第一问

由题可知，近似数 x_1 具有6位有效数字，且 x_1 可以写作：$x_1 = 10\times\left(1+2\times10^{-1}+\cdots\right)$

故其绝对误差限为：$\varepsilon\left(x_1\right)=0.5\times10^{-4}$，同理可得近似数 x_2 的绝对误差限为：
$\varepsilon\left(x_2\right)=0.5\times10^{-4}$

对于第一种算法，

绝对误差限：

$$\varepsilon\left(x_1-x_2\right)\le\varepsilon\left(x_1\right)+\varepsilon\left(x_2\right)=1\times10^{-4}\le0.5\times10^{-3}$$

相对误差限：

$$\varepsilon_r\left(x_1-x_2\right)=\frac{\varepsilon\left(x_1-x_2\right)}{\left|x_1-x_2\right|}\le\frac{1}{0.0021}\times10^{-4}\approx4.762\%$$

同时，$x_1-x_2=10^{-3}\times\left(2+1\times10^{-1}\right)$，因此第一种算法至少具有一位有效数字。

不妨令 $f\left(x,y\right)=\left(x^2+xy+y^2\right)^{-1}$，于是，$x_1^*-x_2^*=f\left(x_1^*,x_2^*\right)$

对于第二种算法，

绝对误差限：

$$\varepsilon\left[f\left(x_1^*,x_2^*\right)\right]\approx\left|f_x'\right|\varepsilon\left(x_1\right)+\left|f_y'\right|\varepsilon\left(x_2\right)\approx0.166\times10^{-7}$$

相对误差限：

$$\varepsilon_r\left[f\left(x_1,x_2\right)\right]=\frac{\varepsilon\left[f\left(x_1,x_2\right)\right]}{\left|f\left(x_1,x_2\right)\right|}\approx0.792\times10^{-5}$$

因此，第二种算法至少具有5位有效数字。

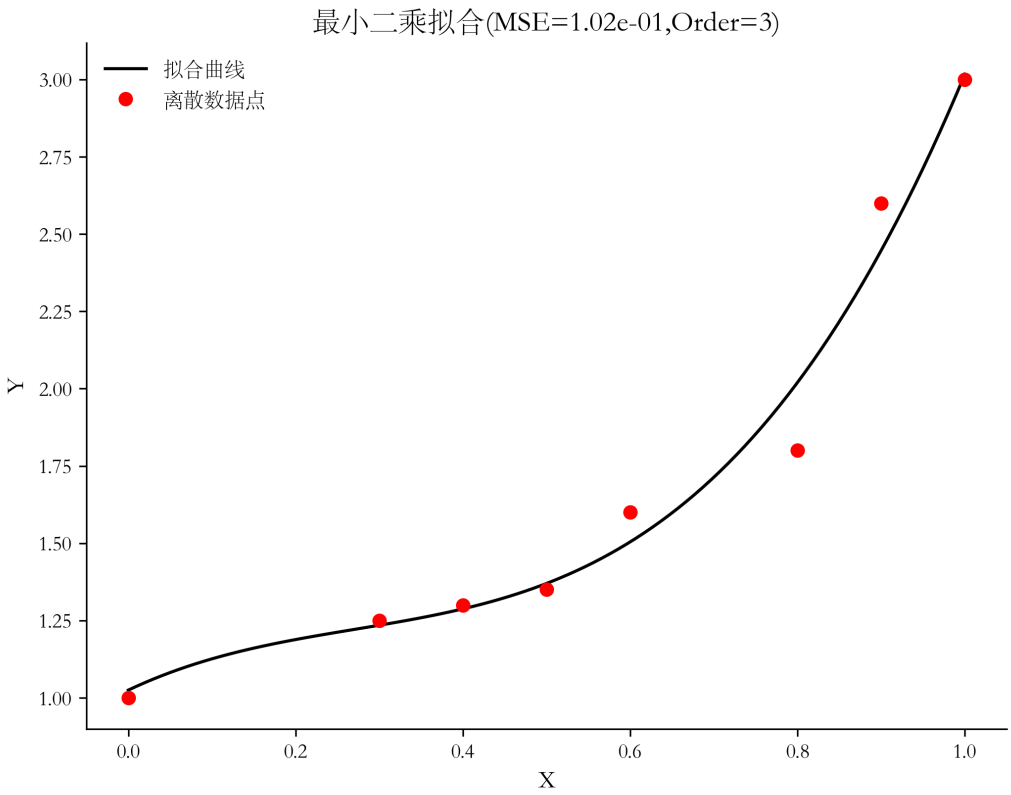
第二问

（代码求解）

解法方程组得到最终得到的多项式为：

$$y=3.786x^3+3.076x^2+1.281x+1.025$$

如图所示：



第三问

（代码求解）

（1）经计算可知，迭代矩阵为：

$$
\begin{bmatrix}
0, & -\frac{1}{4}, & -\frac{1}{20} \\
0, & \frac{1}{32}, & -\frac{19}{160} \\
0, & \frac{3}{64}, & -\frac{11}{960}
\end{bmatrix}
$$

其谱半径约为0.07小于1，因此迭代格式收敛。

（2）需要经至少7次迭代可以保证误差小于5e-6，迭代结果如下：

迭代次数	Xn([x1,x2,x3])
0	[0. 0. 0.]
1	[1.2 0.975 0.18472222]
2	[0.94701389 0.98353299 0.22830874]
3	[0.94270132 0.97862374 0.22820929]
4	[0.9439336 0.97848214 0.22798031]

迭代次数	Xn([x1,x2,x3])
5	[0.94398045 0.9785049 0.2279763]
6	[0.94397496 0.97850609 0.22797741]

最终，方程组的解约为[0.944，0.979，0.228]。

第四问

（1）证明：令 $f(x)=xe^{3x}-2$

容易得到：$f(0)=-2<0, f(0.5)\approx 0.24>0$ 因此，由零点存在性定理，$f(x)$ 在 $(0, 0.5)$ 上存在至少一个零点

对 $f(x)$ 求导得：$f'(x)=e^{3x}(3x+1)$，当 $x\geq 0$ 时，$f'(x)>0$ 恒成立，因此 $f(x)$ 在 $[0,+\infty)$ 上单调递增

因此，$f(x)$ 在 $[0,+\infty)$ 上存在唯一零点。

（2）（代码求解）

有（1）中的结果可知，取0.5时函数值更接近于0，因此取 $x=0.5$ 作为迭代初始值。使用牛顿迭代法。

迭代格式：

$$x_{k+1}=x_k-[f'(x_k)]\cdot f(x_k)=x_k-\frac{x_k e^{3x_k}-2}{e^{3x_k}(3x_k+1)}$$

结果如下：

迭代次数	迭代过程中x的取值（xn）	迭代过程中的误差
0	5.000000e-01	——
1	4.785041e-01	2.149587e-02
2	4.774705e-01	1.033602e-03
3	4.774683e-01	2.267356e-06
4	4.774683e-01	1.088168e-11
5	4.774683e-01	<1e-16

迭代过程中，xn的取值不断逼近于一个确切的值，从表格中数值的结果来看，牛顿迭代法的迭代格式是收敛的。最终得到根的近似值为：

$$x^*=0.48$$

第五问

（1）证明：由题可知，局部截断误差：

$$T_{n+1}=y(x_{n+1})-y(x_n)-\frac{h}{9}(2K_1+3K_2+4K_3)$$

$$=y(x_{n+1})-y(x_n)-\frac{2h}{9}f(x_n,y_n)-\frac{h}{3}f\left(x_n+\frac{1}{2}h,y_n+\frac{1}{2}hK_1\right)-\frac{4h}{9}f\left(x_n+\frac{3}{4}h,y_n+\frac{3}{4}hK_2\right)$$

上式在 (x_n,y_n) 处做泰勒展开：

$$y(x_{n+1})=y_n+hy_n'+\frac{h^2}{2!}y_n''+\frac{h^3}{3!}y_n'''+O(h^4)\cdots\cdots(1)$$

其中，

$$y_n'=f(x_n,y_n)=f_n,y_n''=f_x'(x_n,y_n)+f_y'(x_n,y_n)f_n=f_x'+f_y'f_n$$

$$y_n'''=f_{xx}''+2f_nf_{xy}'+f_n^2f_{yy}''+f_y'(f_x'+f_y'f_n)$$

$$f\left(x_n+\frac{1}{2}h,y_n+\frac{1}{2}hK_1\right)=f_n+(f_x'+f_y'f_n)\frac{h}{2}+\frac{1}{2!}\left(\frac{h^2}{4}f_{xx}''+\frac{h^2}{4}f_n^2f_{yy}''+\frac{h^2}{2}f_nf_{xy}'\right)+O(h^3)$$

$$=f_n+(f_x'+f_y'f_n)\frac{h}{2}+\left(\frac{f_{xx}''}{8}+\frac{f_n^2f_{yy}''}{8}+\frac{f_nf_{xy}'}{4}\right)h^2+O(h^3)\cdots\cdots(2)$$

$$f\left(x_n+\frac{3}{4}h,y_n+\frac{3}{4}hK_2\right)=f_n+(f_x'+K_2f_y')\frac{3h}{4}+\frac{1}{2!}\left(\frac{9h^2}{16}f_{xx}''+\frac{9h^2}{16}K_2^2f_{yy}''+\frac{9h^2}{8}f_{xy}'\right)+O(h^3)$$

$$=f_n+(f_x'+K_2f_y')\frac{3h}{4}+\left(\frac{9}{32}f_{xx}''+\frac{9}{32}K_2^2f_{yy}''+\frac{9}{8}K_2f_{xy}'\right)h^2+O(h^3)\cdots\cdots(3)$$

将 $(1)(2)(3)$ 带入 T_{n+1}：

$$T_{n+1}=y_n+hy_n'+\frac{h^2}{2!}y_n''+\frac{h^3}{3!}y'''-\frac{2h}{9}f_n-\frac{h}{3}\left[f_n+\left(f_x'+f_y'f_n\right)\frac{h}{2}+\left(\frac{f_{xx}''}{8}+\frac{f_n^2f_{yy}''}{8}+\frac{f_nf_{xy}'}{4}\right)h^2\right]$$

$$-\frac{4h}{9}\left[f_n+\left(f_x'+K_2f_y'\right)\frac{3h}{4}+\left(\frac{9}{32}f_{xx}''+\frac{9}{32}K_2^2f_{yy}''+\frac{9}{8}K_2f_{xy}'\right)h^2\right]+O\left(h^4\right)$$

$$=hf_n+\frac{h^2}{2}\left(f_x'+f_y'f_n\right)+\frac{h^3}{6}\left[f_{xx}''+2f_nf_{xy}'+f_n^2f_{yy}''+f_y'\left(f_x'+f_y'f_n\right)\right]-\frac{2h}{9}f_n$$

$$-\frac{h}{3}f_n-\frac{h^2}{6}\left(f_x'+f_y'f_n\right)-\frac{1}{3}\left(\frac{f_{xx}''}{8}+\frac{f_n^2f_{yy}''}{8}+\frac{f_nf_{xy}'}{4}\right)h^3$$

$$-\frac{4h}{9}f_n-\frac{h^2}{3}\left(f_x'+K_2f_y'\right)-\frac{1}{8}\left(f_{xx}''+K_2^2f_{yy}''+2K_2f_{xy}'\right)h^3+O\left(h^4\right)$$

$$=\frac{h^2}{2}\left(f_x'+f_y'f_n\right)-\frac{h^2}{6}\left(f_x'+f_y'f_n\right)-\frac{h^2}{3}\left(f_x'+K_2f_y'\right)+\frac{h^3}{6}\left[f_{xx}''+2f_nf_{xy}'+f_n^2f_{yy}''+f_y'\left(f_x'+f_y'f_n\right)\right]$$

$$-\frac{1}{8}\left(f_{xx}''+K_2^2f_{yy}''+2K_2f_{xy}'\right)h^3-\frac{1}{3}\left(\frac{f_{xx}''}{8}+\frac{f_n^2f_{yy}''}{8}+\frac{f_nf_{xy}'}{4}\right)h^3+O\left(h^4\right)\cdots\cdots(4)$$

$$K_2=f\left(x_n+\frac{1}{2}h,y_n+\frac{1}{2}hK_1\right)=f_n+\left(f_x'+f_y'f_n\right)\frac{h}{2}+\left(\frac{f_{xx}''}{8}+\frac{f_n^2f_{yy}''}{8}+\frac{f_nf_{xy}'}{4}\right)h^2+O\left(h^3\right)\cdots\cdots(5)$$

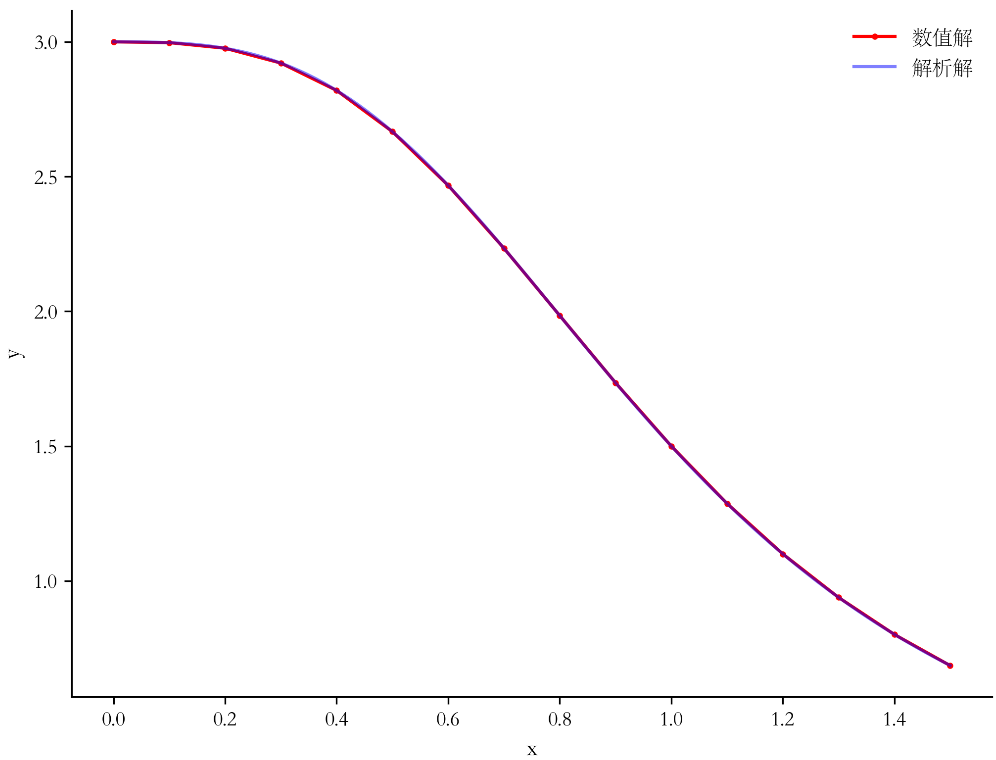
将(5)式带入(4)式，可以发现原式中二次项以及三次项被消去，因此该公式至少是三阶公式。

（2）（代码求解）结果如下：

Xn	Yn	精确解	误差
0.0	3.000000e+00	3.000000e+00	0.000000e+00
0.1	2.997003e+00	2.997003e+00	4.664649e-07
0.2	2.976188e+00	2.976190e+00	2.849130e-06
0.3	2.921123e+00	2.921130e+00	6.401320e-06
0.4	2.819545e+00	2.819549e+00	4.228432e-06
0.5	2.666681e+00	2.666667e+00	-1.437835e-05
0.6	2.467157e+00	2.467105e+00	-5.149769e-05
0.7	2.233895e+00	2.233805e+00	-8.975787e-05
0.8	1.984228e+00	1.984127e+00	-1.008023e-04
0.9	1.735175e+00	1.735107e+00	-6.770531e-05
1.0	1.499997e+00	1.500000e+00	2.971329e-06

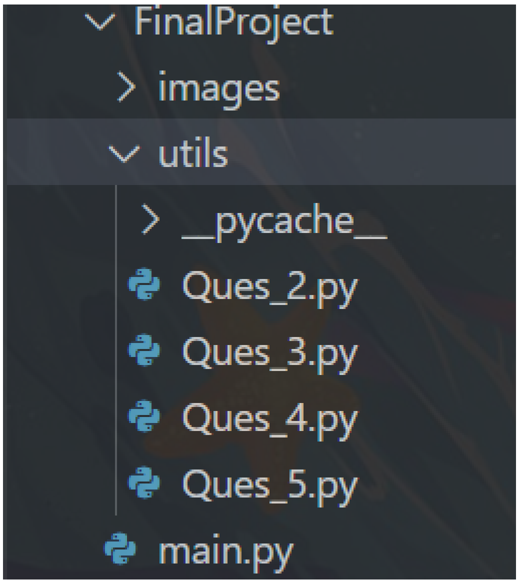
Xn	Yn	精确解	误差
1.1	1.286912e+00	1.287001e+00	8.900664e-05
1.2	1.099539e+00	1.099707e+00	1.678109e-04
1.3	9.381540e-01	9.383797e-01	2.257672e-04
1.4	8.010229e-01	8.012821e-01	2.591613e-04
1.5	6.854435e-01	6.857143e-01	2.707816e-04

如图所示：



附录

按如图所示形式组织各个文件，并运行main.py即可复现结果(images用于存放正文中的图片)。



main.py

```
import numpy as np
import sympy as sp

#=======第2题=======#
from utils.Ques_2 import SolutionToQuesTwo
print("#=======第2题=======#")
t = sp.Symbol('t')
# 勒让德多项式
fun_list = [1, t, (3*t**2-1)/2, (5*t**3-3*t)/2]
# 题目中给出的离散数据点
x = np.array([0, 0.3, 0.4, 0.5, 0.6, 0.8, 0.9, 1])
y = np.array([1, 1.25, 1.3, 1.35, 1.6, 1.8, 2.6, 3])
# 由于勒让德多项式多项式的权函数为1，故计算时采用默认值
sol_2 = SolutionToQuesTwo(x, y, k=3, fun_list=fun_list)
sol_2.fit_curve()
print('拟合多项式为：{}'.format(sol_2.fit_poly))
sol_2.plt_curve_fit()
print("="*20)
print()

#=======第3题=======#
from utils.Ques_3 import SolutionToQuesThree
print("#=======第3题=======#")
mat = np.array([[20,5,1,24],[1,8,1,9],[3,-3,18,4]])      # 方程组的增广矩阵
sol_3 = SolutionToQuesThree(matrix=mat, x0=None,
                            max_iter=1000, epsilon=5e-6, is_print=True)
# 计算迭代矩阵的谱半径
eigval, _ = np.linalg.eig(sol_3.G)
spe_radius = np.max(np.abs(eigval))
if spe_radius < 1:
    print("迭代矩阵的谱半径为：", spe_radius)
    print("迭代矩阵的谱半径小于1，迭代矩阵收敛")
print("="*20)
print()

#=======第4题=======#
from utils.Ques_4 import SolutionToQuesFour
```

```
x = sp.Symbol('x')
fun = x*sp.exp(3*x)-2
sol_4 = SolutionToQuesFour(fun=fun, x0=0.5, eps=1e-15, is_print=True)
print()

#=======第5题=======#
from utils.Ques_5 import SolutionToQuesFive
print("#=======第5题=======#")
# 题目中的微分方程
def f(x, y):
    return -x**2*y**2
# 题目中微分方程的精确解
def g(x):
    return 3/(1+x**3)
sol_5 = SolutionToQuesFive(fun=f, interval=[0,1.5],
                           x0=0, y0=3, h=0.1, is_print=False)
com_df = sol_5.res_df
com_df['解析解'] = com_df['Xn'].apply(g)
com_df['误差'] = com_df['解析解'] - com_df['Yn']
print("最终结果")
print("-"*20)
print(com_df)
sol_5.plot(is_show=True)
print("-"*20)
print("="*20)
```

Ques_2.py

```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

matplotlib.rcParams['font.sans-serif'] = ['STSong']
matplotlib.rcParams['axes.unicode_minus'] = False

class SolutionToQuesTwo:

    """
    第二问
    """

    def __init__(self, x, y, k=None, w=None, fun_list = None):

        """
        参数初始化\n
        x, y: 离散数据点，离散数据点的长度需一致\n
        k: 进行多项式拟合时，必须指定的多项式的最高阶次\n
        w: 权系数，长度需与离散数据点的长度一致，默认情况下为1\n
        fun_list: 自定义的基函数,本题中为勒让德多项式
        """

        self.x = np.asarray(x, dtype=np.float64)
        self.y = np.asarray(y, dtype=np.float64)
        # 如果进行多项式拟合时，此变量为多项式最高阶次，需要指定
```

```python
        self.k = k
        if len(self.x) != len(self.y):
            raise ValueError("离散点数据长度不一致！")
        else:
            self.n = len(self.x)    # 离散点的个数
        if w is None:
            self.w = np.ones(self.n)    # 默认情况下，所有数据权重一致为1
        else:
            if len(w) != self.n:
                raise ValueError("权重长度与离散数据点不一致！")
            else:
                self.w = np.asarray(w, dtype=np.float64)
        self.fit_poly = None                # 曲线拟合的多项式
        self.poly_coefficient = None        # 基函数前的系数组成的向量
        self.polynomial_orders = None       # 系数的阶次
        self.fit_error = None               # 拟合的误差向量
        self.mse = np.infty                 # 拟合的均方根误差
        self.fun_list = list(fun_list)      # 自定义的基函数，本题中为勒让德多项式
        self.m = len(fun_list)              # 自定义基函数的个数


    def __lambdified__(self,fun_list):

        """
        将传入的自定义基函数转化为可以进行计算的函数
        """

        fun_list1 = []
        for fun in fun_list:
            if isinstance(fun, float):
                raise ValueError("请避免使用小数！")
            if isinstance(fun, int):
                fun_list1.append(int(fun))
            else:
                t = fun.free_symbols.pop()
                fun = sp.lambdify(t, fun)
                fun_list1.append(fun)
        return fun_list1

    def fit_curve(self):

        """
        最小二乘法拟合
        """

        C = np.zeros((self.m,self.m))   # 初始化法方程系数矩阵
        d = np.zeros(self.m)            # 初始化法方程右端向量
        # 转化自定义的基函数
        self.fun_list1 = self.__lambdified__(self.fun_list)
        # 提取基函数列表中元素的类型
        fun_type = list(map(type, self.fun_list1))
        # 元素为函数表达式时元素的类型
        function = None
        for t in fun_type:
            if t == int:
```

```
                continue
            if t != int:
                function = t
                break
        # 构造法方程的系数矩阵
        for i in range(self.m):
            for j in range(self.m):
                if isinstance(self.fun_list1[i],function) and\
                   isinstance(self.fun_list1[j],function):
                    C[i,j] = np.dot(self.w,
                             self.fun_list1[i](self.x)*self.fun_list1[j]
(self.x))
                if isinstance(self.fun_list1[i],int) and \
                   isinstance(self.fun_list1[j],function):
                    C[i,j] = np.dot(self.w,
                             self.fun_list1[i]*self.x*self.fun_list1[j](self.x))
                if isinstance(self.fun_list1[i],function) and \
                   isinstance(self.fun_list1[j],int):
                    C[i,j] = np.dot(self.w,
                             self.fun_list1[i](self.x)*self.fun_list1[j]*self.x)
                if isinstance(self.fun_list1[i],int) and \
                   isinstance(self.fun_list1[j],int):
                    C[i,j] = np.dot(self.w,
                             self.fun_list1[i]*self.fun_list1[j]*self.x)
        # 构造法方程的右端向量
        for i in range(self.m):
            if isinstance(self.fun_list1[i],function):
                d[i] = np.dot(self.w, self.fun_list1[i](self.x)*self.y)
            if isinstance(self.fun_list1[i],int):
                d[i] = np.dot(self.w, self.fun_list1[i]*self.x*self.y)
        # 求解法方程
        self.poly_coefficient = np.linalg.solve(C,d)

        t = sp.symbols('t')
        self.fit_poly = self.poly_coefficient[0]*self.fun_list[0]
        for p in range(1,self.m):
            self.fit_poly += self.poly_coefficient[p]*self.fun_list[p]

        self.__cal_fit_error__()    # 误差分析


    def __cal_fit_error__(self):

        """
        计算拟合的误差和均方根误差
        """

        y_fit = self.__cal_x0__(self.x)
        self.fit_error = self.y - y_fit      # 误差向量
        self.mse = np.sqrt(np.mean(self.fit_error**2))     # 均方根误差

    def __cal_x0__(self,x0):

        """
        求解给定数值x0的拟合值
```

```
        """

        t = self.fit_poly.free_symbols.pop()
        fit_poly = sp.lambdify(t, self.fit_poly)
        return fit_poly(x0)

    def plt_curve_fit(self,is_show=True):

        """
        拟合曲线以及离散数据点的可视化
        """

        xi = np.linspace(self.x.min(), self.x.max(), 100)
        yi = self.__cal_x0__(xi)    # 拟合值
        if is_show:
            plt.figure(figsize=(8, 6), facecolor="white", dpi=300)
        plt.plot(xi, yi, 'k-', lw=1.5, label="拟合曲线")
        plt.plot(self.x, self.y, 'ro', lw=1.5, label="离散数据点")
        if self.k:
            plt.title("最小二乘拟合(MSE=%.2e,Order=%d)"%(self.mse,self.k),
                      fontdict={"fontsize":14})
        else:
            plt.title("最小二乘拟合(MSE=%.2e)"%(self.mse),fontdict={"fontsize":14})
        plt.xlabel('X',fontdict={"fontsize":12})
        plt.ylabel("Y",fontdict={"fontsize":12})
        plt.gca().spines['right'].set_visible(False)
        plt.gca().spines['top'].set_visible(False)
        plt.legend(loc='best', frameon=False)
        if is_show:
            plt.savefig(r"FinalProject\images\image_2")
            plt.show()
```

Ques_3.py

```
import numpy as np
import pandas as pd
from fractions import Fraction

class SolutionToQuesThree:

    """
    第三问
    """

    def __init__(self, matrix:np.ndarray, x0=None,
                 max_iter:int=1000, epsilon:float=1e-15, is_print:bool=True):

        """
        参数初始化\n
        matrix: 方程组的增广矩阵\n
        x0: 初始向量\n
        epsilon: 解的精度\n
        如果想获取迭代后的方程组的解以便调用,请查询类属性self.solution
        """
```

```
        self.epsilon = epsilon
        self.max_iter = max_iter
        self.is_print = is_print
        self.iter_num = 0    # 初始化，用于记录迭代次数
        check_matrix = matrix.copy()
        if check_matrix[:,:-1].shape[0] != check_matrix[:,:-1].shape[1]:
            raise ValueError('线性方程组的系数矩阵不是方阵')
        if np.diag(check_matrix[:,:-1]).any() == 0:
            raise ValueError('线性方程组的系数矩阵的对角线元素存在0')
        if np.linalg.matrix_rank(check_matrix[:,:-1]) != \
           np.linalg.matrix_rank(matrix):
            raise ValueError('该线性方程组无解')
        if np.linalg.matrix_rank(check_matrix[:,:-1]) ==\
           np.linalg.matrix_rank(matrix) \
           and np.linalg.matrix_rank(check_matrix[:,:-1]) <\
           check_matrix[:,:-1].shape[0]:
            raise ValueError('该线性方程组有多个解,暂时支持有唯一解的方程组')
        if x0 is not None:
            if x0.shape[0] != matrix.shape[1]-1:
                raise ValueError('初始向量的维度与方程组的维度不匹配')
            else:
                self.x0 = np.asarray(x0, dtype=np.float64)
        else:
            self.x0 = np.zeros(matrix.shape[1]-1, dtype=np.float64)
        self.A = matrix[:,:-1]       # 线性方程组的系数矩阵
        self.b = matrix[:,-1]        # 线性方程组的右端向量
        self.D = np.diag(np.diag(self.A))         # D矩阵
        self.L = np.diag(np.diag(self.A)) - np.tril(self.A)    # L矩阵
        self.U = np.diag(np.diag(self.A)) - np.triu(self.A)    # U矩阵
        self.G = None        # 高斯赛德尔迭代法的迭代矩阵
        self.f = None        # 迭代矩阵
        # 存储迭代过程中方程组的解
        self.solution_list = np.zeros(self.max_iter, dtype=object)
        self.solution = None       # 方程组的解
        self.res_df = None       # 存储迭代过程中方程组的解，尝试以DataFrame存储
        self.__solve__()
        if is_print:
            self.__print__()


    def __solve__(self):

        """
        高斯-赛德尔迭代法
        """

        self.G = np.linalg.inv(self.D-self.L).dot(self.U)
        self.f = np.linalg.inv(self.D-self.L).dot(self.b)
        self.solution_list[0] = self.x0.copy()           # 初始向量
        eigval, _ = np.linalg.eig(self.G)
        if np.max(np.abs(eigval)) > 1:
            raise ValueError("该方程组存在不稳定的迭代方法")
        i = 1
        while i < self.max_iter:
            self.solution_list[i] = self.G.dot(self.solution_list[i-1]) + self.f
```

```
            # 退出循环，满足精度要求时退出循环
            if np.linalg.norm((self.solution_list[i] -\
                               self.solution_list[i-1]), np.inf) < self.epsilon:
                break
            i += 1
        self.iter_num = i    # 记录最终的迭代次数
        # 最终的误差
        self.error = np.linalg.norm((self.solution_list[i-1] -\
                                     self.solution_list[i-2]), np.inf)
        # 方程组的解
        self.solution = self.solution_list[i-1]
        # 储存迭代过程
        self.res_df = pd.DataFrame()
        self.res_df["迭代次数"] = list(range(self.iter_num))
        self.res_df["Xn"] = self.solution_list[:self.iter_num]
        self.res_df.set_index("迭代次数", inplace=True)


    def __print__(self):

        """
        打印输出结果
        """

        np.set_printoptions(
            formatter={'all':lambda x:str(Fraction(x).limit_denominator())})
        print("D矩阵：\n", self.D)
        print("L矩阵：\n", self.L)
        print("U矩阵：\n", self.U)
        print("迭代过程：\n", self.res_df)
        print("高斯赛德尔迭代法的迭代矩阵G：\n", self.G)
        print("高斯赛德尔迭代法的迭代矩阵f：\n", self.f)
        print("高斯赛德尔迭代法解方程组的解：")
        print({"x"+str(i+1):round(self.solution[i], 3)
               for i in range(len(self.solution))})
        print("最终的误差：%e"%self.error)
        print("迭代次数：%d"%self.iter_num)
        np.set_printoptions()
```

Ques_4.py

```
import sympy as sp
import pandas as pd

class SolutionToQuesFour:

    """
    第四问，采用牛顿迭代法
    """

    def __init__(self, fun, x0, eps:float=1e-12, is_print:bool=True):

        """
        参数初始化\n
        fun：带求根的方程\n
```

```python
        x0：迭代初始值\n
        interval：求根区间\n
        eps：求根精度要求\n
        """

        self.fun = fun    # 符号定义
        self.eps = eps
        self.error = None  # 求根误差
        self.res = None    # 求根结果
        self.x0 = x0
        self.xn, self.delta_xn = [self.x0], [" "]  # 记录迭代结果
        self.__cal_res__()
        if is_print:
            self.__print__()


    def __cal_res__(self):

        """
        计算方程的根
        """

        t = self.fun.free_symbols.pop()
        # 牛顿迭代法的迭代格式
        iter_fun = t - self.fun/sp.diff(self.fun, t)
        # 将方程转化为lambda函数
        iter_fun = sp.lambdify(iter_fun.free_symbols.pop(), iter_fun)
        # 进行迭代，两个迭代值之间的误差小于精度要求，迭代结束
        while True:
            self.xn.append(iter_fun(self.xn[-1]))
            self.delta_xn.append(abs(self.xn[-1] - self.xn[-2]))
            if self.delta_xn[-1] < self.eps:
                self.res = self.xn[-1]
                self.error = self.delta_xn[-1]
                break


    def __print__(self):

        """
        打印输出结果
        """

        pd.set_option('display.float_format', lambda x: '%e' % x)
        pd.set_option('display.unicode.ambiguous_as_wide', True)
        pd.set_option('display.unicode.east_asian_width', True)

        print("牛顿迭代实验数据")
        print("="*80)
        iter_df = pd.DataFrame()
        iter_df["迭代次数"] = range(len(self.xn))
        iter_df["迭代过程中x的取值(xn)"] = self.xn
        iter_df["迭代过程中的误差"] = self.delta_xn
        iter_df.set_index("迭代次数", inplace=True)
        print(iter_df)
```

```
        print("="*80)
        print("最终根：%e"%self.res)
        print("误差：%.4e"%self.error)
```

Ques_5.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

matplotlib.rcParams['font.sans-serif'] = ['STSong']
matplotlib.rcParams['axes.unicode_minus'] = False

class SolutionToQuesFive:

    """
    第五问
    """

    def __init__(self, fun, interval:list, x0:float, y0:float,
                 h:float=0.001, is_print:bool=True):

        """
        参数初始化\n
        fun：待求解的方程(df/dx = fun(x,y))，使用数值定义
        interval：方程的求解区间([a,b])
        x0,y0：初值条件
        h：步长
        is_print：是否打印求解结果
        """

        self.fun = fun
        if type(interval) is not list:
            raise TypeError("求解区间请以列表的形式传入")
        else:
            if len(interval) != 2:
                raise ValueError("求解区间设置不对，请以[a,b]的形式传入")
        self.interval = np.asarray(interval)
        a, b, self.h = interval[0], interval[1], h
        self.xn = np.arange(a, b+h, h)
        self.yn = np.zeros(len(np.arange(a, b+h, h))) # 储存结果
        self.x0, self.yn[0] = x0, y0
        self.res_df = None      # 最终结果，以DataFrame的形式呈现
        self.__cal_res__()
        if is_print:
            self.__print__()


    def __cal_res__(self):

        """
        计算结果
        """
```

```python
        for i in range(1, len(self.xn)):
            K1 = self.fun(self.xn[i-1], self.yn[i-1])
            K2 = self.fun(self.xn[i-1] + self.h/2,
                          self.yn[i-1] + self.h*K1/2)
            K3 = self.fun(self.xn[i-1] + 3*self.h/4,
                          self.yn[i-1] + 3*self.h*K2/4)
            self.yn[i] = self.yn[i-1] + self.h*(2*K1 + 3*K2 + 4*K3)/9

        # 最终结果，以DataFrame的形式呈现
        df = pd.DataFrame()
        df["Xn"] = self.xn
        df["Yn"] = self.yn
        self.res_df = df


    def __print__(self):

        """
        打印求解结果
        """

        pd.set_option('display.max_rows', None)
        pd.set_option('display.unicode.ambiguous_as_wide', True)
        pd.set_option('display.unicode.east_asian_width', True)

        print("该方程的数值解为")
        print("-"*20)
        print(self.res_df)
        print("-"*20)


    def plot(self, is_show:bool=True):

        """
        绘制结果
        is_show: 是否显示图像
        """

        if is_show:
            plt.figure(figsize=(8,6), facecolor="white", dpi=300)
        plt.title("该方程的数值解", fontsize=14)
        x = np.asarray(self.xn)
        y = np.asarray(self.yn)
        plt.plot(x, y, 'o-', linewidth=1.5, markersize=4)
        plt.xlabel("x", fontsize=12)
        plt.ylabel("y", fontsize=12)
        plt.gca().spines['right'].set_visible(False)
        plt.gca().spines['top'].set_visible(False)
        if is_show:
            plt.savefig(r"FinalProject\images\image_5")
            plt.show()
```