北京航空航天大学数学科学学院实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称：科学计算通识实验课 | | 实验名称：实验八： 常微分方程的初值问题 | |
| 实验类型： 演示性实验□ 验证性实验☑ 综合性实验□ 设计性实验□ | | | |
| 班级：暑期班 | 姓名：涂磊 | | 学号：22377211 |
| 实验日期： 2024/7/16 | 指导教师：冯成亮 | | 实验成绩： |
| 实验环境：（所用仪器设备及软件）  Windows + VS-code, Ubuntu 20.04.6 + g++ | | | |
| 实验目的与实验内容：  【目的要求】  通过本实验使学生进一步熟悉个人电脑上C++代码的编写与调试，服务器上的代码编译与运行；了解常微分方程初值问题求解中的微分-积分算法设计基本思想，熟练掌握求解一维常微分方程的向前欧拉方法、向后欧拉方法和梯形方法，了解它们对步长h的稳定性要求；了解对欧拉方法的精度改进过程，掌握休恩方法（二级迭代法）的求解过程；了解龙格库塔（R-K）方法的构造思路，掌握使用二级、三级和四级R-K方法求解常微分方程的能力。  【实验内容】  【实验内容】  实验1.1：（向前欧拉法求解常微分方程1）  使用向前欧拉法（显式），分别用步长h= 1,1/2,1/4，。。。，1/64求解常微分方程在区间[0,3]上的初值问题，并比较它们的绝对误差。    实验1.2：（向后欧拉法求解常微分方程1）  使用向后欧拉法（隐式），分别用步长h= 1,1/2,1/4，。。。，1/64求解常微分方程在区间[0,3]上的初值问题，并比较它们的绝对误差。    实验1.3：（预估-修正法向后欧拉法求解常微分方程1）  使用预估-修正法（迭代欧拉法）（显式），分别用步长h= 1,1/2,1/4，。。。，1/64求解常微分方程在区间[0,3]上的初值问题，并比较它们的绝对误差。    实验2.1：（二级龙格库塔（R-K）法求解常微分方程1）  使用二级龙格库塔（R-K）法（显式），分别用步长h= 1,1/2,1/4，。。。，1/64求解常微分方程在区间[0,3]上的初值问题，并比较它们的绝对误差。    实验2.2：（三级龙格库塔（R-K）法求解常微分方程1）  使用三级龙格库塔（R-K）法（显式），分别用步长h= 1,1/2,1/4，。。。，1/64求解常微分方程在区间[0,3]上的初值问题，并比较它们的绝对误差。    实验2.3：（四级龙格库塔（R-K）法求解常微分方程1）  使用四级龙格库塔（R-K）法（显式），分别用步长h= 1,1/2,1/4，。。。，1/64求解常微分方程在区间[0,3]上的初值问题，并比较它们的绝对误差。    实验3.1：（四级龙格库塔（R-K）法求解常微分方程组2）  使用四级龙格库塔（R-K）法（显式），用步长h= 0.02求解常微分方程组在区间[0.0,0.2]上的初值问题，并比较它们的绝对误差。 | | | |
| 实验1.1：（向前欧拉法求解常微分方程1）  #include <iostream>  #include "func.hpp"  using namespace std;  double f(double x, double y){      return (x-y)/2.0;  }  double x\_t(double x, double y, double t){      return 0;  } //定义微分方程的x项  double y\_t(double x, double y, double t){      return 0;  } //定义微分方程的y项  int main(){      double t0 = 0.0;      double y0 = 1.0;      double h = 1/100.0;      double a = 0;      double b = 3;      cout << fixed;      cout.precision(6);      cout << "Forward Euler method" << endl;      cout << "---------------" << endl;      cout << "h\t\ty(3)\t\treal\_y(3)\terror" <<endl;      cout << "---------------" << endl;        for (double h = 1.0; h >= 1.0/128; h /= 2.0){          cout << "1/" << (int)(1/h) << "\t\t" << F\_Euler(a, b, t0, y0, h) << "\t" << 3.0\*exp(-3/2.0)-2+3.0<< "\t" << 3.0\*exp(-3/2.0)-2+3.0 - F\_Euler(a, b, t0, y0, h) << endl;      }      cout << "---------------" << endl;        /\* h = 1.0/100.0;      cout << F\_Euler(a, b, t0, y0, h) << endl;      cout << B\_Euler(a, b, t0, y0, h) << endl;      cout << PC\_B\_Euler(a, b, t0, y0, h) << endl; \*/      cout << endl;      cout << "When h = 1/64, the y(t) is: " << endl;      h = 1.0/64;      cout << "---------------" << endl;      cout << "t\t\ty(t)\t\treal\_y(t)\terror" << endl;      cout << "---------------" << endl;      for(int i = 0; i\*h <= b; i++){          double t = t0 + i\*h;          if (t == 0.0 || t == 0.125 || t == 0.25 || t == 0.375 || t == 0.5 || t == 0.75 ||t == 1.0 || t == 1.5 || t == 2.0 || t == 2.5 || t == 3.0){              double y = F\_Euler(a, t, t0, y0, h);          cout << t << "\t" << y << "\t" << 3.0\*exp(-t/2.0)-2+t << "\t" << 3.0\*exp(-t/2.0)-2+t - y << endl;          }      }        return 0;  }    实验1.2：（向后欧拉法求解常微分方程1）  #include <iostream>  #include "func.hpp"  using namespace std;  double f(double x, double y){      return (x-y)/2.0;  }  double x\_t(double x, double y, double t){      return 0;  } //定义微分方程的x项  double y\_t(double x, double y, double t){      return 0;  } //定义微分方程的y项  int main(){      double t0 = 0.0;      double y0 = 1.0;      double h = 1/100.0;      double a = 0;      double b = 3;      cout << fixed;      cout.precision(6);      cout << "Backward Euler method" << endl;      cout << "---------------" << endl;      cout << "h\t\ty(3)\t\treal\_y(3)\terror" <<endl;      cout << "---------------" << endl;        for (double h = 1.0; h >= 1.0/128; h /= 2.0){          cout << "1/" << (int)(1/h) << "\t\t" << B\_Euler(a, b, t0, y0, h) << "\t" << 3.0\*exp(-3/2.0)-2+3.0<< "\t" << 3.0\*exp(-3/2.0)-2+3.0 - B\_Euler(a, b, t0, y0, h) << endl;      }      cout << "---------------" << endl;      cout << endl;      cout << "When h = 1/64, the y(t) is: " << endl;      h = 1.0/64;      cout << "---------------" << endl;      cout << "t\t\ty(t)\t\treal\_y(t)\terror" << endl;      cout << "---------------" << endl;      for(int i = 0; i\*h <= b; i++){          double t = t0 + i\*h;          if (t == 0.0 || t == 0.125 || t == 0.25 || t == 0.375 || t == 0.5 || t == 0.75 ||t == 1.0 || t == 1.5 || t == 2.0 || t == 2.5 || t == 3.0){              double y = B\_Euler(a, t, t0, y0, h);          cout << t << "\t" << y << "\t" << 3.0\*exp(-t/2.0)-2+t << "\t" << 3.0\*exp(-t/2.0)-2+t - y << endl;          }      }            return 0;  }    实验1.3：（预估-修正法向后欧拉法求解常微分方程1）  #include <iostream>  #include "func.hpp"  using namespace std;  double f(double x, double y){      return (x-y)/2.0;  }  double x\_t(double x, double y, double t){      return 0;  } //定义微分方程的x项  double y\_t(double x, double y, double t){      return 0;  } //定义微分方程的y项  int main(){      double t0 = 0.0;      double y0 = 1.0;      double h = 1/100.0;      double a = 0;      double b = 3;      cout << fixed;      cout.precision(6);      cout << "Estimate Correction Method" << endl;      cout << "---------------" << endl;      cout << "h\t\ty(3)\t\treal\_y(3)\terror" <<endl;      cout << "---------------" << endl;        for (double h = 1.0; h >= 1.0/128; h /= 2.0){          cout << "1/" << (int)(1/h) << "\t\t" << EC\_Euler(a, b, t0, y0, h) << "\t" << 3.0\*exp(-3/2.0)-2+3.0<< "\t" << 3.0\*exp(-3/2.0)-2+3.0 - EC\_Euler(a, b, t0, y0, h) << endl;      }      cout << "---------------" << endl;      cout << endl;      cout << "When h = 1/64, the y(t) is: " << endl;      h = 1.0/64;      cout << "---------------" << endl;      cout << "t\t\ty(t)\t\treal\_y(t)\terror" << endl;      cout << "---------------" << endl;      for(int i = 0; i\*h <= b; i++){          double t = t0 + i\*h;          if (t == 0.0 || t == 0.125 || t == 0.25 || t == 0.375 || t == 0.5 || t == 0.75 ||t == 1.0 || t == 1.5 || t == 2.0 || t == 2.5 || t == 3.0){              double y = EC\_Euler(a, t, t0, y0, h);          cout << t << "\t" << y << "\t" << 3.0\*exp(-t/2.0)-2+t << "\t" << 3.0\*exp(-t/2.0)-2+t - y << endl;          }      }      return 0;  }    实验2.1：（二级龙格库塔（R-K）法求解常微分方程1）  #include <iostream>  #include "func.hpp"  using namespace std;  double f(double x, double y){      return (x-y)/2.0;  }  double x\_t(double x, double y, double t){      return 0;  } //定义微分方程的x项  double y\_t(double x, double y, double t){      return 0;  } //定义微分方程的y项  int main(){      double t0 = 0.0;      double y0 = 1.0;      double h = 1/100.0;      double a = 0;      double b = 3;      cout << fixed;      cout.precision(6);      cout << "Runge-Kutta 2" << endl;      cout << "---------------" << endl;      cout << "h\t\ty(3)\t\treal\_y(3)\terror" <<endl;      cout << "---------------" << endl;        for (double h = 1.0; h >= 1.0/64; h /= 2.0){          cout << "1/" << (int)(1/h) << "\t\t" << RK\_2(a, b, t0, y0, h) << "\t" << 3.0\*exp(-3/2.0)-2+3.0<< "\t" << 3.0\*exp(-3/2.0)-2+3.0 - RK\_2(a, b, t0, y0, h) << endl;      }      cout << "---------------" << endl;      cout << endl;      cout << "When h = 1/64, the y(t) is: " << endl;      h = 1.0/64;      cout << "---------------" << endl;      cout << "t\t\ty(t)\t\treal\_y(t)\terror" << endl;      cout << "---------------" << endl;      for(int i = 0; i\*h <= b; i++){          double t = t0 + i\*h;          if (t == 0.0 || t == 0.125 || t == 0.25 || t == 0.375 || t == 0.5 || t == 0.75 ||t == 1.0 || t == 1.5 || t == 2.0 || t == 2.5 || t == 3.0){              double y = RK\_2(a, t, t0, y0, h);          cout << t << "\t" << y << "\t" << 3.0\*exp(-t/2.0)-2+t << "\t" << 3.0\*exp(-t/2.0)-2+t - y << endl;          }      }      return 0;  }    实验2.2：（三级龙格库塔（R-K）法求解常微分方程1）  #include <iostream>  #include "func.hpp"  using namespace std;  double f(double x, double y){      return (x-y)/2.0;  }  double x\_t(double x, double y, double t){      return 0;  } //定义微分方程的x项  double y\_t(double x, double y, double t){      return 0;  } //定义微分方程的y项  int main(){      double t0 = 0.0;      double y0 = 1.0;      double h = 1/100.0;      double a = 0;      double b = 3;      cout << fixed;      cout.precision(6);      cout << "Runge-Kutta 3" << endl;      cout << "---------------" << endl;      cout << "h\t\ty(3)\t\treal\_y(3)\terror" <<endl;      cout << "---------------" << endl;        for (double h = 1.0; h >= 1.0/128; h /= 2.0){          cout << "1/" << (int)(1/h) << "\t\t" << RK\_3(a, b, t0, y0, h) << "\t" << 3.0\*exp(-3/2.0)-2+3.0<< "\t" << 3.0\*exp(-3/2.0)-2+3.0 - RK\_3(a, b, t0, y0, h) << endl;      }      cout << "---------------" << endl;        cout << endl;      cout << "When h = 1/64, the y(t) is: " << endl;      h = 1.0/64;      cout << "---------------" << endl;      cout << "t\t\ty(t)\t\treal\_y(t)\terror" << endl;      cout << "---------------" << endl;      for(int i = 0; i\*h <= b; i++){          double t = t0 + i\*h;          if (t == 0.0 || t == 0.125 || t == 0.25 || t == 0.375 || t == 0.5 || t == 0.75 ||t == 1.0 || t == 1.5 || t == 2.0 || t == 2.5 || t == 3.0){              double y = RK\_3(a, t, t0, y0, h);          cout << t << "\t" << y << "\t" << 3.0\*exp(-t/2.0)-2+t << "\t" << 3.0\*exp(-t/2.0)-2+t - y << endl;          }      }      return 0;  }    实验2.3：（四级龙格库塔（R-K）法求解常微分方程1）  #include <iostream>  #include "func.hpp"  using namespace std;  double f(double x, double y){      return (x-y)/2.0;  }  double x\_t(double x, double y, double t){      return 0;  } //定义微分方程的x项  double y\_t(double x, double y, double t){      return 0;  } //定义微分方程的y项  int main(){        double t0 = 0.0;      double y0 = 1.0;      double h = 1/100.0;      double a = 0;      double b = 3;//求解y(b)      cout << fixed;      cout.precision(6);      cout << "Runge-Kutta 4" << endl;      cout << "---------------" << endl;      cout << "h\t\ty(3)\t\treal\_y(3)\terror" <<endl;      cout << "---------------" << endl;        for (double h = 1.0; h >= 1.0/128.0; h /= 2.0){          cout << "1/" << (int)(1/h) << "\t\t" << RK\_4(a, b, t0, y0, h) << "\t" << 3.0\*exp(-3/2.0)-2+3.0<< "\t" << 3.0\*exp(-3/2.0)-2+3.0 - RK\_4(a, b, t0, y0, h) << endl;      }      cout << "---------------" << endl;        cout << endl;      cout << "When h = 1/64, the y(t) is: " << endl;      h = 1.0/64;      cout << "---------------" << endl;      cout << "t\t\ty(t)\t\treal\_y(t)\terror" << endl;      cout << "---------------" << endl;      for(int i = 0; i\*h <= b; i++){          double t = t0 + i\*h;          if (t == 0.0 || t == 0.125 || t == 0.25 || t == 0.375 || t == 0.5 || t == 0.75 ||t == 1.0 || t == 1.5 || t == 2.0 || t == 2.5 || t == 3.0){              double y = RK\_4(a, t, t0, y0, h);          cout << t << "\t" << y << "\t" << 3.0\*exp(-t/2.0)-2+t << "\t" << 3.0\*exp(-t/2.0)-2+t - y << endl;          }      }        return 0;  }    实验3.1：（四级龙格库塔（R-K）法求解常微分方程组2）  #include <iostream>  #include "func.hpp"  using namespace std;  double f(double x, double y) {      return 0;  }  double x\_t(double t, double x, double y) {      return x + 2\*y;  }  double y\_t(double t, double x, double y) {      return 3\*x + 2\*y;  }  int main() {      double a = 0.0;      double b = 0.2;      double t0 = 0;      double x0 = 6;      double y0 = 4;      double h = 0.02;      cout << fixed;      cout.precision(8);      cout << "t\t\tx\t\ty" << endl;      cout << "----------" << endl;      for (int i = 0; i\*h <= b; i++){          double t = t0 + i\*h;          double x = RK\_4\_2\_x(a, t, t0, x0, y0, h);          double y = RK\_4\_2\_y(a, t, t0, x0, y0, h);          cout << t << "\t" << x << "\t" << y << endl;      }      return 0;  }    实验4.1：补充实验（四级龙格库塔（R-K）法求解常微分方程组）  #include <iostream>  #define PI 3.14159265358979323846264338327950288419716939937510  using namespace std;  double x\_t(double t, double x, double y,double p,double q) {      double K = 1000.0;      double l = 6.0;      double a = 0.2;      double omega = 2\*PI\*(38/60.0);      double W = 80.0;      double m = 2500.0;      double d = 0.01;      double result = q;      return result;  }  double dx\_t(double t, double x, double y, double p, double q) {      double K = 1000.0;      double l = 6.0;      double a = 0.2;      double omega = 2\*PI\*(38/60.0);      double W = 80.0;      double m = 2500.0;      double d = 0.01;        //double result = -q\*d + 3\*cos(x)\*K\*(1.0/(l\*m\*a)) \* (exp(a\*(y-l\*sin(x))) - exp(a\*(y)+sin(x)));      //题目中所给公式 印刷错误，应该是：      double result = -q\*d + 3\*cos(x)\*K\*(1.0/(l\*m\*a)) \* (exp(a\*(y-l\*sin(x))) - exp(a\*(y+l\*sin(x))));      return result;  }  double y\_t(double t, double x, double y, double p, double q) {      double K = 1000.0;      double l = 6.0;      double a = 0.2;      double omega = 2\*PI\*(38/60.0);      double W = 80.0;      double m = 2500.0;      double d = 0.01;      double result = p;      return result;  }  double dy\_t(double t, double x, double y,double p, double q) {      double K = 1000.0;      double l = 6.0;      double a = 0.2;      double omega = 2\*PI\*(38/60.0);      double W = 80.0;      double m = 2500.0;      double d = 0.01;      double result = -d\*p - K\*(1.0/(m\*a)) \* (exp(a\*(y-l\*sin(x))) + exp(a\*y+l\*sin(x)) -2) + 0.2\*W\*sin(omega\*t);      return result;  }  double RK\_4\_4\_x(double a, double b, double t0, double x0, double dx0, double y0,double dy0, double h) {      //用4阶龙格库塔法求解x(t),dx(t),y(t),dy(t)以（t0,x0,dx0,y0,dy0）为初值点的微分方程      double f1 = h \* x\_t(t0, x0, y0, dy0, dx0);      double g1 = h \* dx\_t(t0, x0, y0, dy0, dx0);      double h1 = h \* y\_t(t0, x0, y0, dy0, dx0);      double k1 = h \* dy\_t(t0, x0, y0,dy0, dx0);      double f2 = h \* x\_t(t0 + h / 2.0, x0 + f1 / 2.0, y0 + h1 / 2.0, dy0 + k1 / 2.0, dx0 + g1 / 2.0);      double g2 = h \* dx\_t(t0 + h / 2.0, x0 + f1 / 2.0, y0 + h1 / 2.0, dy0 + k1 / 2.0, dx0 + g1 / 2.0);      double h2 = h \* y\_t(t0 + h / 2.0, x0 + f1 / 2.0, y0 + h1 / 2.0, dy0 + k1 / 2.0, dx0 + g1 / 2.0);      double k2 = h \* dy\_t(t0 + h / 2.0, x0 + f1 / 2.0, y0 + h1 / 2.0, dy0 + k1 / 2.0, dx0 + g1 / 2.0);      double f3 = h \* x\_t(t0 + h / 2.0, x0 + f2 / 2.0, y0 + h2 / 2.0, dy0 + k2 / 2.0, dx0 + g2 / 2.0);      double g3 = h \* dx\_t(t0 + h / 2.0, x0 + f2 / 2.0, y0 + h2 / 2.0, dy0 + k2 / 2.0, dx0 + g2 / 2.0);      double h3 = h \* y\_t(t0 + h / 2.0, x0 + f2 / 2.0, y0 + h2 / 2.0, dy0 + k2 / 2.0, dx0 + g2 / 2.0);      double k3 = h \* dy\_t(t0 + h / 2.0, x0 + f2 / 2.0, y0 + h2 / 2.0, dy0 + k2 / 2.0, dx0 + g2 / 2.0);      double f4 = h \* x\_t(t0 + h, x0 + f3, y0 + h3, dy0 + k3, dx0 + g3);      double g4 = h \* dx\_t(t0 + h, x0 + f3, y0 + h3, dy0 + k3, dx0 + g3);      double h4 = h \* y\_t(t0 + h, x0 + f3, y0 + h3, dy0 + k3, dx0 + g3);      double k4 = h \* dy\_t(t0 + h, x0 + f3, y0 + h3, dy0 + k3, dx0 + g3);      double result\_x = x0 ;      double result\_dx = dx0;      double result\_y = y0 ;      double result\_dy = dy0 ;      for (double t = t0 + h; t <= b; t += h) {          result\_x += (f1 + 2.0 \* f2 + 2.0 \* f3 + f4) / 6.0;          result\_dx += (g1 + 2.0 \* g2 + 2.0 \* g3 + g4) / 6.0;          result\_y += (h1 + 2.0 \* h2 + 2.0 \* h3 + h4) / 6.0;          result\_dy += (k1 + 2.0 \* k2 + 2.0 \* k3 + k4) / 6.0;          f1 = h \* x\_t(t, result\_x, result\_y, result\_dy, result\_dx);          g1 = h \* dx\_t(t, result\_x, result\_y, result\_dy, result\_dx);          h1 = h \* y\_t(t, result\_x, result\_y, result\_dy, result\_dx);          k1 = h \* dy\_t(t, result\_x, result\_y, result\_dy, result\_dx);          f2 = h \* x\_t(t + h / 2.0, result\_x + f1 / 2.0, result\_y + h1 / 2.0, result\_dy + k1 / 2.0, result\_dx + g1 / 2.0);          g2 = h \* dx\_t(t + h / 2.0, result\_x + f1 / 2.0, result\_y + h1 / 2.0, result\_dy + k1 / 2.0, result\_dx + g1 / 2.0);          h2 = h \* y\_t(t + h / 2.0, result\_x + f1 / 2.0, result\_y + h1 / 2.0, result\_dy + k1 / 2.0, result\_dx + g1 / 2.0);          k2 = h \* dy\_t(t + h / 2.0, result\_x + f1 / 2.0, result\_y + h1 / 2.0, result\_dy + k1 / 2.0, result\_dx + g1 / 2.0);          f3 = h \* x\_t(t + h / 2.0, result\_x + f2 / 2.0, result\_y + h2 / 2.0, result\_dy + k2 / 2.0, result\_dx + g2 / 2.0);          g3 = h \* dx\_t(t + h / 2.0, result\_x + f2 / 2.0, result\_y + h2 / 2.0, result\_dy + k2 / 2.0, result\_dx + g2 / 2.0);          h3 = h \* y\_t(t + h / 2.0, result\_x + f2 / 2.0, result\_y + h2 / 2.0, result\_dy + k2 / 2.0, result\_dx + g2 / 2.0);          k3 = h \* dy\_t(t + h / 2.0, result\_x + f2 / 2.0, result\_y + h2 / 2.0, result\_dy + k2 / 2.0, result\_dx + g2 / 2.0);          f4 = h \* x\_t(t + h, result\_x + f3, result\_y + h3, result\_dy + k3, result\_dx + g3);          g4 = h \* dx\_t(t + h, result\_x + f3, result\_y + h3, result\_dy + k3, result\_dx + g3);          h4 = h \* y\_t(t + h, result\_x + f3, result\_y + h3, result\_dy + k3, result\_dx + g3);          k4 = h \* dy\_t(t + h, result\_x + f3, result\_y + h3, result\_dy + k3, result\_dx + g3);      }      return result\_x;  }    int main() {      //x = 0;      //y = y;      double a = 0.0;      double b = 600.0;      double t0 = 0.0;      double x0 = 0.00000001;//初始时只有一点点倾斜角                              //若初始时没有倾斜角，x一直为0      double dx0 = 0.0;      double y0 = 0.0;      double dy0 = 0.0;      double h = 1.0;      cout << fixed;      cout.precision(8);      cout << "t\t\tx" << endl;      cout << "----------" << endl;      for (int i = 0; i\*h <= b; i++){          double t = t0 + i\*h;          double x = RK\_4\_4\_x(a, t, t0, x0,dx0, y0,dy0, h);          cout << t << "\t" << x << endl;      }      return 0;  } | | | |
| 实验分析与总结：  经过本次实验，了解到了误差产生的原因以及为什么要避免误差，如何避免误差。  强化了编程能力，学会了如何使用远程服务器辅助完成代码的运行。  向前欧拉法与向后欧拉法 计算结果误差在同一数量级  预估修正法的误差要比向前欧拉法和向后欧拉法小，在相同的间隔下  二级龙哥库达，三级龙哥库达，四级龙哥库达方法的误差逐渐减小，且他们都比向前欧拉法，向后欧拉法，预估修正法的误差小，但计算量大  初值一点点扰动也可能导致桥面发生很大变化，如补充题中，当角度初始为0时，其一直为0，但若将其改为 0.00000001， 600秒后就会变成四万多 | | | |