt{a}}\right)}{\sqrt{a}}$$

解方程组：

```python
import sympy as sp

x, a, b, c = sp.symbols("x a b c")
solution = sp.solveset(a * x**2 + b * x + c, x)
solution = sp.latex(solution)
print(solution)
```

$$\left[-\frac{b}{2a} - \frac{\sqrt{-4ac+b^2}}{2a}, \ -\frac{b}{2a} + \frac{\sqrt{-4ac+b^2}}{2a}\right]$$

`solveset` 解方程的时候要把方程右侧化为 0

或者：

```python
import sympy as sp

x, a, b, c = sp.symbols("x a b c")
solution = sp.solveset(sp.Eq(a * x**2 + b * x, -c), x)
solution = sp.latex(solution)
print(solution)
```

使用 `Eq` 方法创建一个等式。

简单的，可以用 `solve` 替代（效果类似）：

```python
x, y = sp.symbols("x y")
print(sp.latex(sp.solve(sp.Eq(x * 2, 2), x)))
print(sp.latex(sp.solve([x + y - 35, x * 2 + y * 4 - 94], x, y)))
print(sp.latex(sp.solve(x**2 + x - 20, x)))
```

$$[1]$$
$$\{x : 23,\ y : 12\}$$
$$[-5,\ 4]$$

解方程组时，常常使用列表的形式，如上述代码第三行。

有 `nonlinsolve` 来求解非线性方程组：

```
a, b, c, d = sp.symbols("a b c d", real=True)
solution = sp.nonlinsolve([a**2 + a + b, a - b], [a, b])
print(sp.latex(solution))
```

$$\{(-2,\ -2), (0,\ 0)\}$$

但它并不是总是正确：

```
a, b, c, d = sp.symbols("a b c d", real=True)
solution = sp.nonlinsolve([sp.sin(a) - a], [a])
print(sp.latex(solution))
```

$$\{(\{a \mid a \in \mathbb{C} \wedge -a + \sin\left(a\right) = 0\},)\}$$

此为错解。

进行微分运算：

```
f = sp.symbols("f", cls=sp.Function)
x = sp.symbols("x")
f = lambda x: sp.sin(x) / x
print(sp.latex(f(x).diff(x)))
```

$$\frac{\cos\left(x\right)}{x} - \frac{\sin\left(x\right)}{x^{2}}$$

```
f, g = sp.symbols("f g", cls=sp.Function)
x = sp.symbols("x")
g = lambda x: sp.sin(x)
f = lambda t: t / x
print(sp.latex(f(g(x)).diff(x)))
```

scipy数值运算

- `scipy.optimize.fsolve`
- `scipy.optimize.root`
- 方程组以函数形式传入，给出数值解

$$
\begin{cases}
\cos\beta = \dfrac{L_2^2 + L_1^2 - L_3^2}{2L_2L_3} \\
p_2^2 = (x + L_3\cos\theta - x_1)^2 + (y + L_3\sin\theta)^2 \\
p_3^2 = (x + L_2\cos(\beta+\theta) - x_2)^2 + (y + L_2\sin(\beta+\theta) - y_2)^2 \\
p_1^2 = x^2 + y^2
\end{cases}
$$

上述方程是超越方程组，无法用 `sympy` 符号求解，使用`scipy`求数值解。

```python
from scipy.optimize import fsolve
from math import sin, cos, pi


def f(ax):
    x, y, theta = ax[0], ax[1], ax[2]
    return [
        (x + 3 * cos(theta) - 5) ** 2 + (y + 3 * sin(theta)) ** 2 - 25,
        x**2 + y**2 - 25,
        (x + 3 * cos(pi / 3 + theta)) ** 2 + (y + 3 * sin(pi / 3 + theta) - 6) ** 2 - 9,
    ]


# 以函数形式写入，还要给一个初始的猜测值
result = fsolve(f, [-1.37, 4.80, 0.12])
print(result)
```

```
[1.15769945 4.86412705 0.02143414]
PS C:\Code\CodePractice\Homework\Python>
```

与 `fslove` 类似的有 `root`：

```python
from scipy.optimize import fsolve, root
from math import sin, cos, pi


def f(ax):
    x, y, theta = ax[0], ax[1], ax[2]
    return [
        (x + 3 * cos(theta) - 5) ** 2 + (y + 3 * sin(theta)) ** 2 - 25,
        x**2 + y**2 - 25,
        (x + 3 * cos(pi / 3 + theta)) ** 2 + (y + 3 * sin(pi / 3 + theta) - 6) ** 2 - 9,
    ]


# 以函数形式写入，还要给一个初始的猜测值
result = root(f, [-1.37, 4.80, 0.12])
print(result)
```

```
message: The solution converged.
success: True
 status: 1
    fun: [-2.005e-10 -1.399e-10 -4.021e-10]
      x: [ 1.158e+00  4.864e+00  2.143e-02]
 method: hybr
   nfev: 16
   fjac: [[-6.692e-01  3.392e-01  6.612e-01]
          [-6.187e-01 -7.471e-01 -2.429e-01]
          [-4.115e-01  5.717e-01 -7.098e-01]]
      r: [ 4.348e+00 -1.624e+00 -1.964e+01 -1.328e+01 -2.833e+01
           -1.615e+01]
    qtf: [ 3.054e-08 -4.885e-08 -4.575e-08]
```

`scipy` 求解微积分问题：

```python
from scipy.integrate import quad

def func(x, n, k):
    return x * n**k


data = quad(func, 0, 2, args=(2, 3))
print(data)
```

在这段代码中，首先定义了一个函数 `func(x, n, k)`，这个函数接受三个参数 x、n 和 k，并返回 `x * n^k` 的结果。接着使用 `quad` 函数来对函数 `func` 在区间 $[0, 2]$ 上进行数值积分，其中 `args=(2, 3)` 指定了参数 n=2 和 k=3。最后将积分的结果打印输出。

具体解释如下：

1. 定义函数 `func(x, n, k)`：
    - 这个函数接受三个参数 x、n 和 k。
    - 函数的计算逻辑是返回 x * n^k 的结果。
2. 使用 `quad` 函数进行数值积分：
    - `quad` 函数是 SciPy 库中用于数值积分的函数。
    - 第一个参数是要进行积分的函数，这里是函数 `func`。
    - 第二个参数是积分下限，这里是 0。
    - 第三个参数是积分上限，这里是 2。
    - `args=(2, 3)` 指定了额外的参数值，即 n=2 和 k=3。

二重定积分使用`dblquad`：

```python
from scipy.integrate import dblquad

def fun(x, y):
    return 3 * (x**2) * (y**2)


def y_area(x):
    return 1 - x**2


data = dblquad(fun, 0, 1, gfun=0, hfun=y_area)
print(data[0])
```

1. 导入需要的库：

    ```python
    from scipy.integrate import dblquad
    ```

    这里从 `scipy.integrate` 模块中导入了 `dblquad` 函数，该函数用于计算二重积分。

2. 定义被积函数 `fun(x, y)`：

```python
def fun(x, y):
    return 3 * (x**2) * (y**2)
```

这个函数接受两个参数 `x` 和 `y`，并根据表达式 `3 * (x**2) * (y**2)` 计算结果。

3. 定义积分区域的边界函数 `y_area(x)`：

```python
def y_area(x):
    return 1 - x**2
```

这个函数接受一个参数 `x`，并返回 `1 - x**2` 的结果。它定义了积分区域的上下边界函数。

4. 使用 `dblquad` 函数进行二重积分：

```python
data = dblquad(fun, 0, 1, gfun=0, hfun=y_area)
```

- 第一个参数是被积函数 `fun`。
- 第二个参数是 `x` 的积分下限，这里是 0。
- 第三个参数是 `x` 的积分上限，这里是 1。
- `gfun=0` 表示 `y` 的积分下限是常数 0。
- `hfun=y_area` 表示 `y` 的积分上限是函数 `y_area`。

5. 打印积分结果：

```python
print(data[0])
```

积分的结果存储在变量 `data` 中，通过 `data[0]` 可以获取到积分的数值结果，并将其打印输出。