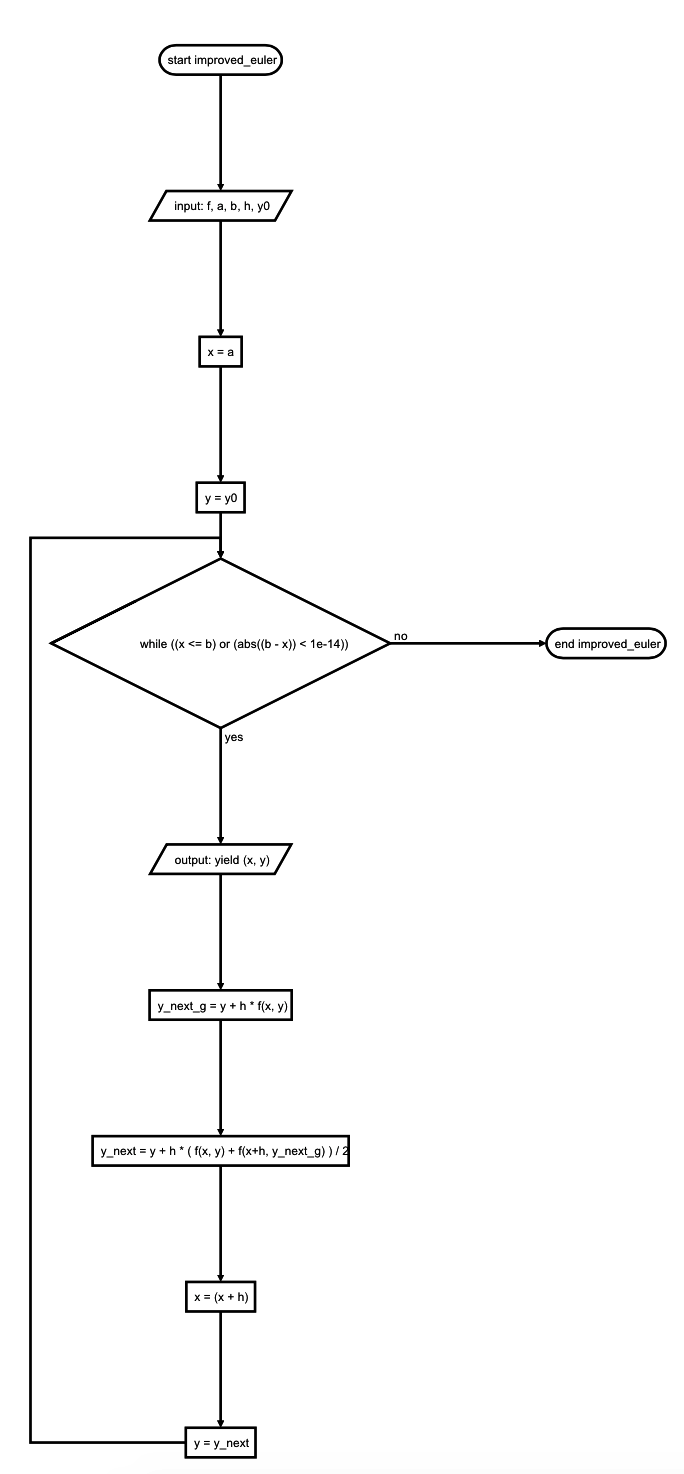
实验五 常微分方程初值问题的数值解法

一、实验目的

① 掌握常微分方程数值解的常用算法；

② 培养编程与上机调试能力。

二、实验过程和结果



【图1】改进欧拉算法

1. 改进欧拉算法

常微分方程的初值问题：

改进欧拉算法：

实现程序[[1]](#footnote-1)：

improved\_euler(f, a, b, h, y0)

输入参数：

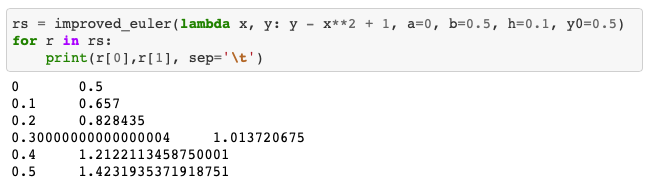
* f: 二元函数：
* a, b: float, x 的区间
* h: float , x 迭代步：
* y0: float, 初值

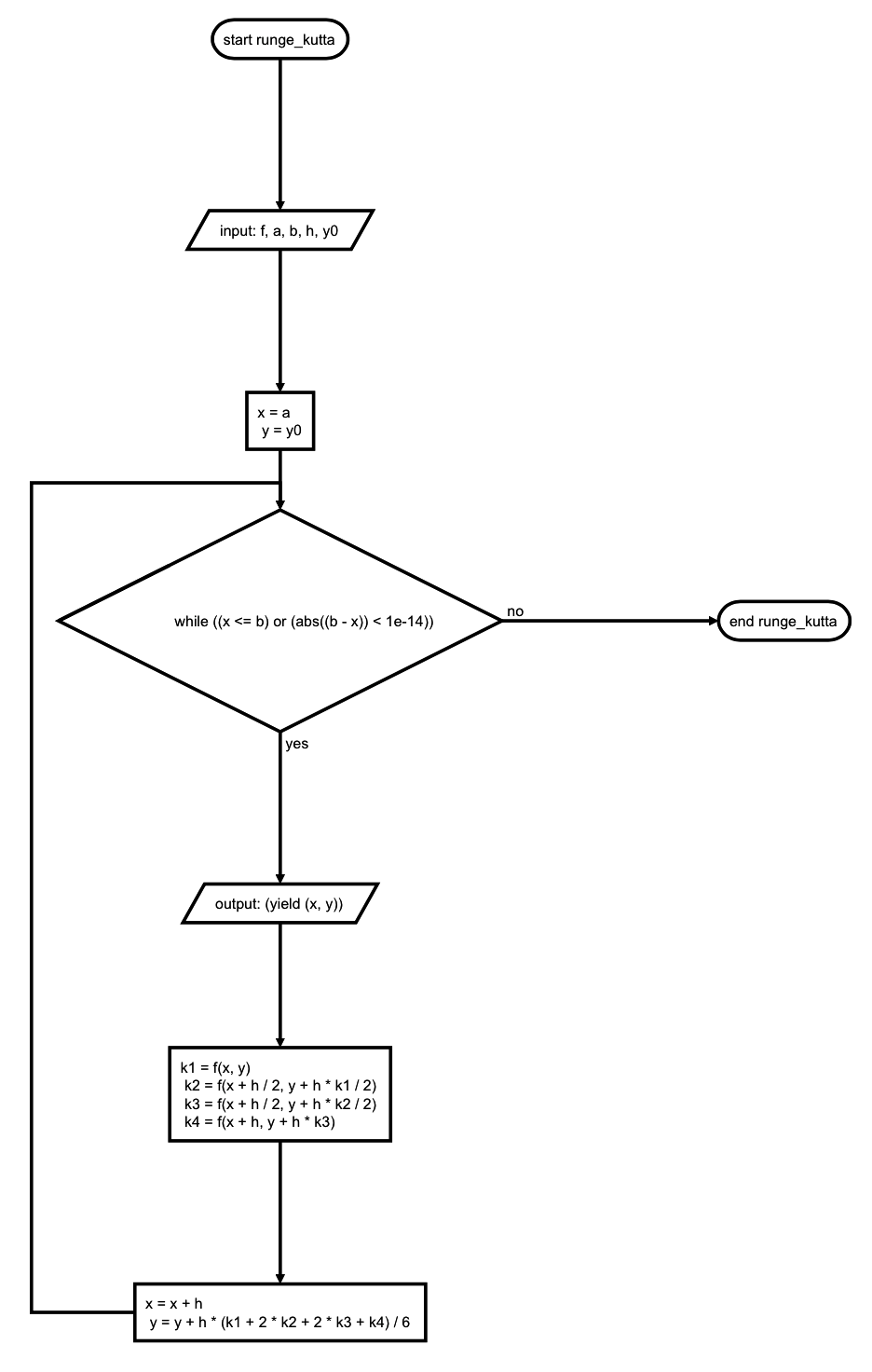
返回输出：

(x, y)：方程的解 for x from a to b, step h.

由于这个东西可以需要返回很多值（当区间 比较大，步长 又比较小时），如果用传统的方法返回一个列表（向量）储存全部的值，会消耗大量内存，计算所有结果也需要程序 CPU 密集运行一段时间。而我们真实使用的时候（例如画图时），只需在结果上做迭代，逐次取用结果，一次只使用一个，很少需要同时使用完整的全部结果。

这种情况下，我把该函数实现为一个 Python 的生成器（Generator），实现惰性计算。每次取用 next(improved\_euler(f, a, b, h, y0)) 时，才完成一步的计算，并输出这一步的值。

改进欧拉法的调用实例[[2]](#footnote-2)：



【图2】RK4

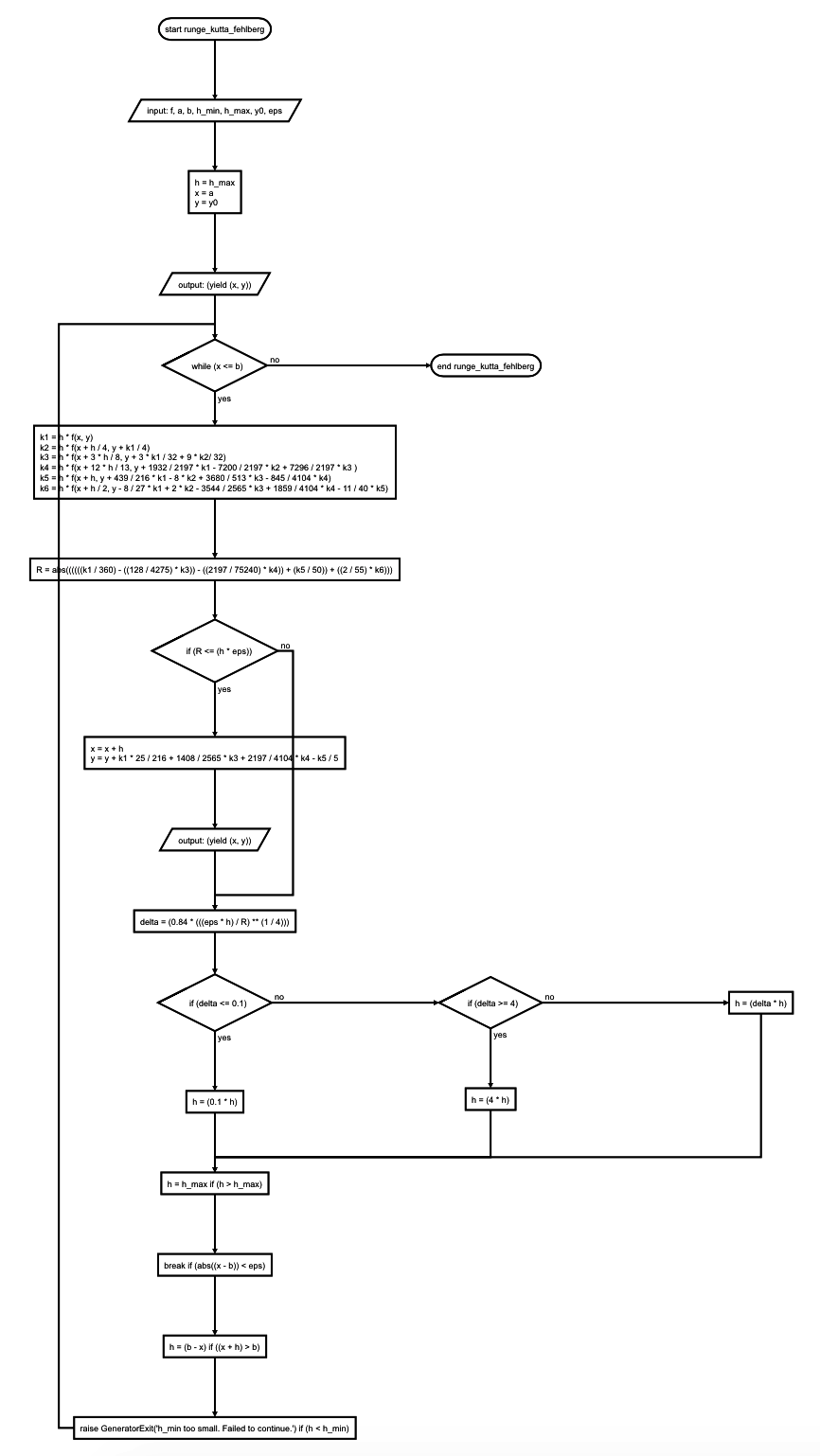
1. 四阶龙格-库塔算法

经典的四阶 Runge-Kutta 算法（RK4）：

看上去比较复杂，但写程序十分容易实现[[3]](#footnote-3)：

runge\_kutta(f, a, b, h, y0)

该函数输入输出都与前面的改进欧拉法相同，内部实现也很类似，同样实现为生成器，做惰性计算。



【图3】RKF

1. RKF 算法

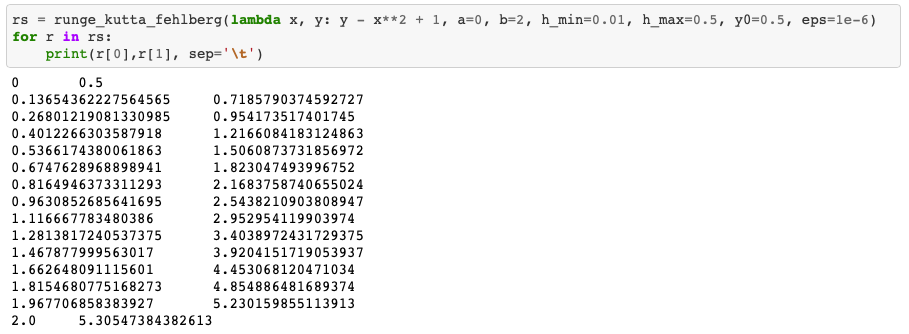
RKF 是自适步长的 Runge-Kutta 方法，在 Runge-Kutta 算法的基础上，增加了动态调整步长大小。

具体来说，RKF 方法使用四阶的 Runge-Kutta 方法来计算结果，并使用五阶 Runge-Kutta 的结果与四阶的结果之差的绝对值估计截断误差，从而预测步长。

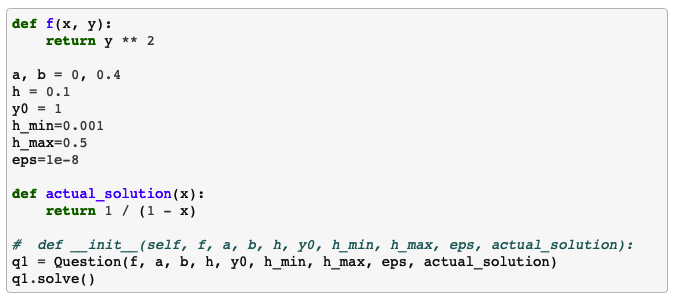
实现程序[[4]](#footnote-4)：

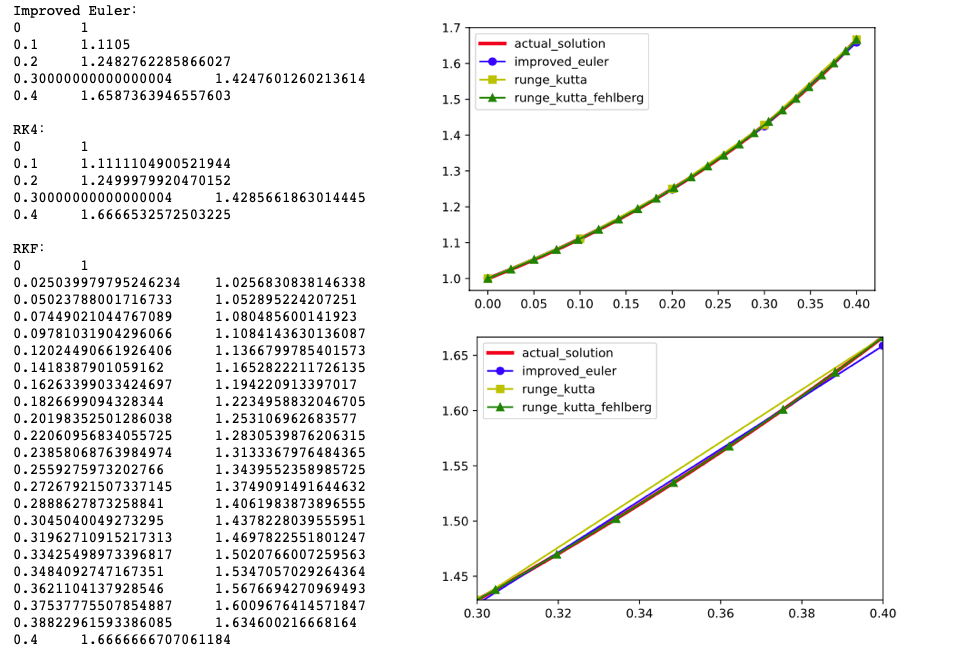
runge\_kutta\_fehlberg(f, a, b, h\_min, h\_max, y0, eps)

这个函数的输入和之前的略有不同，把之前确定的步长 h 改成了步长取值区间 [h\_min, h\_max] ，并且要求容许误差 eps。

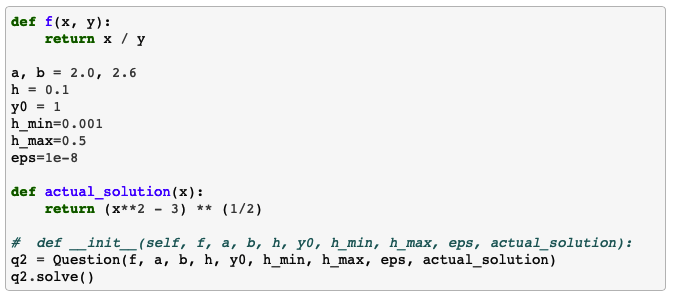
调用实例，可以看到计算时动态的步长调整：

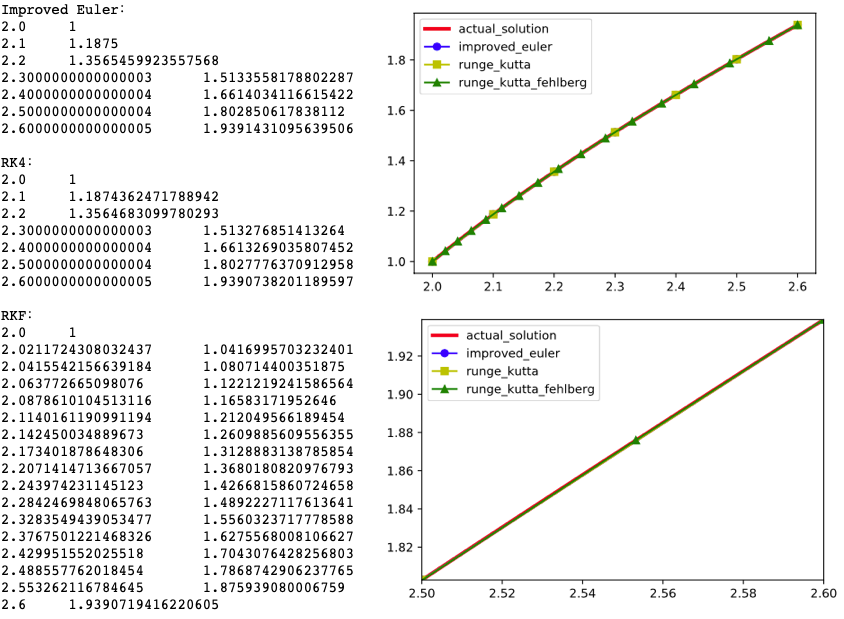
1. 实验内容题目

分别使用改进欧拉算法、RK4 和 RKF 算法进行求解[[5]](#footnote-5)，这里封装了一个类来调用各种算法求解问题，并作图比较。调用即可完成求解：

最终等到的结果：

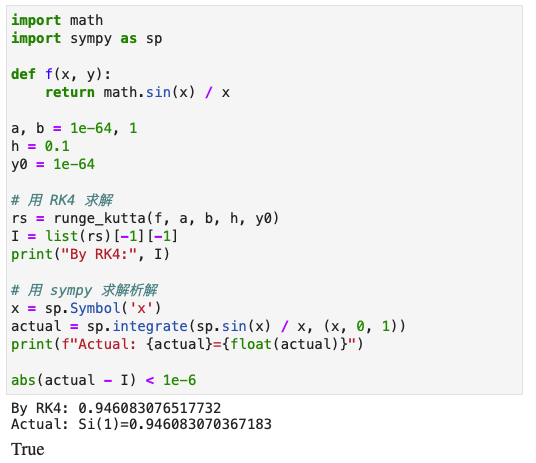


同样使用改进欧拉算法、RK4 和 RKF 算法进行求解：

最终等到的结果（由于结果差距很小，在图像上，先画的改进欧拉法被其他算法盖住了）：

三、思考题分析解答

1. 用常微分方程初值问题的数值方法计算 .

问题转化为求 在 1 处的值。用四阶龙格库塔求解：

1. RKF 算法的实现

详见 [实验内容 3. RKF 算法](#Rkf算法)。

四、重点难点分析

重点：

① 掌握常微分方程数值解的常用算法；

② 培养编程与上机调试能力。

难点：

RKF 方法的编程实现，其中有大量常数的输入、计算，容易错，可以考虑用一个矩阵储存 k1 ～ k6 计算过程中的各种常数，这样可以大幅降低打字错误的风险。这种方法的实现在这一次 Git 提交：[10a08ee](https://github.com/cdfmlr/NumericalAnalysis/commit/10a08eec3f766abba958c247628937f39ed9a034)。

1. 改进欧拉法源码：<https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex5/src/improved_euler.py> [↑](#footnote-ref-1)
2. 调用实例节选自： <https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex5/src/ex5.ipynb> [↑](#footnote-ref-2)
3. RK4 实现源码：<https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex5/src/rk4.py> [↑](#footnote-ref-3)
4. RKF 源码：<https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex5/src/rkf.py> [↑](#footnote-ref-4)
5. 具体的求解过程见： <https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex5/src/ex5.ipynb> [↑](#footnote-ref-5)