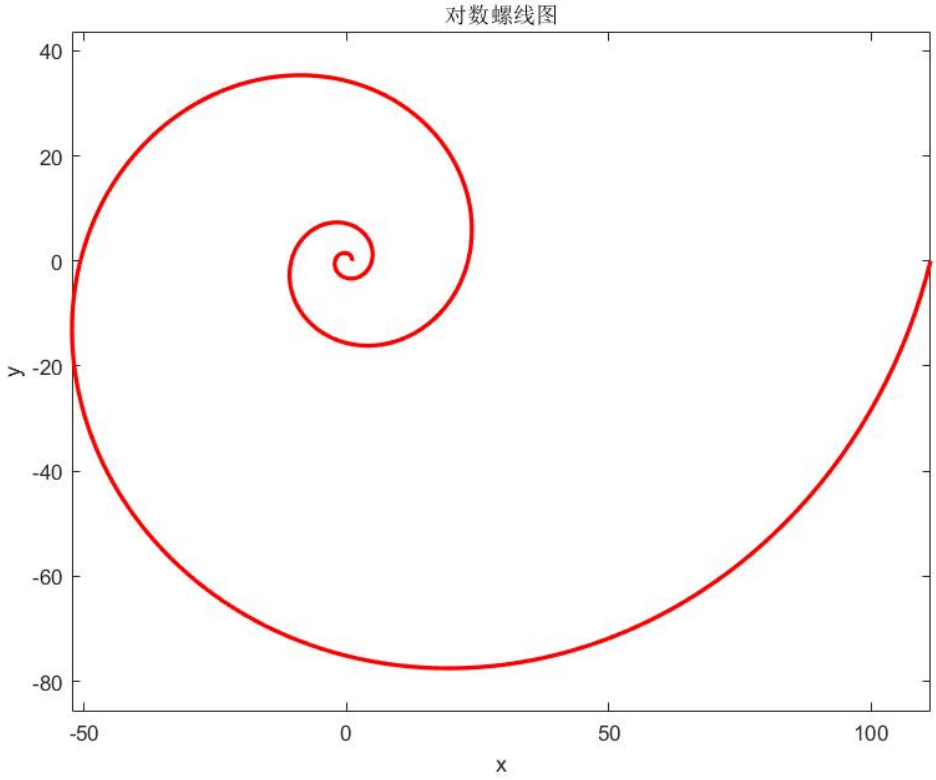
MATLAB 第 2 次作业

第一题

绘制结果图如下：



hw2_1.m

```
get_x=@(theta)(exp(1/4*theta)).*cos(theta); %x 关于 theta 的匿名函数
get_y=@(theta)(exp(1/4*theta)).*sin(theta); %y 关于 theta 的匿名函数
x=get_x((0:pi/100:6*pi));                   %获取 x,theta 范围在 0-6pi
y=get_y((0:pi/100:6*pi));                   %获取 y
plot(x,y,'-r','LineWidth',2)                %绘图
title('对数螺线图')                          %取标题
xlabel('x')                                 %坐标轴命名
ylabel('y')
axis equal
```

第二题

程序运行结果如下所示：

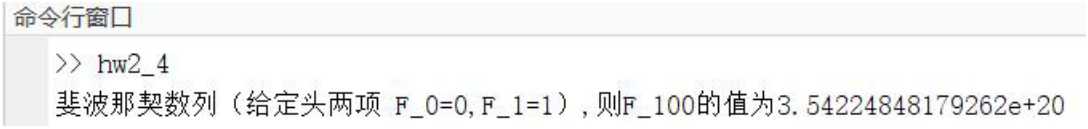
```
>> hw2_2
不晚于2020-2出生的人所占比例为 50 %
重新从矮到高的排序后的学号序列为 6   3   5  10   2   1   7   4   9   8
```

编程内容详见 hw2_2 文件夹中的主程序

hw2_2.m

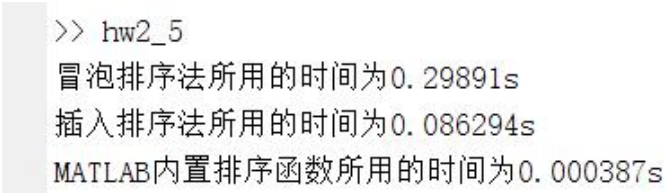
第三题
运行主程序后，输出文件结果详见文档
assignment3_output.txt
编程内容详见 hw2_3 文件夹中的主程序
hw2_3.m

第四题
运行主程序后后，输出框输出结果如下：



编程内容详见 hw2_4 文件夹中的主程序
hw2_4.m

第五题
自行编写了冒泡排序和插入排序算法
运行主程序后得到输出结果如下：



运行时间差异分析：
可以看到，冒泡排序法效率最低，插入排序效率较高但是依然无法和 Matlab 自带的 sort 函数相比，虽然冒泡排序和插入排序算法的时间复杂度最好和最坏都分别是 O(n^2)和 O(n)，但是冒泡排序在过程中需要三个赋值操作,而插入排序只需要一个赋值操作，当排序的规模变大时，两者就有比较大的区别。而 Matlab 内置函数 sort 的算法，根据查询相关资料，是几种快速算法的混合版本，效率相当高（官方技术说明页面原文：Sorting experts recommend a three-level algorithm, insert/quick/bin, where each level calls the one to its left if the size of the array being sorted passes below some experimentally discovered cutoff. When we updated sort, we expected to require four levels (insert/quick/bin/quick).），因此在比较当中最显优势。

编程内容详见文件夹 hw2_5，其中有自编写冒泡排序函数：BubbleSort.m、自编写插入排序函数：InsertSort.m 以及主程序 hw2_5.m 用于调用上述两个函数以及 matlab 内置排序函数进行时间对比
第六题

输出结果（根据要求，建立表格定量展示）：

T =

6×2 table

	X	Y
sqrt运算保留小数点后1位	0.1	0.011123
sqrt运算保留小数点后2位	0.02	0.011123
sqrt运算保留小数点后3位	0.012	0.011123
sqrt运算保留小数点后4位	0.0111	0.011123
sqrt运算保留小数点后5位	0.01112	0.011123
sqrt运算保留小数点后6位	0.011124	0.011123

结果解释与分析：

从上面的结果中，可以看到开根号运算的保留位数越小，通过 $x=\sqrt{n}-\sqrt{n-1}$ 计算得到的结果与通过 $y=\frac{1}{\sqrt{n}+\sqrt{n-1}}$ 得到的结果差距越大，并且明显时通过转换成 $y=\frac{1}{\sqrt{n}+\sqrt{n-1}}$ 计算得到的误差小，这是由于数值计算中，直接地将两近数相减会导致有效数字的严重丢失。在上面的例子中，以保留小数点后一位为例，$\sqrt{2021}$ 为 45.0，$\sqrt{2020}$ 为 44.9，虽然两者都具有三位有效数字，但是因为我们直接显式相减，两者又是近数，相减后结果 0.1，就仅剩 1 位有效数字，造成有效数字的严重缺失，如果要避免这类现象产生，可以将近数相减转换成其它形式的运算，就如同上例所示，利用常见的共轭等式的构造，得到：

$$x=\sqrt{n}-\sqrt{n-1}=\frac{(\sqrt{n}-\sqrt{n-1})\ (\sqrt{n}+\sqrt{n-1})}{(\sqrt{n}+\sqrt{n-1})}=\frac{1}{(\sqrt{n}+\sqrt{n-1})}\Rightarrow y\text{。}$$

编程内容详见 hw2_6 文件夹中的主程序

hw2_6.m

第七题

(1)

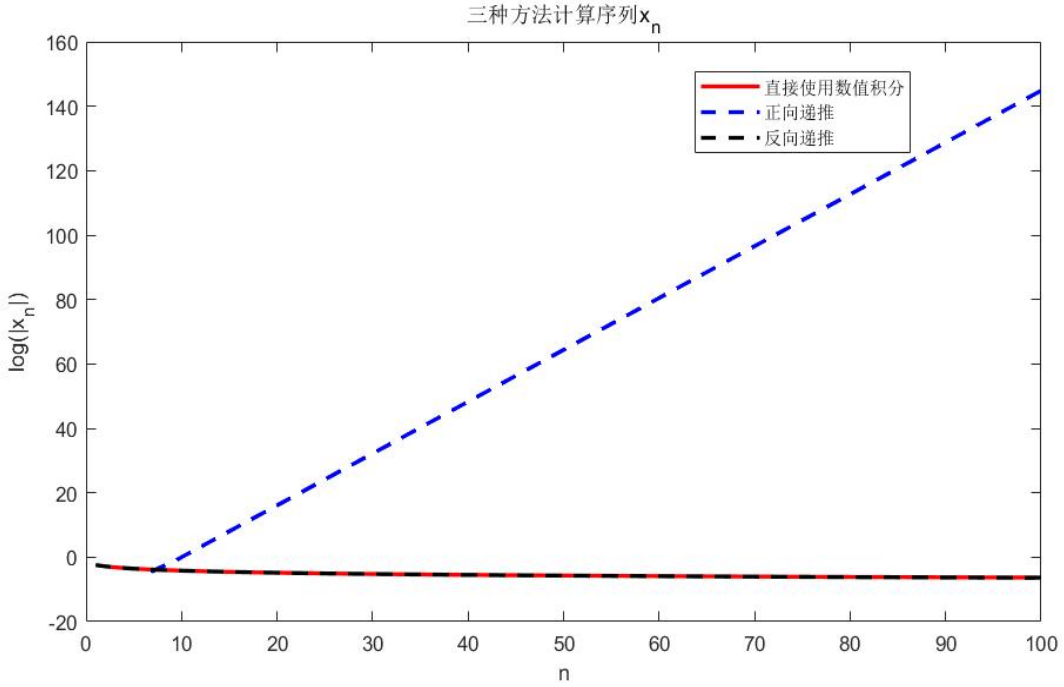
编程内容详见 hw2_7 文件夹中的自编写的数值积分函数（应题目要求为梯形法）Int_Tr.m 以及调用该函数的主程序 hw2_7_1.m

(2)

推导 $\{x_n\}$ 每一项之间的关系式：

$$x_n=\int_0^1\frac{x^n}{x+5}dx=\int_0^1\frac{x^{n-1}[(x+5)-5]}{x+5}dx=\int_0^1x^{n-1}dx-5\int_0^1\frac{x^{n-1}}{x+5}dx=\frac{1}{n}-5x_{n-1}$$

按要求绘制计算结果关于 n 的图像，输出结果如下：



结果分析与说明：

分析结果之前，可以先确定这个积分的范围

$\frac{1}{6}=\int_0^1\frac{x^n}{6}dx<x_n=\int_0^1\frac{x^n}{5+x}dx<\int_0^1\frac{x^n}{5}dx=\frac{1}{5}$，而根据上面的图像来看，可以看到直接使用数值积分和反向递推的算法几乎相同，并且处于合理的范围内，但是正向递推计算得到的结果当计算次数 n 较大时，完全不属于合理范围内，误差极大，结果不可靠。下面进行分析

方法一：直接通过梯形法进行数值积分的方法计算每一项，可以认为其结果较为准确，但是也不能忽略与解析解存在的误差（下面分析中会用到）。

方法二：通过上述方法求出首项 x_1，通过 $x_n=\frac{1}{n}-5x_{n-1}$ 不断从前向后正向递推，经过 n 次计算就有 $x_n=\sum_{i=2}^{n}\frac{5^{n-i}}{i}+(-5)^{n-1}x_1$，从结果看到通过这种方法计算的 x_n 到最后完全超出了该积分的范围，明显是不准确的。原因在于上面的数值积分计算与解析解相比，第一项存在 $E_1=x_1-x_1^*$ 的误差，从 x_n 表达式中的项 $(-5)^{n-1}x_1$ 可以看出正向递推使得误差随着计算次数 n 不断地以指数型放大（比例 5>1），因此最后的计算 x_n 不准确，故这种算法是不稳定的。

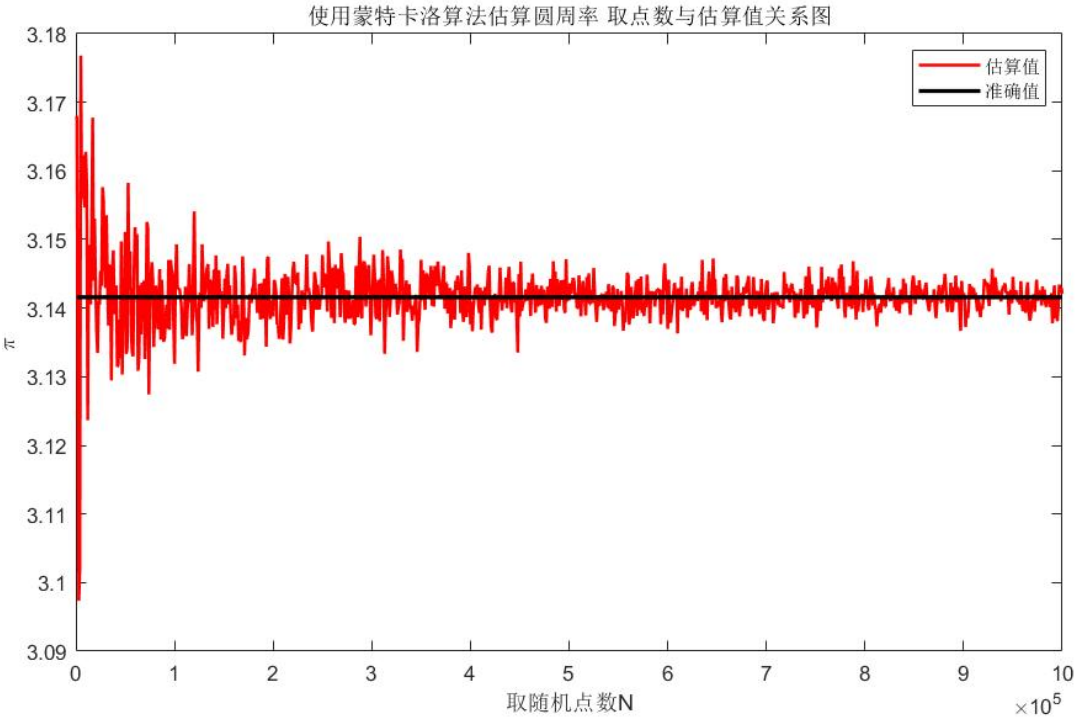
方法三：首先通过求出尾项 x_{100}，通过 $x_{n-1}=\frac{1}{5n}-\frac{1}{5}x_n$ 不断从后向前反向递推，与上面相似，但是区别在于反向递推，移项取倒数，巧妙地使得误差随着计算次数 n 指数级地缩小（比例 $\frac{1}{5}$<1），最终误差便可以最小化，所以这种算法是稳定的。

编程内容详见程序包中 hw2_7 文件夹中的程序

hw2_7_2.m

第八题

按要求绘制蒙特卡洛算法估计圆周率的估计值与采样点数关系图如下：



可以看到，总体上来说，采样点越多，估计值越接近真值。

编程内容详见 hw2_8 文件夹中的程序

hw2_8.m（注：因为采样点数取得较多，程序运行需要约 15 秒时间）

第九题

数学建模：

显然这可以建模成一个线性规划问题。

设生产甲 x_1 台，乙 x_2 台 ，由题目描述：每吨销售后的利润分别为 5 万元、3 万元，问生产甲、乙产品各多少吨，才能是总利润最大。可知总利润（最优化目标函数） $z=5x_1+3x_2$ 万元。

由题目描述：生产甲产品需要 A，B 机器加工，加工时间分别为每吨 2h 和 1h；生产乙机床需要 A,B,C 三种机器加工加工时间为每吨各 1h。若每天可用于加工的机器时长分别为 A 机器 10h、B 机器 8h、C 机器 7h。可以列出约束条件：

$$s.t.\begin{cases} 2x_1 + x_2 \le 10 \\ x_1 + x_2 \le 8 \\ x_2 \le 7 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

求解：

用 matlab 自带函数 linprog 便可求得最优化结果为 $\begin{cases} x_1 = 6 \\ x_2 = 2 \end{cases}$，$z_{\max} = 36$

第十题

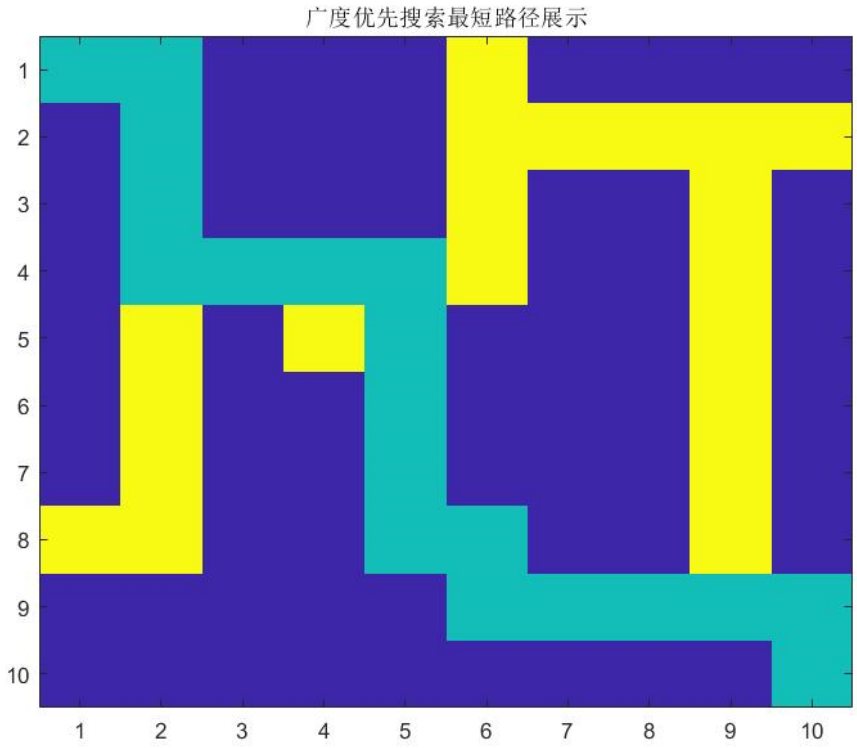
数学建模：

用矩阵 $Maze = m_{ij}$ 表示迷宫，矩阵元素为 1 表示该格可通行，元素为 0 表示该格是墙壁，利用广度优先的思想可以用于求解最短路径，广度优先即在选择时将所有分支放入备选项，并且，每一分支后续路线不能再经过先前的备选项，否则，前一次选择时可以直接到那一重复的备选项。用这样的方式走迷宫，可以达到“不走回头路的效果“，因此求得路径是最短。

求解该问题时，用队列的数据结构便可以实现广度优先搜索，每到一个新的格点 m_{ij}（出队），便向上下左右四个方向探索，可用 $m_{i-1,j}, m_{i+1,j}, m_{i,j+1}, m_{i,j-1}$ 来表示，若可行则入队等待接下来进行探索验证，同时还需要将 m_{ij} 标记为已访问过（元素置 0），这样就实现了上述基于广度优先算法的走迷宫。

运行程序后，输出最短路径长度结果以及最短路径展示如下，其中，最短路径图中，深蓝色代表不可通行，黄色代表可同行，蓝绿色则是经过广度优先搜索找出的最短路径展示：

```
>> hw2_10
最短路径长度为18
```



编程内容详见 hw2_10 文件夹中的程序

hw2_10.m

附加题一

得到输出结果如下：（注：本题采用 Floyd 算法，效率较低，但仍能体现动态规划的思想）

```
>> hw2_b1
点1到点7的最小代价为6,最短路径经过节点：
1
2
6
7
```

编程内容详见程序包中 hw2_b1 文件夹中的程序

hw2_b1.m