实验报告

一、实验目的

用下面方法求方程 $e^x+10x-2=0$ 的根，要求误差不超过 0.5×10^{-3}：

（1） 在区间 $[0,1]$ 上使用二分法；

（2） 取初值 $x_0=0$ 并使用简单迭代法，迭代函数为 $x_{k+1}=\dfrac{2-e^{x_k}}{10}$。

二、实验方法（要求用自己的语言描述算法）

（1）二分法的求解步骤

```
赋初值：left := 0; right := 1; precison := 0.0005; f := (x) => exp(x)
+ 10.0 * x - 2.0;
初始化：计算 left_val := f(left); right_val := f(right);
循环：
    计算 mid := (left + right)/2; mid_val := f(mid)
    如果 right - left < 2 * precison，则循环结束，最终结果为 mid
    否则
        如果 left_val * mid_val > 0，则说明函数零点在右半侧，赋值 left :=
 mid，left_val := mid_val 继续循环计算。
        如果 right_val * mid_val > 0，则说明函数零点在左半侧，赋值
  right := mid，right_val := mid_val 继续循环结算。
        如果 mid_val = 0，则循环结束，最终结果为 mid。
```

(2)简单迭代法的求解步骤

```
赋初值：start := 0; l := e/10; precision := 0.0005;
g := (x) => (2.0 - exp(x)) / 10.0;
初始化：c := start;
循环：
    计算 _sen := g(c)
    如果 l / (1-l) * (_sen - c) <= precision，则循环结束，结果为_sen
    否则 赋值 c := _sen，循环继续。
```

说明:在此例中

$$|g'(x)|=\frac{e^x}{10}\le\frac{e}{10}$$

所以，在简单迭代法求解步骤中，l 取 $\frac{e}{10}$。

三、实验代码

（1）C++代码

class::tools tools.h, tools.cpp

```
#pragma once
#include <iostream>
#include <vector>
#include "point_d.h"

using namespace std;

class tools {
public:
    static void print_a(const vector<point_d>& results);
    static void print_script(const vector<point_d>& results);
};
```

```
#include "tools.h"

void tools::print_a(const vector<point_d>& results) {
    for(point_d p: results){
        cout << p << endl;
    }
}

void tools::print_script(const vector<point_d>& results) {
    cout << "> print scripts" << endl;
    cout << "x = [";
    int _size = results.size();
    for(int i = 0; i < _size; ++i){
        cout << i+1 << " ";
    }
    cout << "];" << endl << "y = [";
    for(point_d p: results){
        cout << p.x << " ";
    }
    cout << "];" << endl;
}
```

class::div_iter div_iter.h div_iter.cpp(求解二分法的核心文件)

```
#include <iostream>
#include <vector>
#include "point_d.h"
```

```cpp
#include "pd.h"

using namespace std;

class div_iter {
public:
    explicit div_iter(double left = 0, double right = 1, double
precision = 0.0005, func_d f = nullptr):
        left(left), right(right), precision(precision),f(f) {
        left_val = f(left);
        right_val = f(right);
    }

    void solve();
    vector<point_d> results;
private:
    func_d f;

    double precision;
    double left;
    double right;
    double left_val;
    double right_val;
};
```

```cpp
#include "div_iter.h"

void div_iter::solve() {
    while (true) {
        double _mid = (left + right) / 2.0;
        double _mid_val = f(_mid);
        results.emplace_back(_mid, _mid_val);
        if (abs(right - left) < 2 * precision) {
            break;
        }
        if ((left_val < 0 && _mid_val < 0) || (right_val < 0 &&
_mid_val > 0)) {
            left = _mid;
            left_val = _mid_val;
        } else if((left_val < 0 && _mid_val > 0) || (right_val < 0 &&
_mid_val < 0)) {
            right = _mid;
            right_val = _mid_val;
```

```
        } else if(_mid_val == 0){
            break;
        }
    }
}
```

class::simple_iter simple_iter.h simple_iter.cpp(求解简单迭代法的核心文件)

```
#pragma once
#include <iostream>
#include <vector>
#include "point_d.h"
#include "pd.h"

using namespace std;

class simple_iter {
public:
    simple_iter(double start, double l, double precision, func_d f):
        c(start), l(l), precision(precision), f(f) {
    }

    void solve();
    vector<point_d> results;
private:
    double c;
    double l;
    double precision;
    func_d f;
};
```

```
#include "simple_iter.h"

void simple_iter::solve() {
    while(true){
        double _sen = f(c);
        results.emplace_back(c, 0);
        if(l / (1-l) * abs(c - _sen) <= precision){
            break;
        }
        c = _sen;
    }
```

```
}
```

main.cpp 文件

```
#include <iostream>
#include <queue>
#include <cmath>
#include "point_d.h"
#include "div_iter.h"
#include "tools.h"
#include "simple_iter.h"

double f(double x){
    return exp(x) + 10.0 * x - 2.0;
}

double g(double x){
    return (2.0 - exp(x)) / 10.0;
}

using namespace std;
int main() {
    div_iter iter = div_iter(0,1,0.0005,f);
    iter.solve();
    //tools::print_a(iter.results);
    tools::print_script(iter.results);

    simple_iter iter1 = simple_iter(0, exp(1)/ 10.0, 0.0005, g);
    iter1.solve();
    //tools::print_a(iter1.results);
    tools::print_script(iter1.results);

    return 0;
}
```

（2）matlab 脚本文件

div_iter.m

```
clear
clc


figure()
```

```
x = [1 2 3 4 5 6 7 8 9 10 11 ];
y = [0.5 0.25 0.125 0.0625 0.09375 0.078125 0.0859375 0.0898438
0.0917969 0.0908203 0.090332 ];
plot(x,y,'-.r*')
hold on
z = [0 0.1 0.0894829 0.0894829 0.0894829 0.0894829 0.0894829
0.0894829 0.0894829 0.0894829 0.0894829];
plot(x,z,'-b.')
```

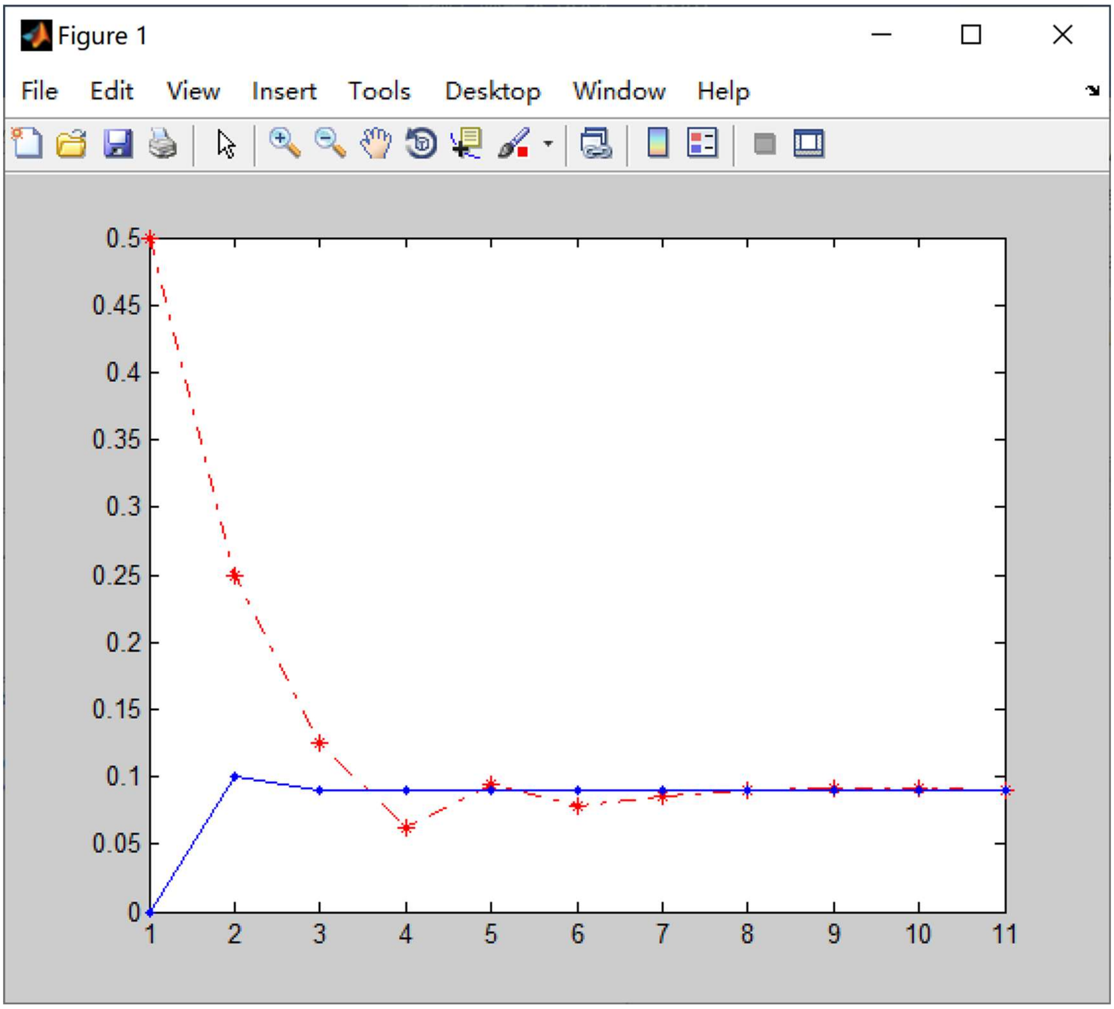
四、实验结果及其讨论

（1）C++代码运行结果

```
Run:    ProjCal
C:\Users\t1542\Program\C++\ProjCal\cmake-build-debug\ProjCal.exe
> print scripts
x = [1 2 3 4 5 6 7 8 9 10 11 ];
y = [0.5 0.25 0.125 0.0625 0.09375 0.078125 0.0859375 0.0898438 0.0917969 0.0908203 0.090332 ];
> print scripts
x = [1 2 3 ];
y = [0 0.1 0.0894829 ];

Process finished with exit code 0
```

（2）matlab 脚本运行结果



五、总结

本次实验使用 C++进行迭代求解，使用 matlab 进行画图。从实验知道，近似解为 0.090332（二分），0.0894829（简单迭代），解符合精度要求。二分法用了 11 次才达到精度，而简单迭代用了 3 次。分析主要原因如下：

①简单迭代的初值 0 非常接近 0.1

②使用的简单迭代函数比较平缓，有利于快速逼近符合要求的近似解。

二分法法求解比较稳定，而要让简单迭代函数更快需要设计比较好的函数（不仅要满足收敛的条件，曲线也应该尽可能平缓）