常微分方程 (组) 初值问题数值解法

汪杰
10185102223

华东师范大学

2021 年 1 月 1 日

目录

目录

成果演示

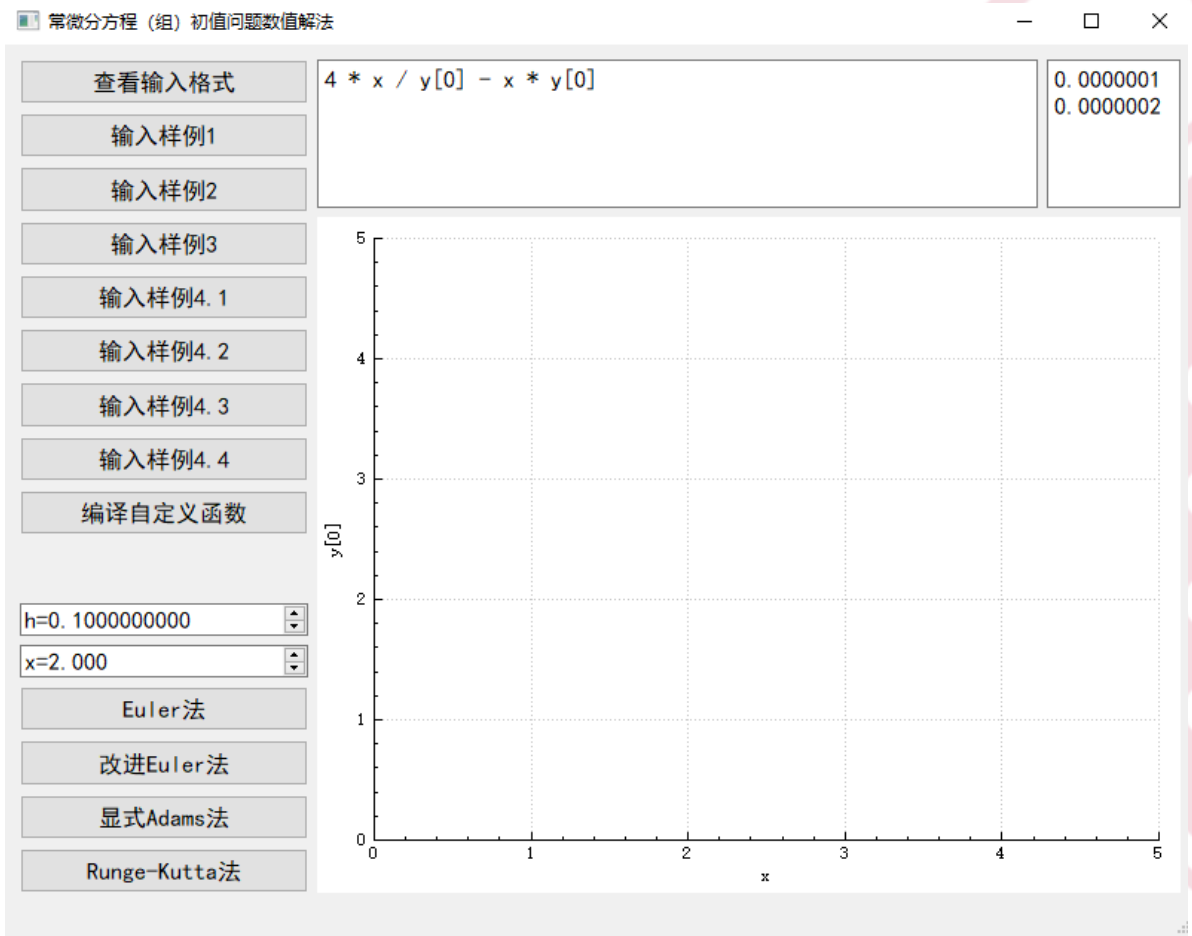
常微分方程（组）初值问题数值解法
查看输入格式
输入样例1
输入样例2
输入样例3
输入样例4.1
输入样例4.2
输入样例4.3
输入样例4.4
编译自定义函数
h=0.1000000000
x=2.000
Euler法
改进Euler法
显式Adams法
Runge-Kutta法
4 * x / y[0] - x * y[0]
0.0000001
0.0000002
y[0]
x
0
1
2
3
4
5

图: UI 界面

目录

向量函数

- 数学基础

$$\begin{cases} y_1' = f_1(x, y_1, y_2, \ldots, y_n) \\ y_2' = f_2(x, y_1, y_2, \ldots, y_n) \\ \ldots \\ y_n' = f_n(x, y_1, y_2, \ldots, y_n) \end{cases} \Longrightarrow \boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y})$$

$$\boldsymbol{f}: \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$$

- 代码

```
typedef double real_type;
typedef std::vector<real_type> ndarray;
typedef ndarray (*func_t)(real_type, const ndarray&);
```

应用举例

- 函数代码

```
ndarray f(real_type x, const ndarray &y) {
    ndarray rst(y.size());
    rst[0] = y[1];
    rst[1] = y[0] / (exp(x) + 1);
    return rst;
}
```

- 对应的常微分方程 (组)

$$\begin{cases} y_0' = y_1 \\ y_1' = \dfrac{y_0}{e^x + 1} \end{cases} \Longrightarrow y'' = \frac{y}{e^x + 1}$$

实现 ODE 迭代

- 欧拉迭代函数（参数类型略）：

```
ndarray EulerIter(func, x, y, h) {
    return y + h * func(x, y);
}
```

- 四阶龙格库塔法迭代函数：

```
ndarray RK4Iter(func_t func,
                real_type x, const ndarray &y,
                real_type h) {
    ndarray k1 = func(x, y);
    ndarray k2 = func(x + h / 2, y + h / 2 * k1);
    ndarray k3 = func(x + h / 2, y + h / 2 * k2);
    ndarray k4 = func(x + h, y + h * k3);
    return y + h / 6 * (k1 + 2 * k2 + 2 * k3 + k4);
}
```

目录

通用 UI 界面

- 支持任意用户输入
- 几种可能的实现方式：
  1. 函数解释器：字符串处理
  2. 调用编译器：动态编译、链接
  3. 人生苦短，我用 Python：eval() 函数

动态编译流程

1. 输入表达式
2. 扩展为 C++ 源代码
3. 创建子进程编译：源代码 $\rightarrow$ 动态链接库 (*.dll)
4. 链接动态链接库，获取其中的函数指针
5. 回到 ODE 数值计算

扩展——核心代码

```
static const QString header(
    "#include <vector>\n"
    "#include <cmath>\n"
    "using namespace std;\n"
    "typedef vector<double> ndarray;\n"
    "\n"
);
static const QString func_head(
    "__declspec (dllexport) ndarray func(double x, const ndarray &y) {\n"
    "    ndarray rst(y.size());\n"
);
static const QString func_tail(
    "    return rst;\n"
    "}\n"
);
QTextStream out(&file);
out << header << func_head;
for (int i = 0; i < func_code.size(); ++i)
    out << QString("    rst[%1] = ").arg(i) << func_code[i] << ";\n";
out << func_tail;
file.close();
```

输入与扩展——要点

- 输入时要符合 C++ 语法
- 扩展时要符合 Windows 下的 DLL 格式要求
  - 如果是 Linux，*.so 的代码和普通代码一样，无需修改

命令行编译

- 编译普通可执行文件：

```
g++ main.cpp -o main
```

- 编译动态链接库 *.dll（Windows）

```
g++ -c dllmain.cpp
g++ -shared -o dllmain.dll dllmain.o
```

- 编译共享目标文件 *.so（Linux）

```
g++ -fpic -c dllmain.cpp
g++ -shared -o dllmain.so dllmain.o
```

相关函数接口

功能	Windows API	Linux 系统调用
创建子进程 执行编译器	CreateProcess()	fork() exec*()
等待子进程结束	WaitForSingleObject()	wait()
获得子进程返回值	GetExitCodeProcess()	wait()
关闭子进程句柄	CloseHandle()	-
加载 *.dll/*.so	LoadLibrary()	dlopen()
获取函数指针	GetProcAddress()	dlsym()
关闭 *.dll/*.so	FreeLibrary()	dlclose()

局限性

- 要求运行时，环境中存在编译器 g++
- 要求函数代码符合 C++ 语法
- 越界访问的错误难以提前检测

感谢聆听