数值分析上机报告

XXXXXX　XX

第一章：

17.（上机题）舍入误差与有效数

设 $S_N=\sum_{j=2}^{N}\frac{1}{j^2-1}$，其精确值为 $\frac{1}{2}\left(\frac{3}{2}-\frac{1}{N}-\frac{1}{N+1}\right)$。

（1）编制按从大到小的顺序 $S_N=\frac{1}{2^2-1}+\frac{1}{3^2-1}+\cdots+\frac{1}{N^2-1}$，计算 S_N 的通用程序；

（2）编制按从小到大的顺序 $S_N=\frac{1}{N^2-1}+\frac{1}{(N-1)^2-1}+\cdots+\frac{1}{2^2-1}$，计算 S_N 的通用程序；

（3）按两种顺序分别计算 S_{10^2}，S_{10^4}，S_{10^6}，并指出有效位数（编制程序时用单精度）；

（4）通过本上机题你明白了什么？

（1），（2）计算程序：

```
// C++
// Created by yangtan on 2022/9/8.
//
#include "stdio.h"
#include "math.h"
#include "iostream"
#include "iomanip"

using namespace std;

class solution{
public:
    void compute(int N){
        //compute baseline
        float res = 0;
        res = 0.5 * (1.5 - 1.0/N - 1.0/(N+1));
        cout << "Baseline: \t" << fixed << setprecision(7)  << res << endl;

        // compute in ascent
        res = 0;
        for (int i=2; i<= N; i++){
            res += 1.0 / (pow(i, 2)-1);
        }
        cout << "Ascent: \t" << fixed << setprecision(7) <<res << endl;

        // compute in descent
        res = 0;
        for (int i=N; i >= 2; i--){
            res += 1.0 / (pow(i, 2)-1);
        }
        cout << "Descent: \t" << fixed << setprecision(7) << res << "\n" <<endl;
    }
};


int main(){
    int N[3] = {100, 10000,1000000};
    for (int j = 0; j<=2; j++){
        cout << "start" << " epoch " << j+1 << endl;
        solution().compute(N[j]);
    }
```

```
    return 0;
}
```

（3）运行结果整理如下：

	S_{10^2}	S_{10^4}	S_{10^6}
精确值	0.7400495	0.7499000	0.7499990
从小到大（有效数字位数）	0.7400495 (7 位)	0.7498521 (4 位)	0.7498521 (4 位)
从大到小（有效数字位数）	0.7400495 (7 位)	0.7499000 (7 位)	0.7499990 (7 位)

（4）在迭代次数有限时，从小到大累加与从大到小累加的误差均在可控范围内；随着迭代次数增加，从小到大的计算方式累积误差逐渐增大，其结果的有效数字位数减少。

第二章：

20. （上机题）**Newton 迭代法**

（1）给定初值 x_0 及容许误差 ε，编制 Newton 法解方程 $f(x)=0$ 根的通用程序。

（2）给定方程 $f(x)=x^3/3-x=0$，易知其有三个根 $x_1^*=-\sqrt{3}, x_2^*=0, x_3^*=\sqrt{3}$。

① 由 Newton 方法的局部收敛性可知存在 $\delta>0$，当 $x_0\in(-\delta,\delta)$ 时 Newton 迭代序列收敛于根 x_2^*，试确定尽可能大的 δ；

② 试取若干初始值，观察当 $x_0\in(-\infty,-1),(-1,-\delta),(-\delta,\delta),(\delta,1),(1,+\infty)$ 时 Newton 序列是否收敛以及收敛于哪一个根。

（3）通过本上机题，你明白了什么？

（1）通用程序

```
// C++
// Created by yangtan on 2022/9/18.

#include <iostream>
#include <math.h>
#include <vector>
#include <iomanip>

using namespace std;
class Solution{
public:
    double fun(double x){
        return pow(x, 3)/3 - x;
        // return sin(x);
    }

    double autograd(double x, double step){
        return (fun(x) - fun(x-step))/step;
    }

    double grad(double x, double step){
        return (pow(x, 2) - 1);
    }

    vector<double> find_ranges(int start, int end, double step){
        vector <double> ranges;
        for (double i=start; i<=end; i+=step){
            if (fun(i) * fun(i+step) < 0){
                ranges.push_back(i);
                ranges.push_back(i+step);
            }
        }
        return ranges;
    }

    double iterate(double x0, int maxiter, double epsilon, double grad_step=0.02){
        double x1;
        for (int i=0; i<= maxiter; i++){
            x1 = x0 - fun(x0) / grad(x0, grad_step);
            if (i%maxiter == 0) {
                //cout << "Epoch " << i + 1 << ",  x: "  << setprecision(10) << x1 << endl;
            }
            if (abs(x1 - x0) <= epsilon){
                break;
            }
            x0 = x1;
        }
        return x1;
    }
};
```

```cpp
int main() {
    // Init the solution
    double x, x0, search_step = 0.02, grad_step = 0.02;
    int maxiter = 50;
    int start = -10, end = 10;
    double epsilon = pow(10, -8);
    double delta;
    double r;
    vector<double> ans;

    //零点定理寻找零点所在区间
    vector<double> ranges = Solution().find_ranges(start, end, search_step);
    int N = ranges.size()/2; //区间的总对数

    // Start solving the equation
    for (int i=0; i<N; i++){
        cout << "----------------------------" << endl;
        cout << "Start solving "<< i << "-th solution." << endl;
        x0 = (ranges[2*i] + ranges[2*i+1])/2;
        x = Solution().iterate(x0, maxiter, epsilon, grad_step);
        ans.push_back(x);
    }

    // Output the results
    cout << "-------------------------\n" << "Solutions for equation are \n" ;
    for (auto j=ans.begin(); j != ans.end(); j++){
        cout << setprecision(10) << *j << endl;
    }

    // compute the delta

    cout << "-------------------------\n" << "Starting computing delta ....\n" ;
    double delta_start = -0.9, delta_end = 0.9;
    for (double x=delta_start; x<=delta_end; x+=0.01){
        r = Solution().iterate(x, maxiter, epsilon);
        cout << "delta: "<< x << "\t  r: " << r << endl;
        //if (abs(r) >= pow(10,-4)){
        //    break;
        //}
        //if (abs(r) <= pow(10,-4)){
        //    break;
        //}
    }

    cout << "Completed." << endl;

    return 0;
}
```

（2）

① 由运行结果得 δ 最大为 0.7745 。

②

区间	$(-\infty, -1)$	$(-1, -\delta)$	$(-1, -\delta)$	$(-\delta, \delta)$	$(-\delta, \delta)$
选取初值	−1.5	−0.8	−0.7791	−0.75	0.5
收敛值	−1.732050808	−1.732050808	1.732050808	0	0
区间	$(\delta, 1)$	$(\delta, 1)$	$(1, +\infty)$		
选取初值	0.7791	0.785	2.5		
收敛值	−1.732050808	1.732050808	1.732050808		

（3）牛顿迭代法不同初值的选取会导致算法收敛于不同的根，如果可以，在使用牛顿迭代法时应首先确定某非线性方程组的根的数量，并尽可能确定其区间分布，从而达到求解的快速收敛并获得较高的精度。

第三章

39.（上机题）列主元 Gauss 消去法

对于某电路的分析，归结为求解线性方程组 $\boldsymbol{RI}=\boldsymbol{V}$，其中

$$\boldsymbol{R}=\begin{bmatrix} 31 & -13 & 0 & 0 & 0 & -10 & 0 & 0 & 0 \\ -13 & 35 & -9 & 0 & -11 & 0 & 0 & 0 & 0 \\ 0 & -9 & 31 & -10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -10 & 79 & -30 & 0 & 0 & 0 & -9 \\ 0 & 0 & 0 & -30 & 57 & -7 & 0 & -5 & 0 \\ 0 & 0 & 0 & 0 & -7 & 47 & -30 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -30 & 41 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5 & 0 & 0 & 27 & -2 \\ 0 & 0 & 0 & -9 & 0 & 0 & 0 & -2 & 29 \end{bmatrix}$$

$$\boldsymbol{V}^{\mathrm{T}}=(-15,27,-23,0,-20,12,-7,7,10)^{\mathrm{T}}$$

(1) 编制解 n 阶线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 的列主元 Gauss 消去法的通用程序；

(2) 用所编程序解线性方程组 $\boldsymbol{RI}=\boldsymbol{V}$，并打印出解向量，保留 5 位有效数字；

(3) 本题编程之中，你提高了哪些编程能力？

（1）通用程序

```
# Python
# Created by yangtan on 2022/10/18.
#

import copy
import matplotlib.pyplot as plt
import numpy as np


def Solve(AugmentedMatrix):
    col = AugmentedMatrix.shape[1]  # 增广矩阵列数
    # 消元
    for i in range(col - 2):
        current_column = AugmentedMatrix[i:, i]
        max_index = np.argmax(current_column) + i  # 寻找最大元
        if (AugmentedMatrix[max_index, i] == 0):
            print("无唯一解")
            return
        tempA = AugmentedMatrix[[i, max_index], :].copy()
        AugmentedMatrix[[i, max_index], :] = AugmentedMatrix[[max_index, i], :]  # 交换
        AugmentedMatrix[[max_index, i], :] = tempA
        l = AugmentedMatrix[i + 1:, i] / AugmentedMatrix[i, i]  # 计算系数
        m = np.tile(AugmentedMatrix[i, :], (l.shape[0], 1)) * np.tile(l, (col, 1)).T  #
计算消元时减去的矩阵
        AugmentedMatrix[i + 1:, :] = AugmentedMatrix[i + 1:, :] - m  # 消元
    if (AugmentedMatrix[col - 2, col - 2] == 0):
        print("无唯一解")
        return
    # 代入
    x = np.zeros(col - 1)
    for i in range(col - 2, -1, -1):
        x[i] = (AugmentedMatrix[i, -1] - np.dot(AugmentedMatrix[i, :-1], x.T)) /
AugmentedMatrix[i, i]
    return x


A = np.array([[31, -13, 0, 0, 0, -10, 0, 0, 0],
              [-13, 35, -9, 0, -11, 0, 0, 0, 0],
              [0, -9, 31, -10, 0, 0, 0, 0, 0],
              [0, 0, -10, 79, -30, 0, 0, 0, -9],
              [0, 0, 0, -30, 57, -7, 0, -5, 0],
              [0, 0, 0, 0, -7, 47, -30, 0, 0],
              [0, 0, 0, 0, 0, -30, 41, 0, 0],
```

```python
             [0, 0, 0, 0, -5, 0, 0, 27, -2],
             [0, 0, 0, -9, 0, 0, 0, -2, 29]])
A = A.astype(np.float64)
b_ = np.array([-15, 27, -23, 0, -20, 12, -7, 7, 10])
b = b_[:, np.newaxis]
# 增广矩阵
A_ = np.concatenate((A, b), 1)

x = Solve(A_)
print(np.around(x, 5).tolist())
```

（2）经编程求解，$RI = V$ 的解向量为：

```
[-0.28923, 0.34544, -0.71281, -0.22061, -0.4304, 0.15431, -0.05782, 0.20105, 0.29023]
```

（3）提高了在代码中灵活使用矩阵计算，而非对元素进行多重 for 循环的能力。经过试验，在编程过程中使用矩阵运算不仅可以增加程序的可读性，还可以将运算以并行计算的形式进行，大大提高程序的运行效率。

第四章

37. (上机题)3 次样条插值函数

(1) 编制求第一型 3 次样条插值函数的通用程序；

(2) 已知汽车门曲线型值点的数据如下：

i	0	1	2	3	4	5
x_i	0	1	2	3	4	5
y_i	2.51	3.30	4.04	4.70	5.22	5.54
i	6	7	8	9	10	
x_i	6	7	8	9	10	
y_i	5.78	5.40	5.57	5.70	5.80	

端点条件为 $y'_0 = 0.8, y'_{10} = 0.2$，用所编程序求车门的 3 次样条插值函数 $S(x)$，并打印出 $S(i+0.5), i = 0,1,\cdots,9$。

(1) 通用程序

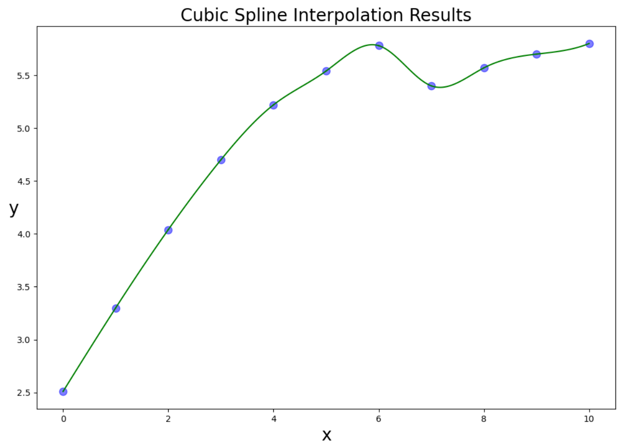
```
# Python
# Created by yangtan on 2022/11/8.
import numpy as np
import matplotlib.pyplot as plt


def difference(x, y, g):
    x_ = np.array(x)
    x_ = np.insert(np.append(x_, x_[-1]), 0, x_[0])
    x = x_[:, np.newaxis]

    y_ = np.array(y)
    y_ = np.insert(np.append(y_, y_[-1]), 0, y_[0])
    y = y_[:, np.newaxis]

    n = x.shape[0]
    t = np.zeros((n, 2))

    table = np.concatenate([x, y, t], 1)
    table[0, 2] = g[0]
    table[n - 2, 2] = g[1]
    for j in range(2, 4):
        for i in range(n - 1):
            if (i == 0 and j == 2) or (i == n - 2 and j == 2):
                continue
            if j == 2:
                table[i, j] = (table[i + 1, j - 1] - table[i, j - 1]) / (table[i + 1, 0]
- table[i, 0])
            else:
                if i >= n - 2:
                    break
                table[i, j] = (table[i + 1, j - 1] - table[i, j - 1]) / (table[i + 2, 0]
- table[i, 0])

    return table[1:n - 1, -2], table[:n - 2, -1]


def spline(X, Y, g):
    n = len(X)
    m = n - 1
    h = np.array([X[i + 1] - X[i] for i in range(n - 1)])

    mu = np.array([h[i] / (h[i + 1] + h[i]) for i in range(h.shape[0] - 1)])
    lam = 1 - mu
```

```python
    mu = np.append(mu, 1)
    lam = np.insert(lam, 0, 1)

    # 差商表
    d1, d2_ = difference(X, Y, g)
    # 获取三转角方程系数矩阵
    d2 = 6 * d2_
    A = 2 * np.eye(n) + np.diag(lam, k=1) + np.diag(mu, k=-1)

    # 解三转角方程
    M = np.linalg.solve(A, d2)

    c = np.zeros((n - 1, 4))
    a, b = c.shape

    for i in range(a):
        c[i, 0] = Y[i]
        c[i, 1] = d1[i] - (1 / 3 * M[i] + 1 / 6 * M[i + 1]) * h[i]
        c[i, 2] = 0.5 * M[i]
        c[i, 3] = 1 / (6 * h[i]) * (M[i + 1] - M[i])

    return np.around(c, 4)

def plot_a_range(x1, x2, c, color):
    x = np.linspace(x1, x2, 30)
    y = c[0] + c[1] * (x - x1) + c[2] * (x - x1) ** 2 + c[3] * (x - x1) ** 3
    plt.plot(x, y, color)


def plot_interp(x, y, c):
    fontsize = 20
    plt.figure(figsize=(12, 8))
    plt.scatter(x, y, color='b', s=300, marker='.', alpha=0.5)

    n = len(x) - 1
    for i in range(n):
        plot_a_range(x[i], x[i + 1], c[i, :], 'g')
    plt.xlabel('x', fontsize=fontsize)
    plt.ylabel('y', fontsize=fontsize, rotation=0)
    plt.title('Cubic Spline Interpolation Results', fontsize=fontsize)
    plt.show()

#确定 x0 区间
def search(x, x0):
    n = len(x)
    for i in range(n - 1):
        if x0 >= x[i] and x0 < x[i + 1]:
            return x[i], x[i + 1], i

#预测
def fit(x0, C):
    x1, x2, index = search(x, x0)
    c = C[index, :]
    return c[0] + c[1] * (x0 - x1) + c[2] * (x0 - x1) ** 2 + c[3] * (x0 - x1) ** 3

# 开始插值,求解各区间上插值系数
x = [i for i in range(11)]
y = [2.51, 3.3, 4.04, 4.70, 5.22, 5.54, 5.78, 5.4, 5.57, 5.70, 5.80]
g = [0.8, 0.2]
c = spline(x, y, g)
print('各区间系数矩阵为：')
print(c)

#画图
plot_interp(x, y, c)

#开始预测
r = []
```

```
for i in range(10):
    x0 = i + 0.5
    y0 = np.around(fit(x0, c), 5)
    r.append(y0)

print(r)
```

运行结果：

```
各区间系数矩阵为:
[[ 2.510e+00  8.000e-01 -1.500e-03 -8.500e-03]
 [ 3.300e+00  7.715e-01 -2.700e-02 -4.500e-03]
 [ 4.040e+00  7.041e-01 -4.040e-02 -3.700e-03]
 [ 4.700e+00  6.123e-01 -5.140e-02 -4.090e-02]
 [ 5.220e+00  3.868e-01 -1.741e-01  1.074e-01]
 [ 5.540e+00  3.606e-01  1.479e-01 -2.685e-01]
 [ 5.780e+00 -1.491e-01 -6.575e-01  4.266e-01]
 [ 5.400e+00 -1.844e-01  6.222e-01 -2.679e-01]
 [ 5.570e+00  2.565e-01 -1.814e-01  5.490e-02]
 [ 5.700e+00  5.840e-02 -1.680e-02  5.840e-02]]
```



（2）

$S(i+0.5)$, $i=0, 1, ..., 9$ 的运算结果如下：

```
[2.90856, 3.67844, 4.38149, 4.98819, 5.3833, 5.72371, 5.5944, 5.42986, 5.65976, 5.7323]
```

第五章

上机题：用 Romberg 求积法计算积分

$$\int_{-1}^{1}\frac{1}{1+100x^2}dx$$

的近似值，要求误差不超过 0.5×10−7.

```
// C++
// Created by yangtan on 2022/11/18.

#include <iostream>
#include <math.h>
#include <algorithm>
#include <vector>
#include <iomanip>

using namespace std;
class Romberg{
public:
    double fun(double x){
        return 1.0/(1+100*pow(x, 2));
        //return sin(x)/x;
    }

    double Tn(double a, double b, int n){
        double h = (b-a) / double(n);
        double x1, x2, t = 0;
        for (int i=0; i<n; i++){
            x1 = a + i*h;
            x2 = a + (i+1)*h;
            t += 0.5*h*(fun(x1)+fun(x2));
        }
        return t;
    }

    double Sn(double t2n, double tn){
        return 4.0/3*t2n - 1.0/3*tn;
    }

    double Cn(double s2n, double sn){
        return 16.0/15*s2n - 1.0/15*sn;
    }

    double Rn(double c2n, double cn){
        return 64.0/63*c2n - 1.0/63*cn;
    }

};

int main(){
    int n=1, a=-1, b=1;
    vector<double> Tn, Sn, Cn, Rn;
    double tn, sn, cn, rn;
    double diff=100;

    tn = Romberg().Tn(a, b, pow(2, 0));
    Tn.push_back(tn);

    while (diff>0.5*pow(10,-7)){
        n++;
        tn = Romberg().Tn(a, b, pow(2, n-1));
        Tn.push_back(tn);

        if(n>=2){
            sn = Romberg().Sn(Tn.at(Tn.size()-1), Tn.at(Tn.size()-2));
```

```
            Sn.push_back(sn);
        }

        if (n>=3){
            cn = Romberg().Cn(Sn.at(Sn.size()-1), Sn.at(Sn.size()-2));
            Cn.push_back(cn);
        }

        if (n>=4){
            rn = Romberg().Rn(Cn.at(Cn.size()-1), Cn.at(Cn.size()-2));
            Rn.push_back(cn);
            cout << "Epoch " << n-3 << ",   R_2n = " << setprecision(8) << rn << endl;
        }

        if (Rn.size()>=2){
            double r1=Rn.at(Rn.size()-1), r2=Rn.at(Rn.size()-2);
            diff = abs(r1 - r2);
        }
    }
    return 0;
}
```

输出结果：

```
Epoch 1,   R_2n = 0.27711804
Epoch 2,   R_2n = 0.28111937
Epoch 3,   R_2n = 0.29385002
Epoch 4,   R_2n = 0.29431614
Epoch 5,   R_2n = 0.29422487
Epoch 6,   R_2n = 0.29422553
Epoch 7,   R_2n = 0.29422553
```

最终积分结果为 0.29422553。

第六章：

23.（上机题）**常微分方程初值问题数值解**

(1) 编制 RK_4 方法的通用程序；

(2) 编制 AB_4 方法的通用程序(由 RK_4 提供初值)；

(3) 编制 AB_4-AM_4 预测校正方法通用程度(由 RK_4 提供初值)；

(4) 编制带改进的 AB_4-AM_4 预测校正方法通用程序(由 RK_4 提供初值)；

(5) 对于初值问题

$$\begin{cases} y' = -x^2y^2 & (0 \leqslant x \leqslant 1.5), \\ y(0) = 3 \end{cases}$$

取步长 $h = 0.1$,应用(1)～(4)中的四种方法进行计算,并将计算结果和精确解 $y(x) = 3/(1+x^3)$ 作比较;

(6) 通过本上机题,你能得到哪些结论?

（1）~（4）通用程序：

```
//C++
//Created by yangtan on 2022/12/7

#include <iostream>
#include <iomanip>
#include <cmath>
#include <vector>

using namespace std;
class Solution{
public:
    void print_all(vector<double> V1, vector<double> V2,vector<double> V3,vector<double>
V4,vector<double> V5,
                  double x0, double h){
        cout << "                    RK4                 AB4                 AB4-AM4
"
              "Improved-AB4-AM4            精确解" << endl;
        for (int i=0; i<V1.size(); i++){
            if (i==0){
                cout << "x = " << setprecision(2) << x0+i*h
                     << ";  \t  y = "<<setprecision(10) << V1[i] << " "
                     << ";  \t\t\t\t  y = "<<setprecision(10) << V2[i] << " "
                     << ";  \t\t\t\t  y = "<<setprecision(10) << V3[i] << " "
                     << ";  \t\t\t\t  y = "<<setprecision(10) << V4[i] << " "
                     << ";  \t\t\t\t  y = "<<setprecision(10) << V5[i] << " " << endl;
            }
            else{
                cout << "x = " << setprecision(2) << x0+i*h
                     << ";  \t  y = "<<setprecision(10) << V1[i] << " "
                     << ";  \t  y = "<<setprecision(10) << V2[i] << " "
                     << ";  \t  y = "<<setprecision(10) << V3[i] << " "
                     << ";  \t  y = "<<setprecision(10) << V4[i] << " "
                     << ";  \t  y = "<<setprecision(10) << V5[i] << " " << endl;
            }
        }
    }

    void print_vector(vector<double> V, double x0, double h){
        for (int i=0; i<V.size(); i++){
            cout << "x = " << setprecision(2) << x0+i*h
            << ";  \t  y = "<<setprecision(10) << V[i] << "  " << endl;
        }
    }

    double fun(double x, double u){
        //return 2.0/x*u + pow(x,2)*exp(x);
        return -pow(x,2)*pow(u,2);
    }

    double f(double x){
```

```
        return 3.0/(1+pow(x,3));
    }

    vector<double> rk4(double x0, double u0, double h, double xup){
        vector<double> u={u0};
        double k1,k2,k3,k4;
        while(x0<=xup){
            k1 = fun(x0, u0);
            k2 = fun(x0+0.5*h, u0+0.5*h*k1);
            k3 = fun(x0+0.5*h, u0+0.5*h*k2);
            k4 = fun(x0+h, u0+h*k3);
            u0 = u0 + h/6*(k1+2*k2+2*k3+k4);
            x0 += h;
            u.push_back(u0);
        }
        return u;
    }

    vector<double> ab4(vector<double> X, vector<double> U, double h, double xup){
        vector<double> u={U[0], U[1], U[2], U[3]};
        double tu3;
        while (X[3]<=xup){
            tu3 = U[3] + h/24*(55*fun(X[3],U[3]) - 59*fun(X[2],U[2]) + 37*fun(X[1],U[1]) -
9*fun(X[0],U[0]));
            u.push_back(tu3);
            X[0]+=h; X[1]+=h; X[2]+=h; X[3]+=h;
            U[0]=U[1]; U[1]=U[2]; U[2]=U[3]; U[3]=tu3;
        }
        return u;
    }

    vector<double> ab4_am4(vector<double> X, vector<double> U, double h, double xup){
        vector<double> u={U[0], U[1], U[2], U[3]};
        double y_p;
        while (X[3]<=xup){
            y_p = U[3] + h/24*(55*fun(X[3],U[3]) - 59*fun(X[2],U[2]) + 37*fun(X[1],U[1]) -
9*fun(X[0],U[0]));
            y_p = U[3] + h/24*(9*fun(X[3]+h,y_p) + 19*fun(X[3],U[3]) - 5*fun(X[2],U[2]) +
fun(X[1],U[1]));
            u.push_back(y_p);
            X[0]+=h; X[1]+=h; X[2]+=h; X[3]+=h;
            U[0]=U[1]; U[1]=U[2]; U[2]=U[3]; U[3]=y_p;
        }
        return u;
    }

    vector<double> improved_ab4am4(vector<double> X, vector<double> U, double h, double
xup){
        vector<double> u={U[0], U[1], U[2], U[3]};
        double y_p, y_c, y;
        while (X[3]<=xup){
            y_p = U[3] + h/24*(55*fun(X[3],U[3]) - 59*fun(X[2],U[2]) + 37*fun(X[1],U[1]) -
9*fun(X[0],U[0]));
            y_c = U[3] + h/24*(9*fun(X[3]+h,y_p) + 19*fun(X[3],U[3]) - 5*fun(X[2],U[2]) +
fun(X[1],U[1]));
            y = 251.0/270*y_c + 19.0/270*y_p;
            u.push_back(y);
            X[0]+=h; X[1]+=h; X[2]+=h; X[3]+=h;
            U[0]=U[1]; U[1]=U[2]; U[2]=U[3]; U[3]=y;
        }
        return u;
    }

    vector<double> compute(double x0, double h, double xup){
        vector<double> ut;
        for (double x=x0; x<xup+h; x+=h){
            ut.push_back(f(x));
        }
        return ut;
```

```
    }

};

int main() {
    double u0=3;
    double h=0.1;
    double xup=1.5;
    double x0=0.0;

    vector<double> x0_ab4 = {x0, x0+h, x0+2*h, x0+3*h};
    vector<double> x0_ab4_am4 = {x0, x0+h, x0+2*h, x0+3*h};
    vector<double> x0_ipv = {x0, x0+h, x0+2*h, x0+3*h};
    vector<double> u_rk4, u_ab4, u_ab4am4, u_ipv, ut;

    cout << "------------RK4 算法求解结果-------------" << endl;
    u_rk4 = Solution().rk4(x0, u0, h, xup);
    Solution().print_vector(u_rk4, x0, h);

    cout << "\n------------AB4 算法求解结果-------------" << endl;
    vector<double> u_f4(u_rk4.begin(), u_rk4.begin()+4);
    u_ab4 = Solution().ab4(x0_ab4, u_f4, h, xup);
    Solution().print_vector(u_ab4, x0, h);

    cout << "\n----------AB4-AM4 算法求解结果-----------" << endl;
    u_ab4am4 = Solution().ab4_am4(x0_ab4_am4, u_f4, h, xup);
    Solution().print_vector(u_ab4am4, x0, h);

    cout << "\n---------加速 AB4-AM4 算法求解结果---------" << endl;
    u_ipv = Solution().improved_ab4am4(x0_ipv, u_f4, h, xup);
    Solution().print_vector(u_ipv, x0, h);

    cout << "\n-----------精确解------------" << endl;
    ut = Solution().compute(x0, h, xup);
    Solution().print_vector(ut, x0, h);

    cout << "\n                        -------------------------综上，上述算法求解结果--------
-----------------" << endl;
    Solution().print_all(u_rk4, u_ab4, u_ab4am4, u_ipv, ut, x0, h);

    return 0;
}
```

（5）程序运行结果如下：

```
                        ------------------------综上，上述算法求解结果------------------------
                 RK4                    AB4                    AB4-AM4                Improved-AB4-AM4       精确解
x = 0;       y = 3 ;                y = 3 ;                y = 3 ;                y = 3 ;                y = 3
x = 0.1;     y = 2.99700281 ;       y = 2.99700281 ;       y = 2.99700281 ;       y = 2.99700281 ;       y = 2.997002997
x = 0.2;     y = 2.976190085 ;      y = 2.976190085 ;      y = 2.976190085 ;      y = 2.976190085 ;      y = 2.976190476
x = 0.3;     y = 2.921128745 ;      y = 2.921128745 ;      y = 2.921128745 ;      y = 2.921128745 ;      y = 2.921129503
x = 0.4;     y = 2.819547261 ;      y = 2.818389261 ;      y = 2.819678433 ;      y = 2.819587713 ;      y = 2.819548872
x = 0.5;     y = 2.666663489 ;      y = 2.664672471 ;      y = 2.666875983 ;      y = 2.666712854 ;      y = 2.666666667
x = 0.6;     y = 2.467100258 ;      y = 2.46520263 ;       y = 2.467251764 ;      y = 2.467097029 ;      y = 2.467105263
x = 0.7;     y = 2.233799142 ;      y = 2.233078952 ;      y = 2.233731415 ;      y = 2.233682492 ;      y = 2.233804914
x = 0.8;     y = 1.984122855 ;      y = 1.984950578 ;      y = 1.983786704 ;      y = 1.983884685 ;      y = 1.984126984
x = 0.9;     y = 1.735107114 ;      y = 1.737043286 ;      y = 1.734607435 ;      y = 1.734808011 ;      y = 1.735106998
x = 1;       y = 1.500005807 ;      y = 1.502194547 ;      y = 1.499515936 ;      y = 1.499731907 ;      y = 1.5
x = 1.1;     y = 1.287012594 ;      y = 1.288763445 ;      y = 1.286657139 ;      y = 1.286820676 ;      y = 1.287001287
x = 1.2;     y = 1.099722169 ;      y = 1.100724202 ;      y = 1.099533146 ;      y = 1.09962178 ;       y = 1.099706745
x = 1.3;     y = 0.9383974582 ;     y = 0.9387104961 ;     y = 0.9383425229 ;     y = 0.938367316 ;      y = 0.938379731
x = 1.4;     y = 0.8013004266 ;     y = 0.8011349485 ;     y = 0.8013273743 ;     y = 0.8013113543 ;     y = 0.8012820513
x = 1.5;     y = 0.6857320857 ;     y = 0.6853345757 ;     y = 0.6857961111 ;     y = 0.6857604461 ;     y = 0.6857142857
```

（6）感想