上海电力大学

实践课程报告

学　　院：数理学院

专　　业：信息与计算科学专业

课程编号：2812101.04　课程名称：数值计算方法训练

学生姓名：刘伟涛　学号：20222421　班级：2022122

指导老师：某老师

2024 年 5 月 28 日

成绩：

教师评语：

一、非线性方程组的 Newton 法

（1）请详细论述非线性方程组的 Newton 法。（加分项）
（2）用 Newton 法求解非线性方程组

$$F(x) = \begin{bmatrix} (x_1+3)(x_2^3-7)+18 \\ \sin(x_2 e^{x_1}-1) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{1}$$

初始点取 $x^{(0)} = (0,0)^T$，终止条件为 $\left\|F(x^{(k)})\right\|_2 \leq 10^{-6}$。汇报计算结果，迭代次数及每次迭代的残量 $\left\|F(x^{(k)})\right\|_2$

1. 题目分析与思路

设有单变量方程 $f(x)=0$ 牛顿法求根的迭代公式为

$$\begin{cases} x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \\ x_(0,0) = 0 \end{cases} \quad n = 0,1,\cdots$$

给定 $\infty 0$ 给定初始值 $x_(0,0)$， 为根的容许误差，tol 为的精度要求，设 frequency 为当前迭代次数。
(1) 计算 $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}, n = n+1$
(2) 若 $|x_1 - x_0| < \varepsilon$，则输出近似根 ci 及迭代次数 n，程序结束。否则转 (1)

2. 数值计算方法

首先, 求 F1,F2 对于 x,y 的偏导数。

$$(x_1+3)(x_2^3-7)+18$$

$$dx_1 \Rightarrow \theta x_1^3 - 7$$

$$dx_2 \Rightarrow 3x_1x_2 + 9x_2^2$$

$$\sin(x_2 e^{x_1} - 1)$$

$$dx_1 \Rightarrow \theta x_2 cos((x_2 e^(x_1-1))e_1^x)$$

$$dx_2 \Rightarrow cos((x_2 e^x - 1)e^x)$$

关于牛顿迭代法，设方程组为

$$F'(x_1^{(0)}, x_2^{(0)})[\begin{matrix} \varDelta X_1^{(0)} \\ \varDelta X_2^{(0)} \end{matrix}] = -F(x_1^{(0)}, x_2^{(0)}) \tag{2}$$

$$[\begin{matrix} X_1^{(1)} \\ X_2^{(1)} \end{matrix}] = [\begin{matrix} X_1^{(0)} \\ X_2^{(0)} \end{matrix}] + [\begin{matrix} \varDelta X_1^{(0)} \\ \varDelta X_2 \end{matrix}] \tag{3}$$

每次将 (3) 式中的结果带入 (2) 式中，直到满足终止条件。

3. Python 代码

```python
'''第 1 题：非线性方程组的 Newton 法'''
import numpy as np
import math

e=math.e
def f1(x,y):
    return (x+3)*(y**3-7)+18

def f2(x,y):
    return math.sin(y*e**x-1)

def dxf1(x,y):
    return y**3-7

def dyf1(x,y):
    return 3*x*y+9*y**2

def dxf2(x,y):
    return y*math.cos((y*e**x-1)*e**x)

def dyf2(x,y):
    return math.cos((y*e**x-1)*e**x)

def Newton(f1,f2,y0,x0,tol=1e-6,frequency=0):
    x1=x0
    y1=y0
    tol1=1
    while (tol<tol1):
        A=np.linalg.inv(np.array([[dxf1(x0,y0),dyf1(x0,y0)],[dxf2(x0,y0),dyf2(x0,y0)]]))
        B=np.array([[f1(x0,y0)],[f2(x0,y0)]])
        X=A@-B
        x1=x0+X[0,0]
        y1=y0+X[1,0]
        tol1=abs(x1-x0)
        x0=x1
        y0=y1
        frequency=frequency+1
        print('第{}次，残量为：{}'.format(frequency, tol1))

    return x1,y1
x0=0
y0=0
print('迭代结果为：')
print(Newton(f1,f2,y0,x0))
```

4. 结果分析

运行的迭代结果为：

步数	残量
1	0.42857142857142855
2	0.27638000665965523
3	0.14100261406768316
4	0.011152133275396825
5	$3.6673596615888034e-05$
6	$9.720775516202986e-10$

最后结果为 $(1.1068554027794276e-16, 0.9999999999999999)$

$$1.1068554027794276e-16 \approx 0$$

$$0.9999999999999999 \approx 1$$

所以结果约等于为 0 和 1。

二、常微分方程数值解

取步长 h=0.1，分别用 Euler 法与 4 阶 Runge-Kutta 法解下列初值问题

$$\begin{cases} y' = 1 + (x-y)^2, \\ y(2) = 1, \end{cases} \quad 2 \leq x \leq 4$$

该问题的精确解为并将结果与精确解 $y = x + \frac{1}{1-x}$ 比较。

1. 题目分析与思路

根据题目分别用两种方法求解，并且与题目中的标准式做比较。并用图片的方法直观的展示结果。

2. 数值计算方法

了解相关算法：
关于 4 阶 Runge-Kutta 法

$$\begin{aligned} &\mathrm{y_{n+1}} = \mathrm{y_n} + \frac{\mathrm{h}}{6}(\mathrm{k_1} + 2\mathrm{k_2} + 2\mathrm{k_3} + \mathrm{k_4}) \\ &\mathrm{k_1} = \mathrm{f(x_n, y_n)} \\ &\mathrm{k_2} = \mathrm{f}(\mathrm{x}_{\mathrm{n}+\frac{1}{2}}, \mathrm{y_n} + \frac{\mathrm{h}}{2}\mathrm{k_1}) \\ &\mathrm{k_3} = \mathrm{f}(\mathrm{x}_{\mathrm{n}+\frac{1}{2}}, \mathrm{y_n} + \frac{\mathrm{h}}{2}\mathrm{k_2}) \\ &\mathrm{k_4} = \mathrm{f}(\mathrm{x_{n+1}}, \mathrm{y_n} + \mathrm{hk_3}) \end{aligned} \tag{4}$$

关于 Euler 法

$$\begin{cases} y'(x) = y - \frac{2x}{y}, x \in [0,1] \\ y(0) = 1 \end{cases} \tag{5}$$

3. Python 代码

```python
'''常微分方程数值解'''
import numpy as np
import matplotlib.pyplot as plt
def f(x,y):
    return 1+(x-y)**2

t0, tf, x0, h = 2, 4, 1, 0.1

def euler_method(f, t0, tf, y0, h):
    n_steps = int((tf - t0) / h)
    x = np.linspace(t0, tf, n_steps + 1)
    y = np.zeros(n_steps + 1)
    y[0] = y0

    for i in range(n_steps):
        y[i + 1] = y[i] + h * f(y[i], y[i])

    return x, y

def runge_kutta_4(f, t0, tf, y0, h):
    n_steps = int((tf - t0) / h)
    x = np.linspace(t0, tf, n_steps + 1)
    y = np.zeros(n_steps + 1)
    y[0] = y0

    for i in range(n_steps):
        k1 = h * f(x[i], y[i])
        k2 = h * f(x[i] + h/2, y[i] + k1/2)
        k3 = h * f(x[i] + h/2, y[i] + k2/2)
        k4 = h * f(x[i] + h, y[i] + k3)
        y[i + 1] = y[i] + (k1 + 2*k2 + 2*k3 + k4) / 6

    return x, y

x_rk, y_rk = runge_kutta_4(f, t0, tf, x0, h)
print("Runge-Kutta:")
print("t =", x_rk)
print("x =", y_rk)

x_euler, y_euler = euler_method(f, t0, tf, x0, h)
print("Euler:")
print("t =", x_euler)
print("x =", y_euler)

def func(x):
    return x+1/(1-x)

x = np.linspace(1, 5, 400)
y=func(x)
plt.plot(x, y, label='Function')

```

```
plt.xlabel('X')
plt.ylabel('Y')

plt.scatter(x_euler, y_euler, color='red')

plt.savefig('数理方程\math201-latex-report-main\math201-latex-report-main\images/Figure_1.png')
plt.show()
x = np.linspace(1, 5, 400)
y=func(x)
plt.plot(x, y, label='Function')

plt.xlabel('X')
plt.ylabel('Y')
plt.scatter(x_rk, y_rk, color='blue')


plt.savefig('数理方程\math201-latex-report-main\math201-latex-report-main\images/Figure_2.png')
plt.show()
```

4. 结果分析

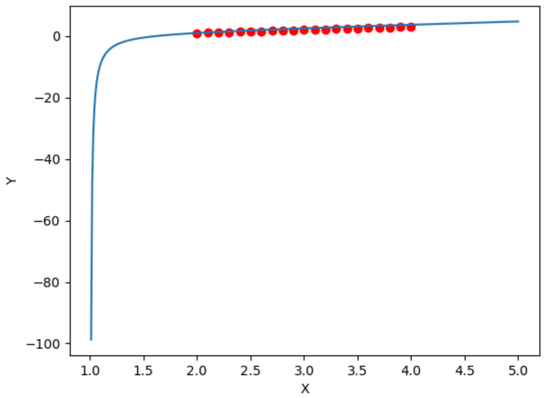

图 1: Euler 法

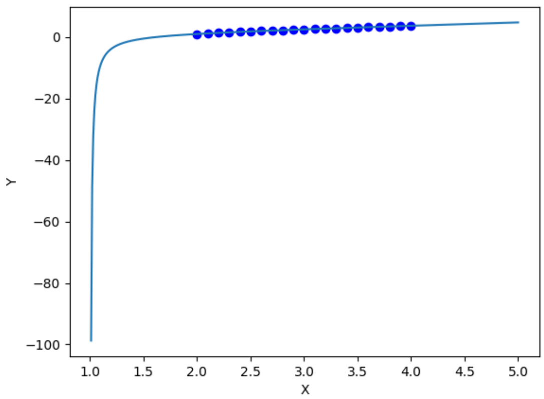

图 2: Runge-Kutta 法

如图可见，散点与函数线基本拟合

三、最小二乘问题

在服药后的每小时度量的药物血液浓度数据由下表给出。使用模型 $W = c_1 t e^{c_2 t}$ 拟合。找出估计的极大值以及半衰期。假设药物的疗效范围是 4 15ng/ml，使用你选择的方程求解器估计药物有效的时间。

小时	浓度 (ng/ml)	小时	浓度 (ng/ml)
1	6.2	6	13.5
2	9.5	7	13.3
3	12.3	8	12.7
4	13.9	9	12.4
5	14.6	10	11.9

1. 题目分析与思路

在服药后的每小时度量的药物血液浓度的关系使用模型 $W = c_1 t e^{c_2 t}$ 拟合。然后用二分法找出估计的极大值以及用找出半衰期。

2. 数值计算方法

$$A = \begin{bmatrix} 1 & 1n1 & 1 \\ 1 & 1n2 & 2 \\ 1 & \ln 3 & 3 \\ 1 & \ln 4 & 4 \\ 1 & \ln 5 & 5 \\ 1 & \ln 6 & 6 \\ 1 & \ln 7 & 7 \\ 1 & \ln 8 & 8 \\ 1 & \ln 9 & 9 \\ 1 & \ln 10 & 10 \end{bmatrix} f = \begin{pmatrix} \ln 6.2 \\ \ln 9.5 \\ \ln 12.3 \\ \ln 13.9 \\ \ln 14.6 \\ \ln 13.9 \\ \ln 13.3 \\ \ln 12.7 \\ \ln 12.4 \\ \ln 11.9 \end{pmatrix} \tag{6}$$

3. Python 代码

```python
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

list=[6.2,9.5,12.3,13.9,14.6,13.5,13.3,12.7,12.4,11.9]
A=np.array([[1 for i in range(1,11)],[math.log(i) for i in range(1, 11)],[i for i in range(1, 11)]])
Ln_list=np.array(list).reshape(10,1)
Ln = np.log(Ln_list)
ln_t=np.array([math.log(i) for i in range(1, 11)])
A=A.T
```

```python
ln_t=ln_t.reshape(-1, 1)
B=np.dot(np.linalg.inv(A.T@A),(A.T@(Ln-ln_t)))
print(B)

def func(x,B):
    return np.exp(B[0])*x*np.exp(B[2]*x)

x = np.linspace(1, 10, 400)
y = func(x,B)

m=([i for i in range(1, 11)])
plt.figure(figsize=(10, 6))
plt.scatter(m, list)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')

plt.savefig('数理方程\math201-latex-report-main\math201-latex-report-main\images/Figure_3.png')
plt.show()
def dtfunc(x,B):
    return B[0]*np.exp(B[2]*x)+B[0]*x*B[2]*np.exp(B[2]*x)
def bisection_method(B, a, b, tol=1e-6, max_iter=1000):
    if dtfunc(a, B) * dtfunc(b, B) >= 0:
        print(" 无")
        return "Not found"

    for _ in range(max_iter):
        c = (a + b) / 2
        if abs(dtfunc(c, B)) < tol:
            return c
        elif dtfunc(a, B) * dtfunc(c, B) < 0:
            b = c
        else:
            a = c
a, b = 1, 10
root = bisection_method(B, a, b)
print(f" 最大值时 t 大约为: {root}")
print(f" 最大值时 W 大约为: {func(root,B)}")

def funcy(x, B, target_y):
    return np.exp(B[0]) * x * np.exp(B[2] * x) - target_y

x4 = fsolve(lambda x: funcy(x, B, 4), 1)
x4_value = x4[0]

x15 = fsolve(lambda x: funcy(x, B, 15), 1)
x15_value = x15[0]

x42 = fsolve(lambda x: funcy(x, B, 4), 20)
x42_value = x42[0]

x152 = fsolve(lambda x: funcy(x, B, 15), 2)
x152_value = x152[0]

```

```python
print(f" 有效期为{x4_value}~{x15_value}至{x42_value}~{x152_value}")
N=func(root,B)/2
x_2 = fsolve(lambda x: funcy(x, B, N), 15  )
x_2_value = x_2[0]
print(f" 半衰期为{x_2_value}")
```

4. 结果分析

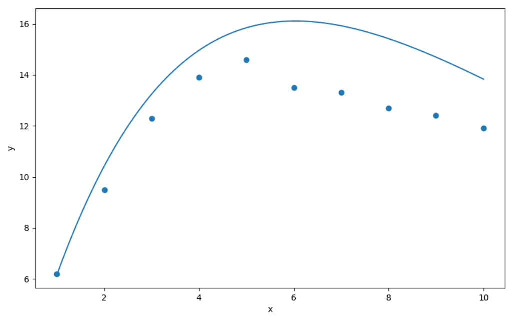

图 3: 最小二乘拟合

$$c_1 = 1.98226312 c_2 = -0.16578157$$

最大值时 t 大约为: 6.03203010559082
最大值时 W 大约为: 16.10850379
有效期为 0.6096298142605157 4.031463144680515 至 22.329978241837967 4.031463144680695
半衰期为 16.155879045706158

四、矩阵多项式的简便运算

用 $p_6(A)$ 表示 6 次矩阵多项式

$$p_6(A) = b_6A^6 + b_5A^5 + b_4A^4 + b_3A^3 + b_2A^2 + b_1A + b_0I.$$

请写出一个计算 $p_6(A)$ 的通用程序，要求矩阵乘法尽可能少（最好只用 3 次矩阵乘法），并写出必要的理论推导过程。

1. 题目分析与思路

用 $M = A^2$ 代入减少计算复杂度

2. 数值计算方法

令原式 =

$$P_6(A) = A^2\{A^2[b_6A^2 + b_5A + b_4] + b_3A + b_2\} + b_1A + b_0I \tag{7}$$

3. Python 代码

```python
'''用表示 6 次矩阵多项式

请写出一个计算的通用程序，要求矩阵乘法尽可能少（最好只用 3 次矩阵乘法），并写出必要的理论推导过程。'''
import numpy as np

A=np.array([[1,2,3,4,5,6],[1,2,3,4,5,6],[1,2,3,4,5,6],[1,2,3,4,5,6],[1,2,3,4,5,6],[1,2,3,4,5,6]])

b=np.zeros(7)
b=[1,1,1,1,1,1,1]
M=A@A
P=M@(M@(b[6]*A+b[5])+b[4]*A+b[3])+b[1]*A+b[0]

print(P)
```

4. 结果分析

例如

$$\mathbf{b} = [1,1,1,1,1,1,1] \; A = \begin{bmatrix} 1,2,3,4,5,6 \\ 1,2,3,4,5,6 \\ 1,2,3,4,5,6 \\ 1,2,3,4,5,6 \\ 1,2,3,4,5,6 \\ 1,2,3,4,5,6 \end{bmatrix} \tag{8}$$

时，运行结果为

$$\begin{bmatrix} 389846,584769,779692,974615,1169538,1364461 \\ 389846,584769,779692,974615,1169538,1364461 \\ 389846,584769,779692,974615,1169538,1364461 \\ 389846,584769,779692,974615,1169538,1364461 \\ 389846,584769,779692,974615,1169538,1364461 \\ 389846,584769,779692,974615,1169538,1364461 \end{bmatrix} \tag{9}$$

用了 3 次矩阵乘法，

第一次计算

$$A^2$$

第二次计算

$$A^2[b_6A^2 + b_5A + b_4]$$

第三次计算

$$A^2\{A^2[b_6A^2 + b_5A + b_4] + b_3A + b_2\}$$

五、必读材料读后感

初次接触《数值计算方法训练》这门课程的大型作业题时，我的心中是有些忐忑的。这种难度的作业不仅是对我学习成果的一次检验，也是对我个人能力和思维方式的挑战。

作业题涉及了多种数值计算方法和技巧，要求我利用 Python 编程语言实现复杂的数值计算任务。在解题过程中，我深感 Python 的强大与灵活，它提供了一个高效的工具，使我能够更加专注于问题的本质和数值计算的逻辑。

解题的过程并非一帆风顺。我遇到了许多难题和挑战，如算法的选择、代码的调试和优化等。正是这些困难促使我不断地思考和尝试，让我更加深入地理解了数值计算方法的原理和应用。每当我成功解决一个问题时，我都会感到一种由衷的喜悦和成就感。

而且运用一个新的编写模式也会遇见很多困难和新的技能，如 latex 的编译和排版特别是公式的输入。摸索中完成这份作业给我带来了许多锻炼，完成后产生了发自内心的成就感。

最后通过这次大型作业的训练，我不仅巩固了已学的数值计算方法，还掌握了一些新的技巧和工具。