finalexam-part1

23307130428 姚馨悦

一、思路

1、编写函数babylonian(y, eps, x0=1.0)

```
def babylonian(y, eps, x0=1.0):
    x = x0  # 初始化为初始猜测值
    errors = [abs(x - math.sqrt(y))]  # 计算初始误差（与真实平方根的差）
    n = 0  # 迭代次数计数器

    # 当误差大于等于阈值时继续迭代
    while errors[-1] >= eps:
        x = 0.5*(x + y/x)  # 巴比伦算法迭代公式
        errors.append(abs(x - math.sqrt(y)))  # 计算当前误差
        n += 1  # 迭代次数加1
        if n > 1000:  # 安全检查：防止无限循环
            break
    return x, n, errors  # 返回最终结果、迭代次数和误差列表
```

- 算法原理: $x_{n+1} = (x_n + y/x_n)/2$
- 为了防止误差到达不了给定值，迭代超过1000次后自动退出循环

2、实验1：改变eps，（固定y, x0），绘制以n为横轴、errors为纵轴的图

```
eps_list = [1e-1,1e-2,1e-3,1e-4,1e-5,1e-6,1e-7,1e-8,1e-9,1e-10,1e-11,1e-12,1e-
13,1e-14,1e-15]
y=10  # 计算sqrt(10)
x0=1.0  # 初始猜测值为1.0

# 存储每个eps对应的迭代次数和最终误差
iterations=[]  # 存储迭代次数
final_errors=[]  # 存储最终误差

# 对每个误差阈值运行巴比伦算法
for eps in eps_list:
    x,n,errors = babylonian(y,eps,x0)
    iterations.append(n)  # 记录迭代次数
    final_errors.append(errors[-1])  # 记录最终误差

# 绘制迭代次数与最终误差的关系图
plt.figure()
plt.plot(iterations, final_errors, marker='o')  # 用圆点标记每个数据点

# 为每个数据点添加标签
texts = []
for n, err in zip(iterations, final_errors):
    texts.append(
```

```
        plt.text(n, err, f'({n}, {err:.1e})', fontsize=8)  # 显示(迭代次数, 误差)
    )

plt.yscale('log')  # y轴使用对数刻度，便于观察不同数量级的误差
plt.xlabel('n')  # x轴标签：迭代次数
plt.ylabel('error')  # y轴标签：误差

# 自动调整文本标签位置，避免重叠
adjust_text(texts,arrowprops=dict(arrowstyle='->', color='gray',
lw=0.5),verbose=0)

plt.show()  # 显示图形
```

- 对于不同eps，记录达到给定eps时的迭代次数（n，横轴）和误差值（final_errors，纵轴）
- 对于不同的eps，误差值达到给定eps时可能迭代次数相同，即有重叠标签，故标出具体坐标并通过自动调整文本标签位置来避免重叠

3、实验2：改变x0，（固定y, eps），绘制以n为横轴、errors为纵轴的图

```
y = 10  # 计算sqrt(10)
eps = 1e-12  # 固定误差阈值为10^-12
# 定义多个不同的初始猜测值
x0_list = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0,10000]

plt.figure(figsize=(7,5))  # 创建图形，设置大小为7x5英寸

# 对每个初始猜测值运行巴比伦算法
for x0 in x0_list:
    x, n, errors = babylonian(y, eps, x0)
    # 使用半对数坐标绘制误差随迭代次数的变化，观察收敛过程
    plt.semilogy(range(n+1), errors, marker='o', label=f'x0={x0:g}')

plt.xlabel('n')  # x轴标签：迭代次数
plt.ylabel('error')  # y轴标签：误差（对数刻度）
plt.title('Error convergence for different initial guesses $x_0$')  # 图标题
plt.grid(True, which='both', ls='--', alpha=0.3)  # 添加网格线，包括主网格和次网格
plt.legend()  # 显示图例，标识不同的x0值
plt.tight_layout()  # 自动调整布局，避免标签被截断
plt.show()  # 显示图形
```

- 对于不同x0，记录达到给定eps时的误差值（errors，纵轴，对数刻度）
- 横轴为迭代次数，纵轴为误差值

4、编写函数babylonian2(y, eps=1e-10, x0=1.0)

```
def babylonian2(y, eps=1e-10, x0=1.0):
    x = x0  # 初始化
    xs = [x]  # 保存所有x值
    errors=[]  # 保存相邻迭代间的误差
    n = 0  # 迭代计数器
```

```
    while True:
        x_new = 0.5*(x + y/x)  # 巴比伦算法迭代公式
        err = abs(x_new - x)  # 计算本次迭代与上次迭代的差值
        xs.append(x_new)  # 保存新的x值
        errors.append(err)  # 保存误差
        x = x_new  # 更新x为新值
        n += 1  # 迭代次数加1
        if err < eps or n > 1000:  # 如果误差足够小或迭代次数过多，则停止
            break
    return xs, errors  # 返回所有x值和误差列表
```

- 与babylonian()的区别：
  - 这个版本计算的是相邻两次迭代之间的差值 $|x_{n+1} - x_n|$而不是与真实值的差值 $|x_n - sqrt(y)|$
  - 返回值为迭代过程中的所有x值和误差值的数组

5、实验3：对于一组y, eps, x0, 使用babylonian2算法，绘制以n为横轴，errors为纵轴的图

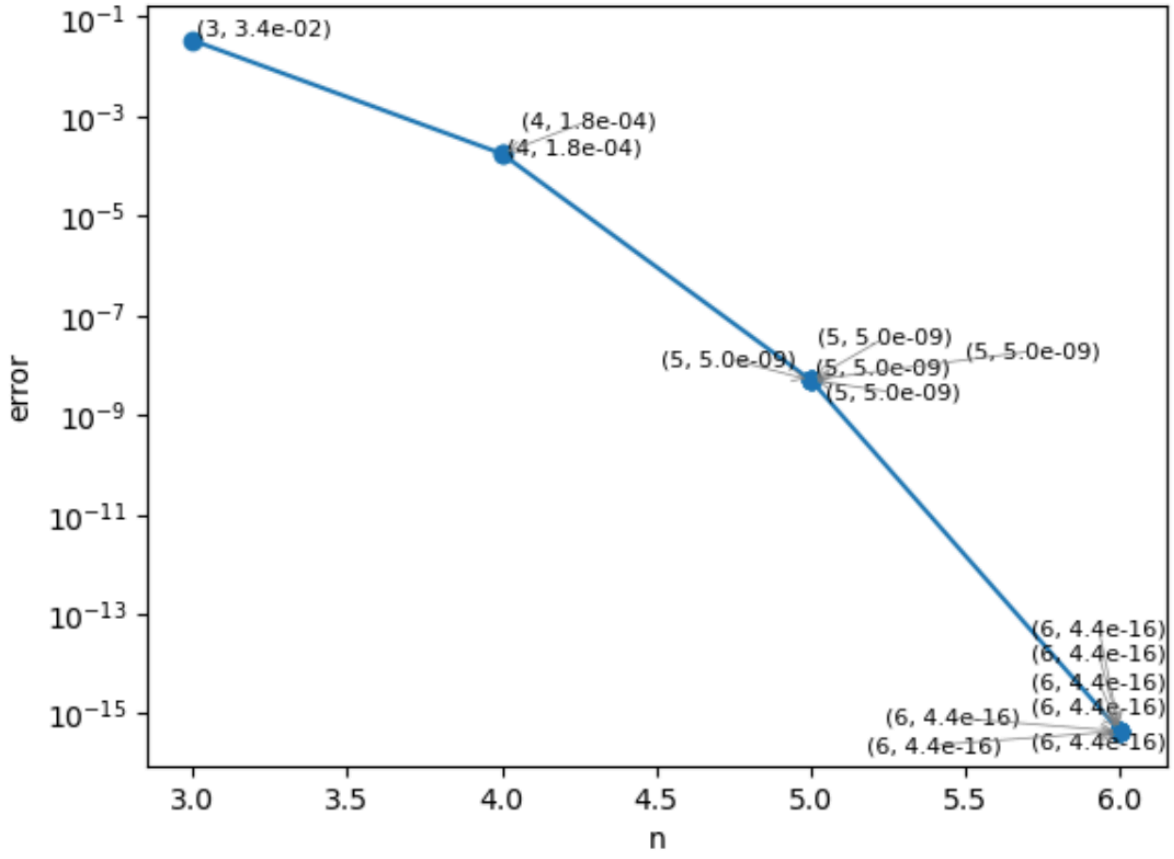
```
# 使用babylonian2计算sqrt(10)，误差阈值为10^-10，初始值为1.0
xs, errors = babylonian2(10,1e-10,1.0)

# 绘制相邻迭代间误差随迭代次数的变化
plt.figure()
plt.plot(errors, marker='o')  # 用圆点标记每个数据点
plt.yscale('log')  # y轴使用对数刻度
plt.xlabel('n')  # x轴标签：迭代次数
plt.ylabel('error')  # y轴标签：相邻迭代间的误差
plt.title('babylonian2 convergence')  # 图标题：babylonian2的收敛性
plt.show()  # 显示图形
```

- y轴使用对数坐标

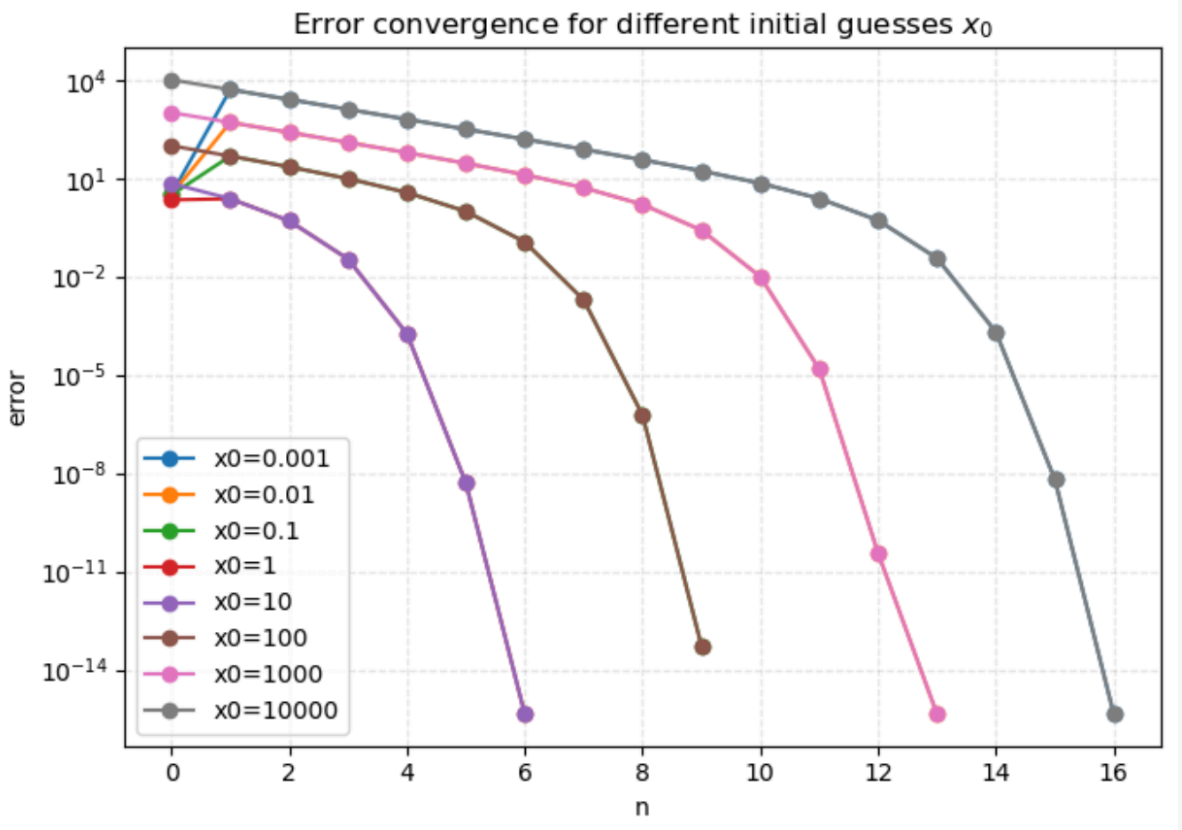
二、分析

实验1：改变eps，（固定y, x0），绘制以n为横轴、errors为纵轴的图 。分析errors和n的关系。



- n越大，errors越小。即迭代次数越多，最终得到的x与$\sqrt{y}$越接近。
- 默认eps越小，最终得到的x与$\sqrt{y}$越接近，符合直觉。

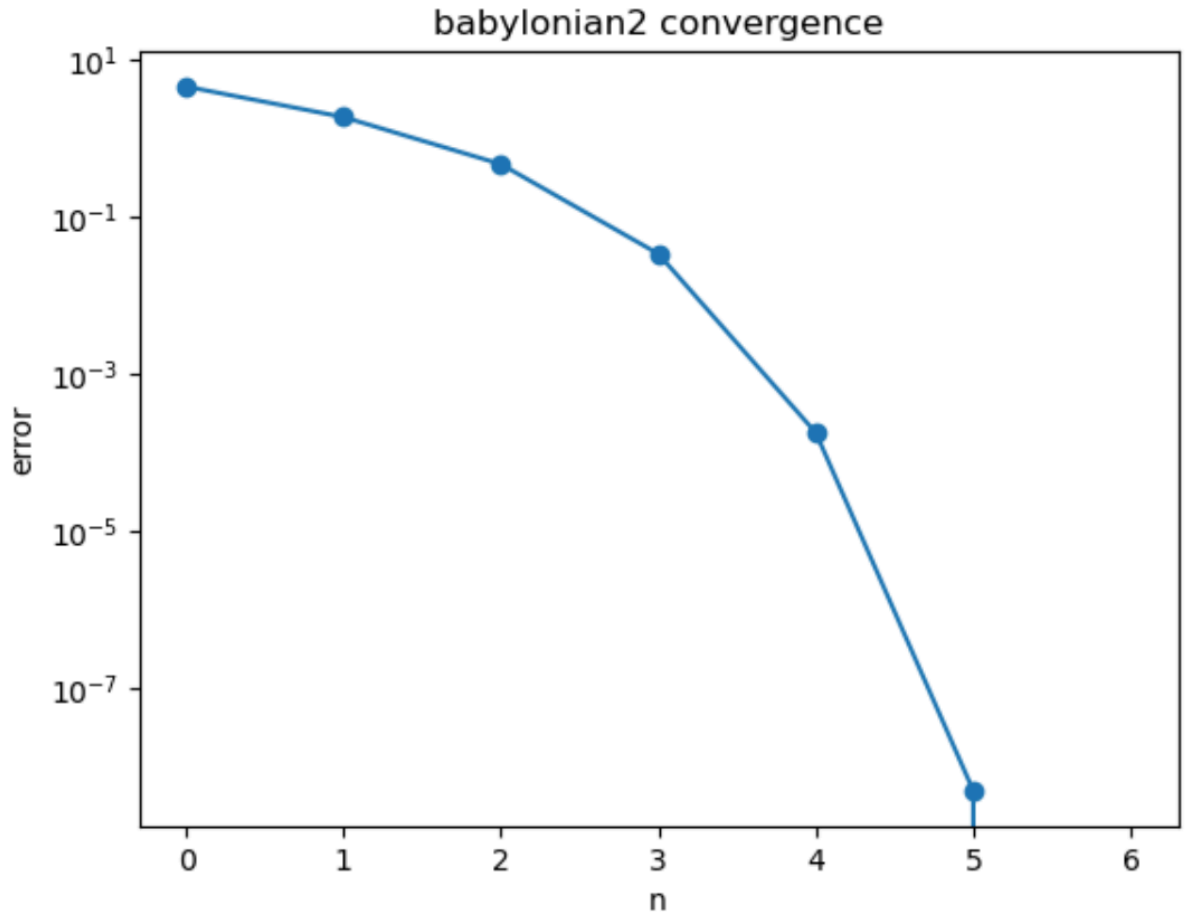
实验2：改变x0，（固定y, eps），绘制以n为横轴、errors为纵轴的图。讨论x0对收敛曲线的影响。



- $\max\left(\frac{x_0}{y}, \frac{y}{x_0}\right)$越大，收敛越慢。

- $\max\left(\frac{x_0}{y}, \frac{y}{x_0}\right)$相同，除初始误差外，收敛曲线相同。

实验3：对于一组y, eps, x0, 使用babylonian2算法，绘制以n为横轴，errors为纵轴的图。分析errors和n的关系。



- n越大，errors越小。即迭代次数越多，x的值的变化幅度越小，直到误差达到小于给定的eps。

三、运行说明

- 顺序运行ipynb文件代码块即可。