《地下水动力学》实验指导书

杨国勇

（中国矿业大学　水文与水资源系）

2023 年 12 月 5 日

1　搭建自己的 Python 编程环境

Python 是一种面向对象、解释型计算机程序设计语言，其语法简洁而清晰，易于上手。尤其是具有丰富和强大的类库，常被昵称为胶水语言，能够把其他语言制作的各种模块轻松地联结在一起。Python 非常适合做科学计算，相应的科学计算软件库齐全，如 NumPy，SciPy，Matplotlib 等。

根据《地下水动力学》课程实验的内容，简要介绍 Python 编程环境的构建及程序设计用到的相关库。

1.1　创建并激活 Python 虚拟环境

Python 的 Window 发行版可从 `https://www.python.org/downloads/windows` 下载。可根据自己需要，下载 32-bit、64-bit 版本以及帮助文件。

安装程序提供了一些默认选项，用户可根据需要修改这些选项。安装完成后可以创建多个 Python 虚拟环境满足不同的需求。

在指定文件夹中创建 Python 虚拟环境

用 `Win + R` + `cmd` 打开 Windows 终端，，切换到要保存 Python 虚拟环境的位置（子目录）。例如，要将 Python 虚拟环境保存到 D 的 Python39 子目录下，Windows 终端行键入如下命令：

```
D:（回车）
python -m venv Python39（回车）
```

操作系统在 D 盘的根目录下建立名称为 Python39 的子目录，该目录及其子目录包含了虚拟环境需要的文件。运行该目录下 scripts 子目录里的批处理文件激活虚拟环境：

```
python39\scripts\activate.bat
```

退出 Python 虚拟环境后重新进入，需要再次运行 activate.bat。若想更方便的进入虚拟环境，建议在桌面建立命令提示符快捷方式，打开快捷方式的属性页，将“目标”改为 `%windir%\system32\cmd.exe /K D:\python39\scripts\activate.bat`，“起始位置”改为 `d:\python39`。

使用 Pip 工具在虚拟环境中安装所需的库

Pip 是一个 Python 内置的库管理系统。Pip 可以安装、更新或删除任何正式的库。Pip 通过访问 Python 库索引服务器 PyPI（`https://pypi.org`）实现版本控制。因为 PyPI 的主服务器在国外，国内用户使用 Pip 安装 Python 库时经常出现超时错误，访问起来非常不便，使用国内镜像站点可避免此类错误。

以下是常用的 PyPI 国内镜像：

- 清华：https://pypi.tuna.tsinghua.edu.cn/simple
- 阿里云：http://mirrors.aliyun.com/pypi/simple/
- 中国科技大学 https://pypi.mirrors.ustc.edu.cn/simple/
- 华中理工大学：http://pypi.hustunique.com/
- 山东理工大学：http://pypi.sdutlinux.org/
- 豆瓣：http://pypi.douban.com/simple/

如下为 Pip 常用的命令：

- 升级 pip：`pip install --upgrade pip` 或 `python -m pip install --upgrade pip`
- 查看 pip 版本号：`pip --version`
- 用 pip 安装指定库：`pip install some-package`
- 用 pip 卸载某一库：`pip uninstall some-package`
- 用 pip 升级某一库：`pip install some-package --upgrade` 或 `pip install -U some-package`
- 查看所有可以升级的库：`pip list --outdated`
- 查看已安装库的版本号：`pip freeze`
- 从 PyPI 镜像安装库：`pip install -i https://pypi.tuna.tsinghua.edu.cn/simple some-package`
- 将 PyPI 镜像设为默认:`pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple`

如果需要升级的库数量很多，可用 pip-review 工具。

- 安装 pip-review：`pip install pip-review`
- 升级全部库：`pip-review --local --interactive`

实验用到的库可用如下命令一起安装：

```
pip install Jupyter Jupyterlab numpy scipy matplotlib ipympl mpl_interactions
```

库简介：

- IPython：一个非常流行的 Python 解释器，也是 Python 科学计算和交互可视化的最佳平台。
- NumPy：Python 开源科学计算基础软件库，其 N 维数组对象 ndarray 能够快速高效的处理多维数组。
- SciPy：Python 开源科学计算库，涵盖了科学计算中的各种工具，包括统计、积分、插值、最优化，图像处理等。
- Matplotlib：Python 的 2D 绘图库，可以绘制线图、散点图、等高线图、条形图、柱状图、3D 图形等。
- ipympl：为 Jupyter 环境开发提供交互式后端，增强了 Jupyter 环境下的交互式绘图功能。
- IPyWidgets：为 Jupyter Notebook 提供交互功能的工具。若安装了 Jupyter ，则 IPyWidgets 也被默认安装。
- mpl_interactions：另一个在 Jupyter Notebook 中实现交互式绘图功能的工具。

1.2 Jupyter Notebook 与 Jupyter Lab

Jupyter Notebook 是基于网页用于交互计算的应用，可以在页面的 `cell` (单元) 中直接编写和运行代码，运行结果直接显示在代码块下方。

JupyterLab 是基于 Web 用户界面的 Jupyter 项目。JupyterLab 使用户能够以灵活、集成和可扩展方式处理文档。JupyterLab 由同一台服务器提供服务，使用与 Jupyter Notebook 相同的笔记本文档格式。实验中的示例以 Notebook (.ipynb) 形式提供。

启动 Jupyter Notebook：`jupyter notebook`

启动 Jupyterlab：`jupyter lab` 或 `jupyter-lab`

`Jupyter Notebook` 或 `Jupyter Lab` 在浏览器中自动打开。若不成功，可以直接将 Windows 终端显示的 `http://localhost:8888/lab?token=...` 或 `http://127.0.0.1:8888/lab?token=...` 粘贴到浏览器地址栏打开。`Jupyter lab` 服务启动后可以在该启动位置目录下建立、打开 Notebook 文件。若想使用其他路径下的 Notebook 文件，应先在 Windows 终端用 `CD` 命令切换到该路径，然后再启动 `Jupyter Lab` 服务。

在 `notebook` 的 `cell` (单元) 有 3 种格式，即 Code、Markdown、Raw。Code 单元中可以编写代码，Markdown 单元以 Markdown 语法书写文本并渲染，Raw 单元以原生方式显示输入内容。单元可用 `Run` 按钮或 `Shift + Enter` 快捷键运行。

Code 单元中可用 `matplotlib` 库绘图。`matplotlib` 的 `frontend` 就是我们写的 `python` 代码，而 `backend` 就是负责显示图形的底层代码。`backend` 跟具体硬件和显示条件有关，`%matplotlib ipympl` 魔法命令就是指定后端为 `ipympl` 并提供交互绘图的功能，`%matplotlib widget` 具有同样的效果。其他与 `matplotlib` 有关的魔法命令可用 `%matplotlib --list` 查看。

NoteBook 按 Code 单元的顺序运行代码，前面的编号就是运行顺序号。变量的作用域是全局的，若某个单元定义了一个变量，只要该代码执行过，则不管单元的顺序，其他单元都可以存取该变量。因此，有时为避免错误，需要重启 `kernel` (内核)，再按从上到下的次序运行单元中的代码。

2 实验 1：单井稳定流经验公式求水文地质参数

影响半径的经验公式

吉哈尔特（承压水）：

$$R = 10 s_w \sqrt{K}$$

库萨金（潜水）：

$$R = 2 s_w \sqrt{K H_0}$$

式中，K — 渗透系数，m/d；s — 设计降深，m；H_0 — 自底板算起的含水层静止水位（厚度），m。

由于这两个公式为经验公式，若变量取其他单位，则公式形式也会有差别。单孔抽水试验可由 Dupuit 公式与经验公式构造迭代表达式计算 K 及 R：

承压井：

$$\begin{cases} K &= \frac{Q}{2\pi M s_w} \ln \frac{R}{r_w} \\ R &= 10 s_w \sqrt{K} \end{cases}$$

潜水井：

$$\begin{cases} K &= \frac{Q}{\pi(2H_0 - s_w)s_w} \ln \frac{R}{r_w} \\ R &= 2 s_w \sqrt{K H_0} \end{cases}$$

Python 程序

推荐使用 NumPy 库，使用别名 `np` 导入：`import numpy as np`。

NumPy 进行数组运算非常有效，索引从 0 开始。`numpy.array` 将 Python 列表转化为 NumPy 数组；`numpy.ones`、`numpy.zeros` 用于构造元素为 1 或 0 的数组。常用的数学函数在 NumPy 中都有对应的形式，如 `numpy.sin`、`numpy.abs`、`numpy.exp`、`numpy.float_power`、`numpy.log10` 等。

`numpy.vectorize` 可将自定义函数向量化。向量化的函数可以接受向量参数，并以向量返回结果。

`numpy.set_printoptions` 可以设置数组的显示格式。如 `numpy.set_printoptions(precision=4)` 显示 4 位小数位；`numpy.set_printoptions(formatter={'float': '{:.2f}'.format})` 按浮点数显示 2 位小数位。

以承压水为例，定义一个子程序 `empirical`，先设定 K 的初始值，用经验公式计算 R，再循环迭代计算。当两次计算结果误差绝对值小于设定的误差限时终止计算。

例 1：某承压含水层厚度 16.5 m。有 3 眼完整井进行稳定流抽水试验，试根据表中数据求渗透系数 K 及影响半径 R（精度要求 1e-4）。

表稳定流抽水试验数据

序号	抽水井半径 (m)	抽水量 (m^3/d)	抽水井降深 (m)
1	0.4	320.54	1.16
2	0.4	421.63	1.60
3	0.4	536.54	1.90

自定义一个迭代计算函数：

[1]:
```
import numpy as np


# 定义函数
```

```python
def empirical(rw, M, Q, sw):
    K0 = 10
    R0 = 10*sw*np.sqrt(K0)
    while True:
        R = 10*sw*np.sqrt(K0)
        K = 0.5*Q*np.log(R/rw)/M/sw/np.pi
        if np.abs(R - R0) < 1.0e-4 and np.abs(K - K0) < 1.0e-4:
            break
        else:
            K0 = K
            R0 = R
    return K, R
```

由于函数没有向量化，一次只能计算一组数据：

[2]:
```python
rw = 0.4           # 井半径, m
M = 16.5           # 含水层厚度, m
Q = 320.54         # 抽水量, m^3/d
sw = 1.16          # 降深, m

K, R = empirical(rw, M, Q, sw)

print('K = {:.2f} m^2/d'.format(K))
print('R = {:.2f} m'.format(R))
```

```
K = 12.32 m^2/d
R = 40.72 m
```

函数向量化：计算多次抽水试验数据，将 empirical 向量化非常方便。

[3]:
```python
vempirical= np.vectorize(empirical) # 将 empirical 函数向量化

rw = np.ones(3)*0.4                     # 井半径向量, m
M = np.ones(3)*16.5                     # 含水层厚度向量, m
Q = np.array([320.54, 421.63, 536.54])  # 抽水量向量, m^3/d
s = np.array([1.16, 1.60, 1.90])        # 降深值向量, m

K, R = vempirical(rw, M, Q, s)

np.set_printoptions(formatter={'float': '{:.2f}'.format})

print('K = {} m^2/d'.format(K))
print('R = {} m'.format(R))
```

```
K = [12.32 12.60 14.12] m^2/d
R = [40.72 56.79 71.40] m
```

练习 1：某潜水含水层完整井稳定流抽水试验，已知潜水面最初水位高度 43.6 m，抽水孔半径 0.15 m，降深 2.8 m，稳定抽水量 2380 m^3/d。求潜水含水层渗透系数 K 及影响半径 R（精度要求 1e-4）。

3 实验 2：交互式配线法求水文地质参数

井流模型的解大都形如

$$\frac{4\pi Ts}{Q} = W(u, \cdots)$$

$W(u, \cdots)$ 为无量纲的标准井函数。根据 $s - W(u, \cdots)$ 与 $t - \frac{1}{u}$ 的关系，在双对数刻度下，观测的 (s, t) 数据与井函数标准曲线形状相同，根据匹配后的坐标位移量可以求出相应的水文地质参数。

与 Theis 标准曲线不同，Hantush - Jacob 标准曲线有多条，每一条标准曲线对应不同的越流因素 B。

绘制标准曲线可以用教材中的标准函数表，也可用指导书附录的 Theis、Hantush - Jacob 井函数程序计算。

3.1 Theis 公式的配线法

Theis 公式及 u 取对数

$$\lg s + \lg \frac{4\pi T}{Q} = \lg W(u)$$

$$\lg \frac{t}{r^2} + \lg \frac{4T}{S} = \lg \frac{1}{u}$$

可以看出，在双对数坐标系中，$s - \frac{t}{r^2}$ 的形状与 $W(u) - \frac{1}{u}$ 一致，根据垂直位移可以计算 T，根据水平位移与 T 可以计算 S。

记垂直位移为 $\Delta \lg(s) = \lg \frac{4\pi T}{Q}$，水平位移为 $\Delta \lg \frac{t}{r^2} = \lg \frac{4T}{S}$，参数计算公式为

$$T = \frac{Q}{4\pi} 10^{\Delta \lg(s)}, \quad S = 4T \ 10^{-\Delta \lg \frac{t}{r^2}}$$

这种方式相当于标准曲线固定，移动 $s - \frac{t}{r^2}$ 与标准曲线匹配。

上面的配线关系可以用于单观测孔数据，也可用多观测孔数据，编写的程序上通用性好，只是数据准备有所差别。

考虑数据处理的一致性及对数刻度显示，时间单位取 “min” (分)，降深单位取 “m” (米)。

程序设计流程为“库导入 $\Longrightarrow$ 数据准备 $\Longrightarrow$ 绘图准备 $\Longrightarrow$ `widgets.interact` 互动”，特殊说明见注释。

例 2：承压含水层定流量非稳定流群孔抽水试验，抽水量为 60 m^3/h，观 1、观 2、观 3 与观 4 孔距抽水孔分别 43 m、140 m、510 m、780 m，各观测孔水位降深如下表所示。试用观 1 孔数据求含水层水文地质参数。

表群孔抽水试验数据

累计抽水时间 (min)	观 1 降深 (m)	观 2 降深 (m)	观 3 降深 (m)	观 4 降深 (m)
10	0.73	0.16	0.04	
20	1.28	0.48		
30	1.53	0.54		
40	1.72	0.65	0.06	
60	1.96	0.75	0.20	
80	2.14	1.00	0.20	0.04
100	2.28	1.12	0.20	
120	2.39	1.22	0.21	0.08
150	2.54	1.36	0.24	0.09
210	2.77	1.55	0.40	0.16
270	2.99	1.70	0.53	0.25
330	3.10	1.83	0.63	0.34
400	3.20	1.89	0.65	0.42
450	3.26	1.98	0.73	0.50
645	3.47	2.17	0.93	0.71
870	3.68	2.38	1.14	0.87
990	3.77	2.46	1.24	0.96
1185	3.85	2.54	1.35	1.06

Python 程序

程序中标准井函数用 `scipy.special.exp1` 计算，`ipywidgets` 控制位移。

导入库：

[4]:
```
%matplotlib ipympl

# 导入库
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import exp1    # 计算 Theis 井函数
import ipywidgets as widgets      # 导入交互绘图模块

# 控制小数的显示精度
np.set_printoptions(precision=4)
```

```python
# 使不用 Unicode 负号，防止出现乱码
plt.rcParams['axes.unicode_minus']=False
```

准备数据：

[5]:
```python
Q = 60 / 60           # 抽水量, m^3/min
r = 43                # 观测孔距离, m

# 观测时间, min; np.array() 将 list 转换为 numpy 数组
t = np.array([10, 20, 30, 40, 60, 80, 100, 120, 150, 210,
              270, 330, 400, 450, 645, 870, 990, 1185])
# 观测孔降深, m; 用 np.array() 将 list 转换为 numpy 数组
s = np.array([0.73, 1.28, 1.53, 1.72, 1.96, 2.14, 2.28, 2.39, 2.54, 2.77,
              2.99, 3.10, 3.20, 3.26, 3.47, 3.68, 3.77, 3.85])

# 计算 t/r^2
tr2 = t / r**2
```

定义一个绘图函数：

这个函数在后面介绍的多孔求参方法也可使用。注意到 `matplotlib.pyplot` 对象全局适用，因此例程中需要用代码清除上次绘制的内容，这样才不会出错。程序大部分代码都是设置图形的，如图形界限、坐标刻度、图例等。`ax.plot(x, exp1(1/x), label=" 标准曲线")` 绘标准曲线，x 变量是在一个对数坐标格中均匀取值；`ax.plot(tr2*np.float_power(10, delta_x), s*np.float_power(10, delta_y), '*', label=" 观测值")` 绘制平移后的观测数据散点图。

[6]:
```python
# 参数 delta_x, delta_y 分别为水平, 垂直位移
def Theis_fit(delta_x, delta_y):

    global Q, tr2, s # 设置全局变量, 值可以穿透程序

    # Clear the previous plot
    for i in plt.get_fignums():
        plt.gca()
        ax.cla()

    # 设置作图界限
    ymin = 1.0e-2
    ymax = 10.0
    xmin = 1.0e-1
    xmax = 1.0e4
    ax.set_xlim(xmin, xmax)
    ax.set_ylim(ymin, ymax)
```

```
# 设置坐标轴刻度为常用对数
ax.set_xscale("log")
ax.set_yscale("log")

# 设置网格为正方形
ax.set_aspect(1)
ax.grid(True)

# 坐标轴标签
# r 字符串前缀转义字符将被忽略，'$\log 1/u$' 显示为 latex 数学符号
plt.xlabel(r'$\log 1/u$')
plt.ylabel(r'$W(u)$')

# 坐标加密可以绘出光滑的标准曲线
ix = np.linspace(-1, 4, 51)
x = np.float_power(10, ix)      # 1/u 的值
# 绘制标准曲线
ax.plot(x, exp1(1/x), label="标准曲线")

# 按参数 dlogs, dlogtr2 计算 T 与 S
T = np.float_power(10, delta_y)*Q/4/np.pi
S = np.float_power(10, -delta_x)*4*T

# 绘制平移后的散点图形
ax.plot(tr2*np.float_power(10, delta_x), s*np.float_power(10, delta_y),
        '*', label="观测值")

# 指定图例标题与内容的字体
plt.legend(prop={'family':'Simsun'}, loc=4, title="图例",
           title_fontproperties={'family':'SimHei'})
# 指定绘图标题为楷体
ax.set_title("配线法", fontproperties={'family':'KaiTi', 'size':'large'})

# 显示图形
plt.show()

# 输出参数
print('  T = {:.4f} m^2/min'.format(T))
print('  S = {:.4e}'.format(S))
```

交互绘图：

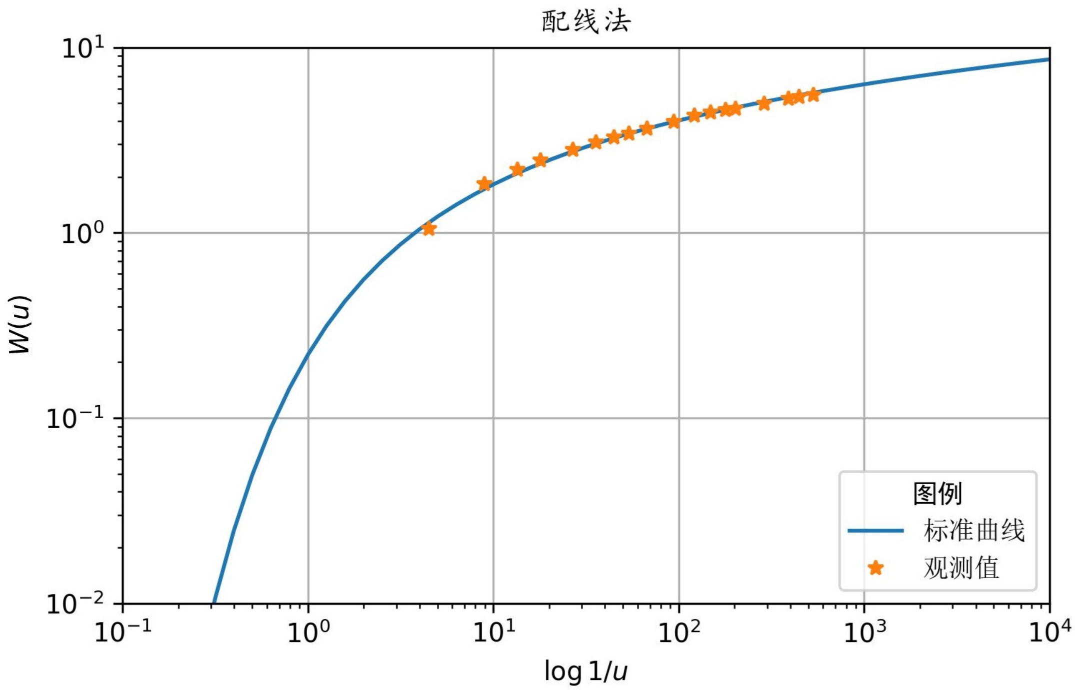
widgets.FloatSlider 用于控制位移量，参数有初始值、最小、最大值，标签等，可参考帮助文档。

初始位移可以设为 0，相当于参数取 $T=\frac{Q}{4\pi}$，$S=4T$。为了使观测数据尽可能靠近标准曲线，可以选取 2 组观测数据按 Jacob 公式计算参数。这样处理可使观测数据充分靠近标准曲线，位移界限可以设置为半个坐标格。

[7]:
```python
# 为使观测数据尽可能靠近标准曲线，最简单方法是取中间相邻两点用 Jacob 公式计算初始参数
# 然后再计算位移

i1 = int(len(tr2)/2)
i2 = i1 + 1
slope = (s[i1]-s[i2])/np.log10(tr2[i1]/tr2[i2])

T = 0.183*Q/slope
S = 2.25*T*tr2[i1]*np.float_power(10, -s[i1]/slope)

# 计算初始平移量
delta_x = np.log10(4*T/S)
delta_y = np.log10(4*np.pi*T/Q)

# 设置位移界限为半个坐标格
delta_xmin = delta_x - 0.5
delta_xmax = delta_x + 0.5
delta_ymin = delta_y - 0.5
delta_ymax = delta_y + 0.5

# 设置 widgets.FloatSlider 参数，包括最小值、最大值、步长、标签、显示格式等
delta_y = widgets.FloatSlider(
        value=delta_y, min=delta_ymin, max=delta_ymax, step=0.05,
        description=r"$\Delta \lg s$ :", continuous_update=False,
        readout_format='.2f', disabled=False)
delta_x = widgets.FloatSlider(
        value=delta_x, min=delta_xmin, max=delta_xmax, step=0.05,
        description=r"$\Delta \lg\frac{t}{r^2}$ :", continuous_update=False,
        readout_format='.2f', disabled=False)

plt.style.use('default')         # 设置图形风格，如可以设为 'seaborn-deep'
fig, ax = plt.subplots(dpi=100)  # 想获得高品质图形，可以设 dpi=300

# ipywidgets 小部件控制参数实现互动。ipywidgets 有缓冲功能，
# 同一个 Notebook 复制的代码得不到所期望的结果
widgets.interact(Theis_fit, delta_y=delta_y, delta_x=delta_x);
```

$\Delta \lg \frac{t}{r^2}$: 2.92

$\Delta \lg s$: 0.16




```
T = 0.1143 m^2/min
S = 5.5115e-04
```

[8]:
```
fig.savefig("2.1.jpg")
# plt.close("all")
# widgets.FloatSlider.close_all()
```

例 3：根据 **例 2** 的观 1、观 2 数据求含水层水文地质参数。

多个数组拼接可用函数 np.concatenate() 。

准备数据：

[9]:
```
# 输入数据
Q = 60/60            # m^3/min

# 观 1
r1 = 43
t1 = np.array([10, 20, 30, 40, 60, 80, 100, 120, 150, 210,
              270, 330, 400, 450, 645, 870, 990, 1185])
# 观测孔降深, m; 用 np.array() 将 list 转换为 numpy 数组
s1 = np.array([0.73, 1.28, 1.53, 1.72, 1.96, 2.14, 2.28, 2.39, 2.54, 2.77,
               2.99, 3.10, 3.20, 3.26, 3.47, 3.68, 3.77, 3.85])
```

```
# 观 2
r2 = 140       # m
t2 = np.array([10, 20, 30, 40, 60, 80, 100, 120, 150, 210,
               270, 330, 400, 450, 645, 870, 990, 1185])
s2 = np.array([0.16, 0.48, 0.54, 0.65, 0.75, 1.00, 1.12, 1.22, 1.36, 1.55,
               1.70, 1.83, 1.89, 1.98, 2.17, 2.38, 2.46, 2.54])

# 数组拼接
tr2 = np.concatenate((t1/r1**2, t2/r2**2))
s = np.concatenate((s1, s2))  # 数组 s 为对应的降深
```

交互绘图：

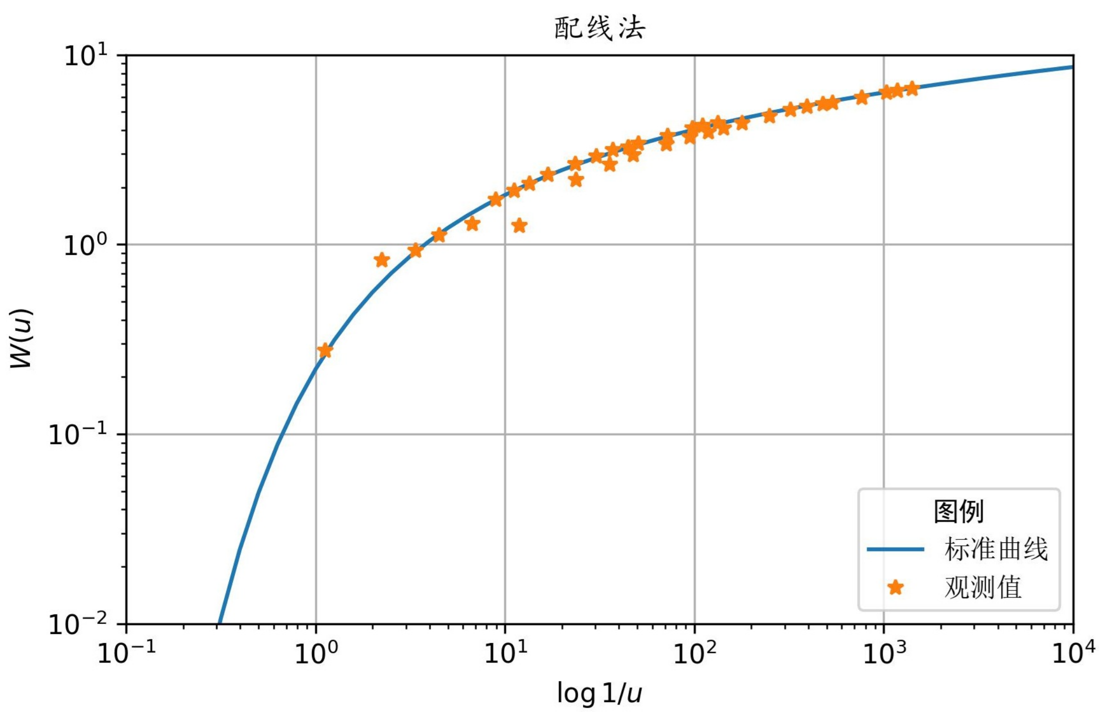
widgets.FloatSlider 用于控制位移量。为了使观测数据尽可能靠近标准曲线，可以选取 2 组观测数据按 Jacob 公式计算参数。这样处理可使观测数据充分靠近标准曲线，位移界限可以设置为半个坐标格。

[10]:
```
# 为使观测数据尽可能靠近标准曲线，最简单方法是取中间相邻两点用 Jacob 公式计算初始参数
# 然后再计算位移

i1 = int(len(tr2)/2)
i2 = i1 + 1
slope = (s[i1] - s[i2])/np.log10(tr2[i1]/tr2[i2])

T = 0.183*Q/slope
S = 2.25*T*tr2[i1]*np.float_power(10, -s[i1]/slope)

# 计算初始平移量
delta_x = np.log10(4*T/S)
delta_y = np.log10(4*np.pi*T/Q)

# 设置位移界限为半个坐标格
delta_xmin = delta_x - 0.5
delta_xmax = delta_x + 0.5
delta_ymin = delta_y - 0.5
delta_ymax = delta_y + 0.5

# 设置 widgets.FloatSlider 参数，包括最小值、最大值、步长、标签、显示格式等
delta_y = widgets.FloatSlider(
        value=delta_y, min=delta_ymin, max=delta_ymax, step=0.05,
        description=r"$\Delta \lg s$ :", continuous_update=False,
        readout_format='.2f', disabled=False)
```

```
delta_x = widgets.FloatSlider(
        value=delta_x, min=delta_xmin, max=delta_xmax, step=0.05,
        description=r"$\Delta \lg\frac{t}{r^2}$ :", continuous_update=False,
        readout_format='.2f', disabled=False)

plt.style.use('default')   # 设置图形风格，如可以设为 'seaborn-deep'
fig, ax = plt.subplots(dpi=100)  # 想获得高品质图形，可以设 dpi=300

# ipywidgets 小部件控制参数实现互动。ipywidgets 有缓冲功能，
# 同一个 Notebook 复制的代码得不到所期望的结果
widgets.interact(Theis_fit, delta_y=delta_y, delta_x=delta_x);
```

$\Delta \lg \frac{t}{r^2}$: 3.34

$\Delta \lg s$: 0.24




```
T = 0.1367 m^2/min
S = 2.4850e-04
```

[11]:
```
fig.savefig("2.2.jpg")
# plt.close("all")
# widgets.FloatSlider.close_all()
```

练习 2：根据 **例 2** 的观测数据，分别用单孔、多孔数据求参，并比较求参结果。

思考题：

为什么用多孔的数据效果并不理想？多个求参结果如何取舍？

3.2 Hantush - Jacob 公式的配线法

Hantush - Jacob 公式的配线法与 Theis 公式配线法原理相同。唯一区别就是根据某一特定曲线求越流因素。以单孔求参为例，对

$$s \cdot \frac{4\pi T}{Q} = W\left(u, \frac{r}{B}\right), \quad t \cdot \frac{4T}{r^2 S} = \frac{1}{u}$$

两边取对数

$$\lg s + \lg \frac{4\pi T}{Q} = \lg W\left(u, \frac{r}{B}\right), \quad \lg t + \lg \frac{4T}{r^2 S} = \lg \frac{1}{u}$$

设计思路

在标准曲线坐标系中移动 $s \sim t$ 散点图，根据平移量计算 (T, S)。记坐标平移量为

$$\Delta \lg(s) = \lg \frac{4\pi T}{Q}, \quad \Delta \lg(t) = \lg \frac{4T}{r^2 S}$$

按如下公式计算参数 T 与 S：

$$T = \frac{Q}{4\pi} 10^{\Delta \lg(s)}, \quad S = \frac{4T}{r^2} 10^{-\Delta \lg(t)}$$

输入参数 $\beta = \frac{r}{B}$ 并绘制对应的 Hantush - Jacob 标准曲线，以最佳匹配结果确定 β。

例 4：对某承压含水层抽水孔进行 12 h 的定流量非稳定流抽水，抽水量为 528 m^3/d，距抽水孔 90 m 处有一观测孔，观测数据如下表，试确定水文地质参数。

表承压含水层抽水试验观测数据

序号	时间 (min)	降深 (cm)	序号	时间 (min)	降深 (cm)
1	1	2.5	9	50	24.7
2	2	3.9	10	60	26.4
3	4	6.1	11	90	30.4
4	6	8.0	12	120	33.0
5	9	10.6	13	150	35.0
6	20	16.8	14	360	42.6
7	30	20.0	16	550	44.0
8	40	22.6	16	720	44.5

Python 程序

导入库：

[12]:
```
%matplotlib ipympl

# 导入库
import math as math
import matplotlib.pyplot as plt
import numpy as np
import scipy.special as sps      # 计算 hantush - jacob 井函数用到 Bessel
from scipy.special import exp1   # 计算 Theis 井函数
import ipywidgets as widgets     # 导入交互绘图模块

# 控制小数的显示精度
np.set_printoptions(precision=4)
# 使不用 Unicode 负号，防止出现乱码
plt.rcParams['axes.unicode_minus']=False
```

定义 hantush - jacob 井函数，并将其向量化：

[13]:
```
# 计算 hantush - jacob 井函数程序
def hantush_jacob(u, beta):
    """
    指数积分级数方法计算 hantush - Jacob 井函数.
        u = r^2 S/(4Tt);
        beta = r/B;
        u, beta 可以为数组并返回数组。
    """
    def wellfunc(u, beta):
        if u < 0:
            print('Negative are not allowed')
            return np.nan
        if u == 0:                      # 稳定解
            return 2.0*sps.k0(beta)

        r = 1
        t = beta**2/(4*u)
        b = 2*u

        if beta <= b:        # beta < 2u
            W = 0
            n = 0
```

```
            term = r*sps.expn(n + 1, u)
            while np.abs(term) > 1e-10:
                W = W + term
                n = n + 1
                r = r*(-t)/n
                term = r*sps.expn(n + 1, u)
        else:
            W = 2.0*sps.k0(beta)
            n = 0
            term = r*sps.expn(n + 1, t)
            while np.abs(term) > 1e-10:
                W = W - term
                n = n + 1
                r = r*(-u)/n
                term = r*sps.expn(n + 1, t)

        return W

    # 向量化函数
    well = np.vectorize(wellfunc)
    # 返回值
    return 1.0*well(u, beta)
```

数据准备：

[14]:
```
# 准备数据
Q = 528/1440   # m^3/min
r = 90 # m
t = np.array([1, 2, 4, 6, 9, 20, 30, 40, 50, 60,
              90, 120, 150, 360, 550, 720])             # min
s = np.array([2.5, 3.9, 6.1, 8.0, 10.6, 16.8, 20.0, 22.6, 24.7, 26.4,
              30.4, 33.0, 35.0, 42.6, 44.0, 44.5])/100  # m
```

定义一个绘图函数：

注意到 `matplotlib.pyplot` 对象全局适用，因此例程中需要用代码清除上次绘制的内容，这样才不会出错。程序大部分代码都是设置图形的，如图形界限、坐标刻度、图例等。`ax.plot(x, exp1(1/x), label="Theis")` 绘 Theis 标准曲线，`ax.plot(x, hantush_jacob(1/x, beta), label="Hantush-Jacob")` 绘对应 beta 的 Hantush-Jacob 标准曲线，x 变量是在一个对数坐标格中均匀取值；`ax.plot(t*np.float_power(10, delta_x), s*np.float_power(10, delta_y) , '*', label=" 观测值")` 绘制平移后的观测数据散点图。

[15]:
```python
# 定义一个绘图函数
# 参数 delta_x, delta_y 分别为水平，垂直位移，beta = r/B
def Hantush_Jacob_fit(delta_x, delta_y, beta):
    global t, s, Q, r    # 设置全局变量，值可以穿透程序

    # Clear the previous plot
    for i in plt.get_fignums():
        plt.gca()
        ax.cla()

    # 设置作图界限
    ymin = 1.0e-2
    ymax = 10.0
    xmin = 1.0e-1
    xmax = 1.0e4
    ax.set_xlim(xmin, xmax)
    ax.set_ylim(ymin, ymax)

    # 设置坐标轴刻度为常用对数
    ax.set_xscale("log")
    ax.set_yscale("log")

    # 设置网格为正方形
    ax.set_aspect(1)
    ax.grid(True)

    # 坐标轴标签
    # r 字符串前缀转义字符将被忽略，'$\log 1/u$' 显示为 latex 数学符号
    plt.xlabel(r'$\log 1/u$')
    plt.ylabel(r'$W(u)$')

    # 坐标加密可以绘出光滑的标准曲线
    ix = np.linspace(-1, 4, 51)
    x = np.float_power(10, ix)       # 1/u 的值
    # 绘制标准曲线
    ax.plot(x, exp1(1/x), label="Theis")
    ax.plot(x, hantush_jacob(1/x, beta), label="Hantush-Jacob")

    # 计算 T, S, u
    T = Q/4/np.pi*np.float_power(10, delta_y)
    S = 4*T/r**2*np.float_power(10, -delta_x)
```

```
    # 绘制平移后的散点图形
    ax.plot(t*np.float_power(10, delta_x), s*np.float_power(10, delta_y),
            '*', label="观测值")

    # 指定图例标题与内容的字体
    plt.legend(prop={'family':'Simsun'}, loc=4, title="图例",
               title_fontproperties={'family':'SimHei'})
    # 指定绘图标题为楷体
    ax.set_title("配线法", fontproperties={'family':'KaiTi', 'size':'large'})

    # 显示图形
    plt.show()

    print('  T = {:.4f} m^2/min'.format(T))
    print('  S = {:.4e}'.format(S))
    if beta > 0:
        print('  B = {:.4e}'.format(r/beta))

    # 计算误差
    u = r**2*S/4/T/t
    RSS = np.sum((Q*hantush_jacob(u, beta)/np.pi/T/4 - s)**2)
    print('Residual Sum of Squares: {:.4e}'.format(RSS))
```

交互绘图：

widgets.FloatSlider 用于控制位移量。为了使观测数据尽可能靠近标准曲线，可以选取 2 组观测数据按 Jacob 公式计算参数。这样处理可使观测数据充分靠近标准曲线，位移界限可以设置为半个坐标格。B 初始可设一个很大的数，如 1e06。

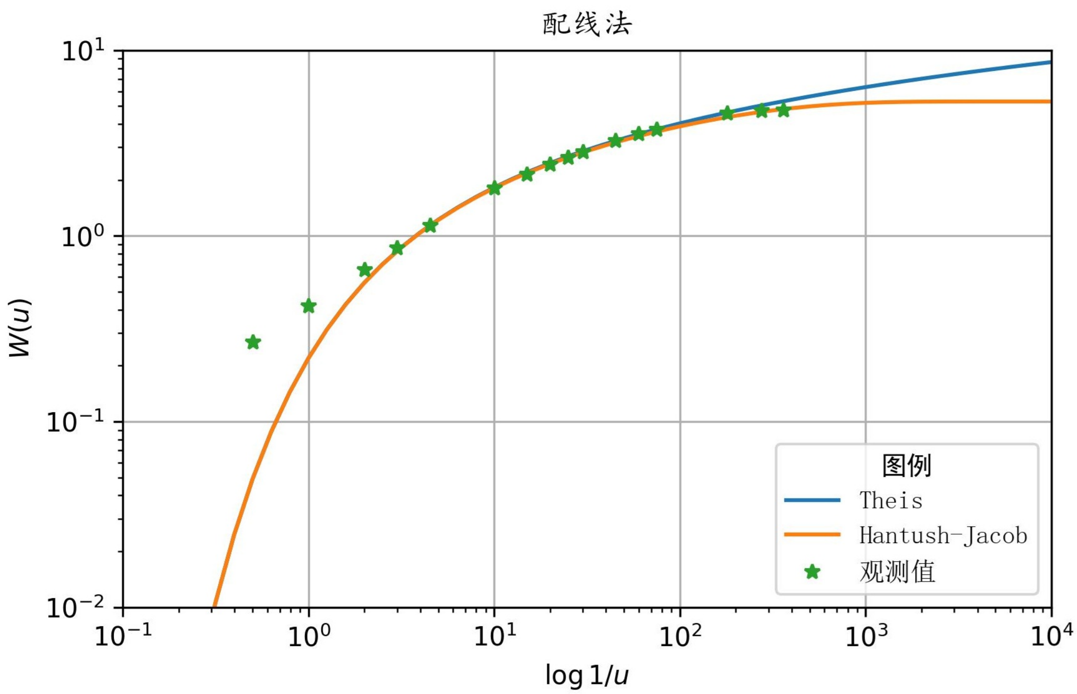
[16]:
```
# 为使观测点尽可能靠近标准曲线，最简单方法是取中间相邻两点用 Jacob 公式计算初始参数
# 然后再计算位移
i1 = int(len(t)/2)
i2 = i1 + 1
slope = (s[i1] - s[i2])/np.log10(t[i1]/t[i2])

T = 0.183*Q/slope
S = 2.25*T*t[i1]/r**2/np.float_power(10, s[i1]/slope)

B = 1e06   # B 很大
```

[17]:
```
# 计算初始平移量
delta_x = np.log10(4*T/S/r**2)
delta_y = np.log10(4*np.pi*T/Q)

# 设置位移界限为半个坐标格
delta_xmin = delta_x - 0.5
delta_xmax = delta_x + 0.5
delta_ymin = delta_y - 0.5
delta_ymax = delta_y + 0.5

# 设置 widgets.FloatSlider 参数，包括最小值、最大值、步长、标签、显示格式等
delta_y = widgets.FloatSlider(
        value=delta_y, min=delta_ymin, max=delta_ymax, step=0.05,
        description=r"$\Delta \lg s$ :", continuous_update=False,
        readout_format='.2f', disabled=False)
delta_x = widgets.FloatSlider(
        value=delta_x, min=delta_xmin, max=delta_xmax, step=0.05,
        description=r"$\Delta \lg t$ :", continuous_update=False,
        readout_format='.2f', disabled=False)

beta = widgets.FloatSlider(
    value=r/B, min=0.0, max=+3.0, step=0.01,
    description=r'$\beta$ :', continuous_update=False,
    readout_format='.2f', disabled=False)

plt.style.use('default')   # 设置图形风格，如可以设为 'seaborn-deep'
fig, ax = plt.subplots(dpi=100)  # 想获得高品质图形，可以设 dpi=300

# ipywidgets 小部件控制参数实现互动。ipywidgets 有缓冲功能，
# 同一个 Notebook 复制的代码得不到所期望的结果
widgets.interact(Hantush_Jacob_fit, delta_x=delta_x, delta_y=delta_y, beta=beta);
```

$\Delta \lg t$: -0.30

$\Delta \lg s$: 1.03

β : 0.08




```
  T = 0.3125 m^2/min
  S = 3.0698e-04
  B = 1.1250e+03
Residual Sum of Squares: 1.4250e-03
```

[18]:
```
fig.savefig("2.3.jpg")
# plt.close("all")
# widgets.FloatSlider.close_all()
```

练习 3： 某河阶地，上部为潜水层，其下为 2 m 厚弱透水的亚砂土，再下为 1.5 m 厚的中、粗砂层 (承压)。水源地抽取承压水，以 T32 号孔做非稳定抽水试验，距它 197 m 处有 T31 孔观恻，抽水量为 $Q = 69.1\ m^3/h$，观测孔水位降深值如下表，试用配线法求水文地质参数。

表某越流含水层抽水试验 T31 观测孔降深资料

抽水累计时间 $t(min)$	水位降深 $s(m)$	抽水累计时间 $t(min)$	水位降深 $s(m)$	抽水累计时间 $t(min)$	水位降深 $s(m)$
1	0.05	60	0.575	300	0.763
4	0.054	75	0.62	330	0.77
7	0.12	90	0.64	360	0.77
10	0.175	120	0.685	390	0.785
15	0.26	150	0.725	420	0.79
20	0.33	180	0.735	450	0.792
25	0.383	210	0.755	480	0.794
30	0.425	240	0.76	510	0.795
45	0.52	270	0.76	540	0.796

抽水累计时间 $t(min)$	水位降深 $s(m)$	抽水累计时间 $t(min)$	水位降深 $s(m)$	抽水累计时间 $t(min)$	水位降深 $s(m)$

4 实验 3：交互式 Hantush - Jacob 公式拐点法求水文地质参数

Hantush-Jacob 公式

$$s = \frac{Q}{4\pi T}\int_u^{\infty}\frac{1}{y}e^{-y-\frac{\beta^2}{4y}}\,dy = \frac{Q}{4\pi T}W(u,\beta)$$

式中，$u = \frac{r^2 S}{4Tt}$，$\beta = \frac{r}{B}$。

在单对数坐标刻度下，$s - \lg t$ 曲线有拐点，且在拐点处有如下性质：

拐点 P 的性质

- 坐标与降深：
$$t_p = \frac{SBr}{2T},\quad s_p = \frac{1}{2}s_{max} = \frac{Q}{4\pi T}K_0\left(\frac{r}{B}\right)$$
- 切线斜率：
$$i_p = \frac{2.3Q}{4\pi T}e^{-\frac{r}{B}}$$
- s_p、i_p 的关系：
$$2.3\frac{s_p}{i_p} = e^{\frac{r}{B}}K_0\left(\frac{r}{B}\right)$$

应用拐点性质可求取参数。

设计思路

1. 调整参数确定拐点坐标与降深 (t_p, s_p)，最大降深 s_{max} 用外推法确定;
2. 调整参数确定拐点切线斜率 i_p；
3. 二分法求解方程: $e^{\frac{r}{B}}K_0\left(\frac{r}{B}\right) - 2.3\frac{s_p}{i_p} = 0$:
$$f'(x) = e^x(K_0(x) - K_1(x)) < 0$$
$f(x)$ 在 $[0.01, 5]$ 上是单调递减的，二分法求解有效。
4. 计算参数：
$$B = \frac{r}{\left[\frac{r}{B}\right]},\quad T = \frac{2.3Q}{4\pi i_p}e^{-\frac{r}{B}},\quad S = \frac{2Tt_p}{Br}$$
5. 验证：避免确定 s_{max} 的随意性, 用 Hantush-Jacob 公式进行验证。

与交互式配线法相同的思路，我们用 `widgets.FloatSlider` 控制拐点位置与切线斜率，拐点降深通过输入设定。程序中用到 Hantush - Jacob 井函数的计算可用已有的程序。

例 5：根据 **例 4** 的数据用拐点法求水文地质参数。

Python 程序

导入库：

[19]:
```python
%matplotlib ipympl

# 导入库
import math as math
import matplotlib.pyplot as plt
import numpy as np
import scipy.special as sps       # 计算 hantush - jacob 井函数用到 Bessel
import ipywidgets as widgets      # 导入交互绘图模块
from scipy.optimize import bisect # 二分法

# 控制小数的显示精度
np.set_printoptions(precision=4)
# 使不用 Unicode 负号，防止出现乱码
plt.rcParams['axes.unicode_minus']=False
```

数据准备：

[20]:
```python
# 准备数据
Q = 69.1/60     # m^3/min
r = 197.0       # m

t = np.array([1, 4, 7, 10, 15, 20, 25, 30, 45, 60,
              75, 90, 120, 150, 180, 210, 240, 270, 300, 330,
              360, 390, 420, 450, 480, 510, 540])        # min
s = np.array([0.05, 0.054, 0.12, 0.175, 0.26, 0.33, 0.383, 0.425, 0.52,
              0.575, 0.62, 0.64, 0.685, 0.725, 0.735, 0.755, 0.76, 0.76,
              0.763, 0.77, 0.772, 0.785, 0.79, 0.792, 0.794, 0.795, 0.796])  # m
```

定义一个绘图函数：

注意到 matplotlib.pyplot 对象全局适用，因此例程中需要用代码清除上次绘制的内容，这样才不会出错。程序大部分代码都是设置图形的，如图形界限、坐标刻度、图例等。ax.plot(x, sp + slope*np.log10(x/tp), label=" 拐点切线") 绘制直线，ax.plot(t, s, '*', label=" 观测值") 绘观测值，ax.plot(tp, sp, marker="o") 绘拐点，ax.plot(x, 0.25*Q*hantush_jacob(u, beta)/np.pi/T, label=" 标准曲线") 绘对应 beta 的 Hantush-Jacob 标准曲线，x 变量是在一个对数坐标格中均匀取值。beta = bisect(lambda x:np.exp(x)*sps.k0(x) - 2.3*sp/slope, 0.01, 5) 用二分法求 beta。

[21]:
```python
# 定义一个绘图函数
# 参数 tp, sp, slope 分别为拐点位置，拐点处降深，拐点处切线斜率
def inflection_point_fit(tp, sp, slope):

    global t, s, Q, r, B   # 全局变量，改变后函数外的值同时改变
```

```
# Clear the previous plot
for i in plt.get_fignums():
    plt.gca()
    ax.cla()

# 计算绘图范围
imin = math.floor(math.log10(min(t)))  # math.floor(x) 返回小于 x 的最大整数
imax = math.ceil(math.log10(max(t)))   # math.ceil(x) 返回大于等于 x 的最小整数
xmin = 10**imin
xmax =  10**imax
ymin = 0.0
ymax = math.ceil(max(s*10))/10
ax.set_xlim(xmin, xmax)
ax.set_ylim(ymin, ymax)

# 设置坐标轴刻度为常用对数
ax.set_xscale("log")

# 坐标轴标签
# r 字符串前缀转义字符将被忽略, '$\log 1/u$' 显示为 latex 数学符号
plt.xlabel(r'$\log t$')
plt.ylabel(r'$s$')

# 显示网格
ax.grid(True)

# 生成均匀的坐标点
ix = np.linspace(imin, imax, (imax - imin)*10 + 1)
x = np.float_power(10, ix)
# 绘制拐点切线, 点斜式
ax.plot(x, sp + slope*np.log10(x/tp), label="拐点切线")
# 绘制观测值的散点图形
ax.plot(t, s, '*', label="观测值")
# 绘制拐点
ax.plot(tp, sp, marker="o")

# 调用 scipy.optimize.bisect(二分法) 求 beta
beta = bisect(lambda x:np.exp(x)*sps.k0(x) - 2.3*sp/slope, 0.01, 5)

# 计算参数
```

```
    T = 2.3*Q*np.exp(-beta)/4/np.pi/slope
    S = 2*T*tp*beta/r**2
    B = r/beta

    # 可以绘出 hantush - jacob 单对数标准曲线，便于观察结果
    u = 0.25*r**2*S/T/x
    ax.plot(x, 0.25*Q*hantush_jacob(u, beta)/np.pi/T, label="标准曲线")

    # 指定图例标题与内容的字体
    plt.legend(prop={'family':'Simsun'}, loc=4, title="图例",
               title_fontproperties={'family':'SimHei'})
    # 指定绘图标题为楷体
    ax.set_title("拐点法", fontproperties={'family':'KaiTi', 'size':'large'})

    # 显示图形
    plt.show()

    # 显示参数计算结果
    print(' T(m^2/min): {:.4f}'.format(T))
    print('          S: {:.4e}'.format(S))
    print('       B(m): {:.4e}'.format(B))
```

交互绘图:

widgets.FloatSlider 用于控制拐点坐标与直线斜率。widgets.FloatText 用于输入最大降深值。

为了使直线尽可能靠近拐点，拐点初始值对应数组 s 索引用 `np.where(s > sp)[0][0]` 确定，取 tp 相邻点用 Jacob 公式计算参数与直线斜率。*B* 初始可设一个很大的数，如 1e06。

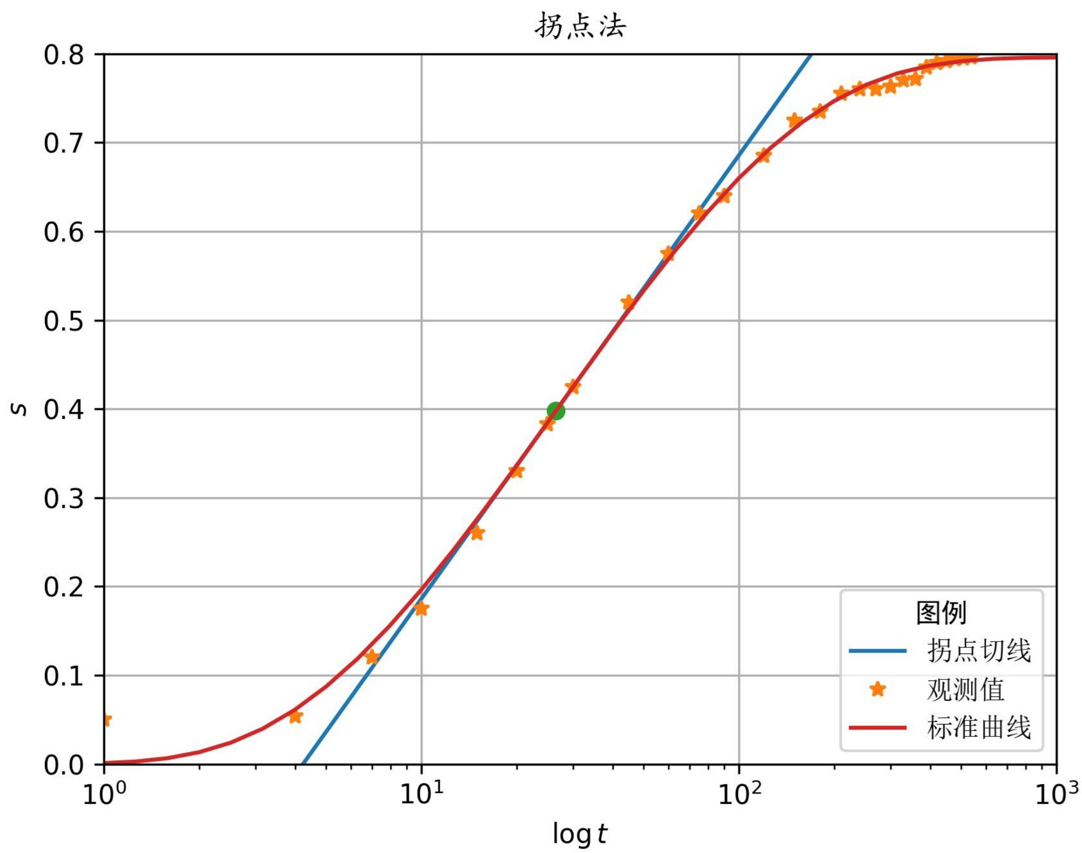
[22]:
```
# 设定初始的 sp, tp
sp = 0.5*s[-1]
idx = np.where(s > sp)[0][0]
tp = t[idx]

# 设置初始的 T, S, beta, slope
# 最简单方法是取 tp 相邻点用 Jacob 公式计算
i1 = idx
i2 = i1 + 1
slope = (s[i1] - s[i2])/np.log10(t[i1]/t[i2])

T = 0.183*Q/slope
S = 2.25*T*t[i1]*np.float_power(10, -s[i1]/slope)/r**2
```

```python
B = 1e06   # B 很大

# 设置 widgets.FloatSlider 参数，包括最小值、最大值、步长、标签、显示格式等

tp_min = tp*np.float_power(10, -0.5)    # 左右平移量为半个对数做表格
tp_max = tp*np.float_power(10, +0.5)
tp_step = (tp_max - tp_min)/50

tp = widgets.FloatSlider(
    value=tp, min=tp_min, max=tp_max, step=tp_step,
        description=r'$t_p$ :', continuous_update=False,
        readout_format='.1f', disabled=False)

sp = widgets.FloatText(
    value=sp, description=r'$s_p$ :',
    continuous_update=False, disabled=False)

slope = widgets.FloatSlider(
    value=slope, min=slope-0.5, max=slope+0.5, step=0.01,
    description='slope :', continuous_update=False,
    readout_format='.2f', disabled=False)


plt.style.use('default')   # 设置图形风格，如可以设为 'seaborn-deep'
fig, ax = plt.subplots(dpi=100)  # 想获得高品质图形，可以设 dpi=300

# ipywidgets 小部件控制参数实现互动。ipywidgets 有缓冲功能，
# 同一个 Notebook 复制的代码得不到所期望的结果
widgets.interact(inflection_point_fit, tp=tp,sp=sp, slope=slope);
```

t_p : 26.6

s_p : 0.398

slope : 0.50




```
T(m^2/min): 0.3098
          S: 1.3104e-04
       B(m): 6.3765e+02
```

[23]:
```
fig.savefig("3.1.jpg")
# plt.close("all")
# widgets.FloatSlider.close_all()
```

练习 4：根据 **例 4** 的数据用拐点法求参。

5 实验 4：交互式 Jacob 公式直线图解法求水文地质参数

为了程序通用性，以 $s \sim \lg \frac{t}{r^2}$ 为例：

$$s \sim \frac{0.183Q}{T} \lg \frac{t}{r^2} + \frac{0.183Q}{T} \lg \frac{2.25T}{S}$$

写成点斜式

$$s \sim i \lg x - i \lg x_0$$

式中，$i=\frac{0.183Q}{T}$，$x=\frac{t}{r^2}$，$x_0=\frac{S}{2.25T}$。

其中，x_0 就是 0 降深截距，通过调整 x_0 与 i 拟合直线，并用如下公式计算参数

$$T=\frac{0.183Q}{i},\quad S=2.25Tx_0$$

例 6：根据 **例 4** 的数据用直线图解法求水文地质参数。

Python 程序

导入库：

[24]:
```
%matplotlib ipympl

# 导入库
import numpy as np
import math as math
import matplotlib.pyplot as plt
import ipywidgets as widgets

# 控制小数的显示精度
np.set_printoptions(precision=4)
# 使不用 Unicode 负号，防止出现乱码
plt.rcParams['axes.unicode_minus']=False
```

准备数据：

[25]:
```
# 准备数据
Q = 528/1440   # m^3/min
r = 90         # m

t = np.array([1, 2, 4, 6, 9, 20, 30, 40, 50, 60,
              90, 120, 150, 360, 550, 720])      # min
x = t/r**2
s = np.array([2.5, 3.9, 6.1, 8.0, 10.6, 16.8, 20.0, 22.6, 24.7, 26.4,
              30.4, 33.0, 35.0, 42.6, 44.0, 44.5])/100    # m
```

定义一个绘图函数：

注意到 matplotlib.pyplot 对象全局适用，因此例程中需要用代码清除上次绘制的内容，这样才不会出错。程序大部分代码都是设置图形的，如图形界限、坐标刻度、图例等。

ax.plot(x, s, '*', label="观测值") 绘观测值，其中 x 为取值 t/r^2 数组，ax.plot(x_, slope*np.log10(x_/x0), label="拟合直线") 绘直线，x_ 为生成的均匀分布的坐标点。

[26]:
```python
# 定义一个绘图函数
# 参数 x0, slope 分别为截距与斜率
def line_fit(x0, slope):
    global Q, r, x, s    # 全局变量

    # Clear the previous plot
    for i in plt.get_fignums():
        plt.gca()
        ax.cla()

    # 计算绘图范围
    ymin = 0.0
    ymax = math.ceil(max(s*10))/10
    imin = math.floor(math.log10(min(x)))
    imax = math.ceil(math.log10(max(x)))
    xmin = 10**imin
    xmax = 10**imax
    ax.set_xlim(xmin, xmax)
    ax.set_ylim(ymin, ymax)

    # 设置坐标轴刻度为常用对数
    ax.set_xscale("log")
    # 坐标轴标签
    # r 字符串前缀转义字符将被忽略，'$\log 1/u$' 显示为 latex 数学符号
    plt.xlabel(r'$\log t/r^2$')
    plt.ylabel(r'$s$')

    # 显示网格
    ax.grid(True)

    # 绘制观测值的散点图形
    ax.plot(x, s, '*', label="观测值")

    # 生成均匀的坐标点，并绘直线，点斜式
    x_ = np.linspace(xmin, xmax, 50)
    ax.plot(x_, slope*np.log10(x_/x0), label="拟合直线")

    # # 计算参数
    T = 0.183*Q/slope
    S = 2.25*T*x0
```

```
    # 指定图例标题与内容的字体
    plt.legend(prop={'family':'Simsun'}, loc=4, title="图例",
               title_fontproperties={'family':'SimHei'})
    # 指定绘图标题为楷体
    ax.set_title("直线图解法", fontproperties={'family':'KaiTi', 'size':'large'})

    # 显示图形
    plt.show()

    # 显示参数计算结果
    print('  T = {:.4f} m^2/min'.format(T))
    print('  S = {:.4e}'.format(S))
```

交互绘图:

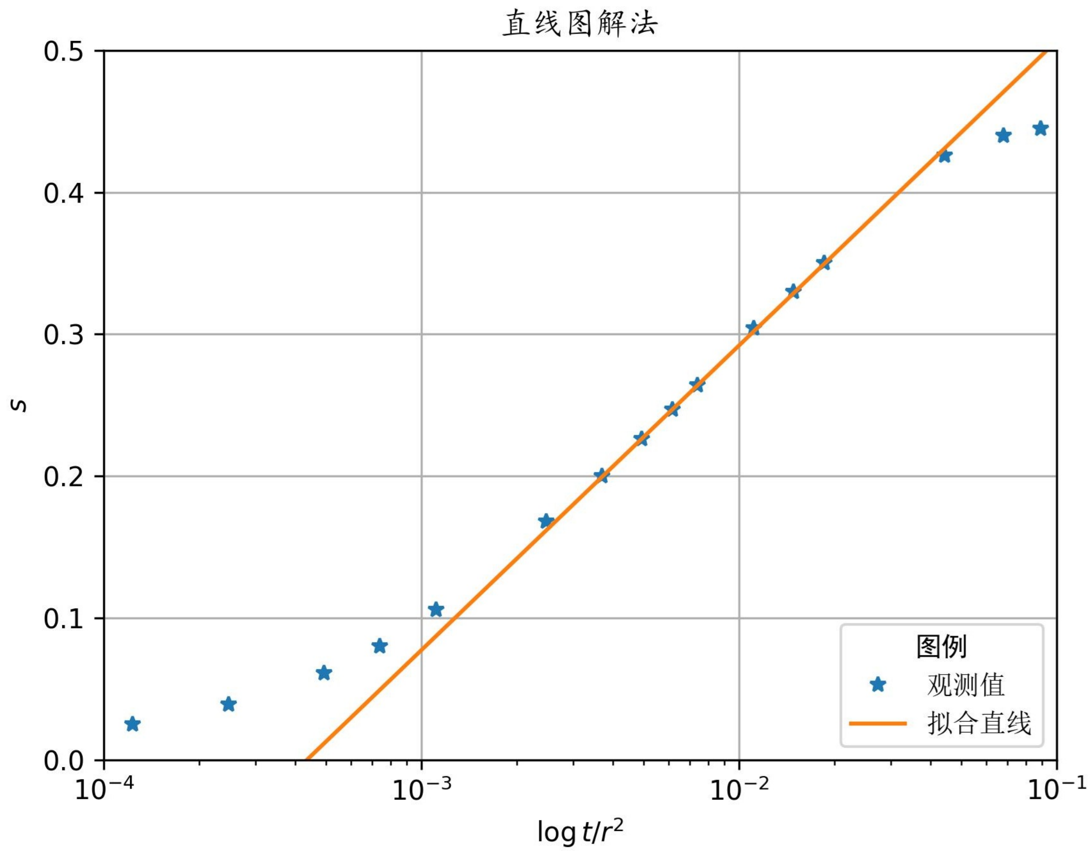
widgets.FloatSlider 用于控制直线截距与直线斜率。为了使直线尽可能靠近观测数据，最简单方法是取中间相邻两点用 Jacob 公式计算初始斜率与截距。

[27]:
```
# 初始参数，最简单方法是取中间相邻两点用 Jacob 公式计算
i1 = int(len(x)/2)
i2 = i1 + 1
slope = (s[i1] - s[i2])/np.log10(x[i1]/x[i2])

T = 0.183*Q/slope
S = 2.25*T*x[i1]*np.float_power(10, -s[i1]/slope)

# 设置 widgets.FloatSlider 参数，包括最小值、最大值、步长、标签、显示格式等

x0 = S/2.25/T
x0_min = x0*np.float_power(10, -0.5)  # 左右平移量为半个对数坐标格
x0_max = x0*np.float_power(10, +0.5)
x0_step = (x0_max - x0_min)/50
x0 = widgets.FloatSlider(
    value=x0, min=x0_min, max=x0_max, step=x0_step,
    description=r'$\frac{t}{r^2}$ :', continuous_update=False,
    readout_format='.5f', disabled=False)

slope = widgets.FloatSlider(
    value=slope, min=slope-0.5, max=slope+0.5, step=0.01,
    description='slope :', continuous_update=False,
    readout_format='.3f', disabled=False)
```

```
plt.style.use('default')          # 设置图形风格，如可以设为 'seaborn-deep'
fig, ax = plt.subplots(dpi=100)   # 想获得高品质图形，可以设 dpi=300

# ipywidgets 小部件控制参数实现互动。ipywidgets 有缓冲功能，
# 同一个 Notebook 复制的代码得不到所期望的结果
widgets.interact(line_fit, x0=x0, slope=slope);
```

$\frac{t}{r^2}$: 0.00044

slope : 0.215




```
T = 0.3125 m^2/min
S = 3.0698e-04
```

[28]:
```
fig.savefig("4.1.jpg")
# plt.close("all")
# widgets.FloatSlider.close_all()
```

练习 5：根据 **例 2** 的数据用直线图解法求参。

思考题：

(1) 对比 Theis 配线、Jacob 直线求参结果的差异并分析原因。

(2) 选择拟合直线数据应该考虑哪些问题？

(3) 试用单个观测孔、多个观测孔求参，并比较结果。(提示：修改准备数据的代码，用函数 `np.concatenate()` 拼接数组。)

6 实验 5：最小二乘法应用

6.1 最小二乘法原理

根据观测数据 $\{(x_i, y_i), i = 1, 2, \cdots, m\}$，求拟合直线 $y = \beta_1 + \beta_2 x$，使 $E(\beta_1, \beta_2) = \sum_{i=1}^{m}(y_i - \beta_1 - \beta_2 x_i)^2$ 取最小值。该问题也可认为是求解超定方程组，即确定系数 β，使 $X\ \beta = Y$ 在最小二乘意义上成立。

式中，$X = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_m \end{bmatrix}^T$，$Y = [y_1, y_2, \cdots, y_m]^T$，$\beta = [\beta_1, \beta_2]^T$

一般形式

考虑超定方程组（超定指未知数小于方程个数）

$$\sum_{j=1}^{n} x_{ij}\beta_j = y_i, \quad i = 1, 2, 3, \cdots, m$$

其中，m 代表样本数，n 代表参数维度，写成矩阵形式

$$X\ \beta = Y \tag{6.1}$$

式中，

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

为得到 β 的最佳估计 $\hat{\beta}$，将问题转化为如下的最值问题：

$$\min E(\beta) = \min \left\| X\beta - Y \right\|^2$$

通过微分求最值，得

$$(X^T X)\beta = (X^T Y) \tag{6.2}$$

若 $X^T X$ 为非奇异矩阵，则 β 有唯一解，即

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

可以看出，求解最小二乘问题的关键是构造方程组。

numpy 库中的 numpy.linalg.solve 可用于求解形如 (6.2) 的方程组，numpy.linalg.lstsq 就是最小二乘法，可用于求解形如 (6.1) 的超定方程组。scipy 库的 scipy.linalg.lstsq 也是最小二乘法。

例 7：一次模拟实验中，输入 x（自变量）为 [0.00, 0.30, 0.60, 0.90, 1.08, 1.20, 1.30, 1.48, 1.60, 1.70, 1.78, 1.85, 1.90, 1.95, 2.00]，观测到的输出 y（因变量）为 [1.78, 1.91, 2.01, 2.12, 2.20, 2.22, 2.25, 2.32, 2.38, 2.41, 2.43, 2.47, 2.49, 2.48, 2.51]。根据实验分析，y 与 x 成线性关系，试确定关系表达式。

Python 程序

方法一：构造形如 (6.2) 的方程组，用 numpy.linalg.solve 求解，返回参数数组。np.ones() 生成元素都为 1 的向量，np.vstack() 按垂直方向（行顺序）堆叠数组构成一个新的数组，numpy.array.T 将矩阵转置。

导入库：

[29]:
```python
%matplotlib ipympl

# 导入库
import numpy as np
import matplotlib.pyplot as plt

# 控制小数的显示精度
np.set_printoptions(precision=4)
# 使不用 Unicode 负号，防止出现乱码
plt.rcParams['axes.unicode_minus']=False
```

数据准备：

[30]:
```python
# 将原始数据转化为 numpy 的 array 数组
x = np.array([0.00, 0.30, 0.60, 0.90, 1.08, 1.20, 1.30, 1.48, 1.60, 1.70,
              1.78, 1.85, 1.90, 1.95, 2.00])
y = np.array([1.78, 1.91, 2.01, 2.12, 2.20, 2.22, 2.25, 2.32, 2.38, 2.41,
              2.43, 2.47, 2.49, 2.48, 2.51])
```

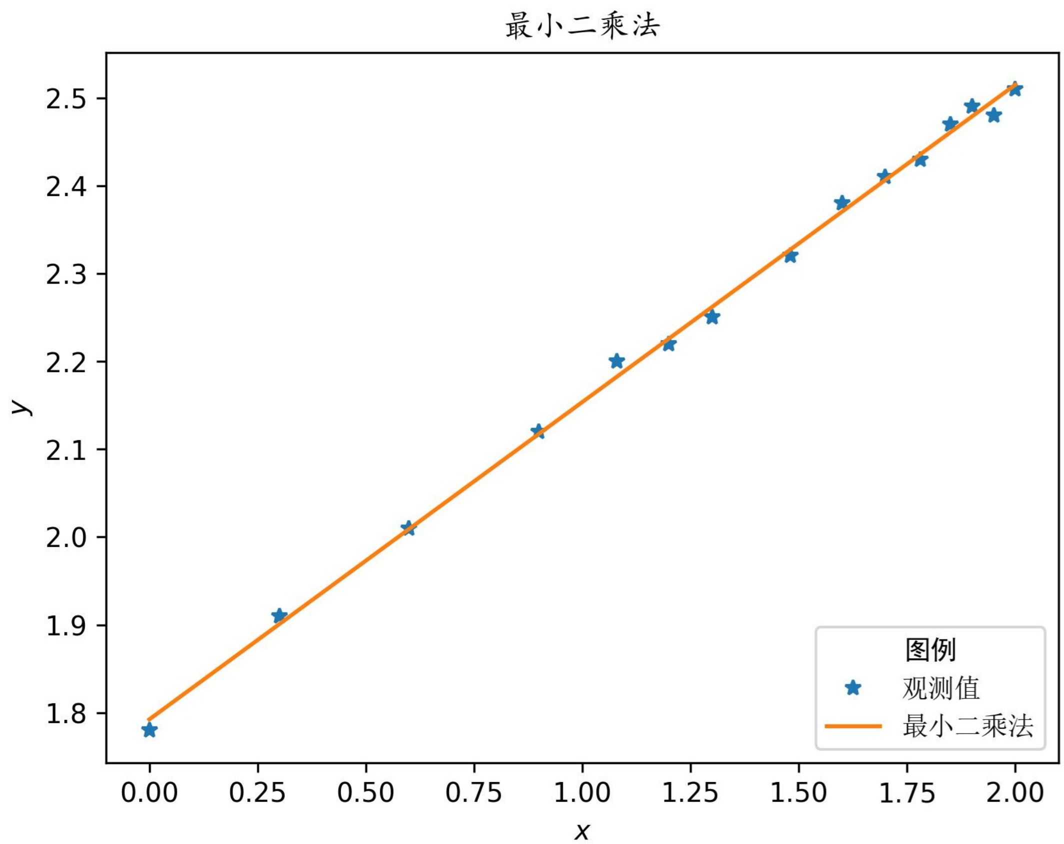
形成线性方程组并求解：

[31]:
```python
# 形成方程组
X = np.vstack([np.ones(len(x)), x]).T
A = np.dot(X.T, X)
b = np.dot(X.T, y)
```

```
# 求解方程组，beta 返回解
beta = np.linalg.solve(A, b)
```

输出计算结果：

[32]:
```
# 绘图
fig, ax = plt.subplots(dpi=100)

# 绘制散点图
ax.plot(x, y, "*", label="观测值")
# 绘直线图
ax.plot(x, beta[0] + beta[1]*x, "-", label="最小二乘法")

# 设置数轴标签
plt.xlabel(r"$x$")
plt.ylabel(r"$y$")

# 指定图例标题与内容的字体
plt.legend(prop={'family':'Simsun'}, loc=4, title="图例",
           title_fontproperties={'family':'SimHei'})
# 指定绘图标题为楷体
ax.set_title("最小二乘法", fontproperties={'family':'KaiTi', 'size':'large'})

# 显示图形
plt.show()

# 显示计算结果与误差
print("beta =", beta, "residuals =",
      "{:.4f}".format(sum((beta[0] + beta[1]*x - y)**2)))
```



beta = [1.7924 0.3612] residuals = 0.0014

[33]:
```
fig.savefig("5.1.jpg")
# plt.close("all")
# widgets.FloatSlider.close_all()
```

方法 2：构造形如 (6.1) 的方程组，用 np.linalg.lstsq 求解。

导入库：

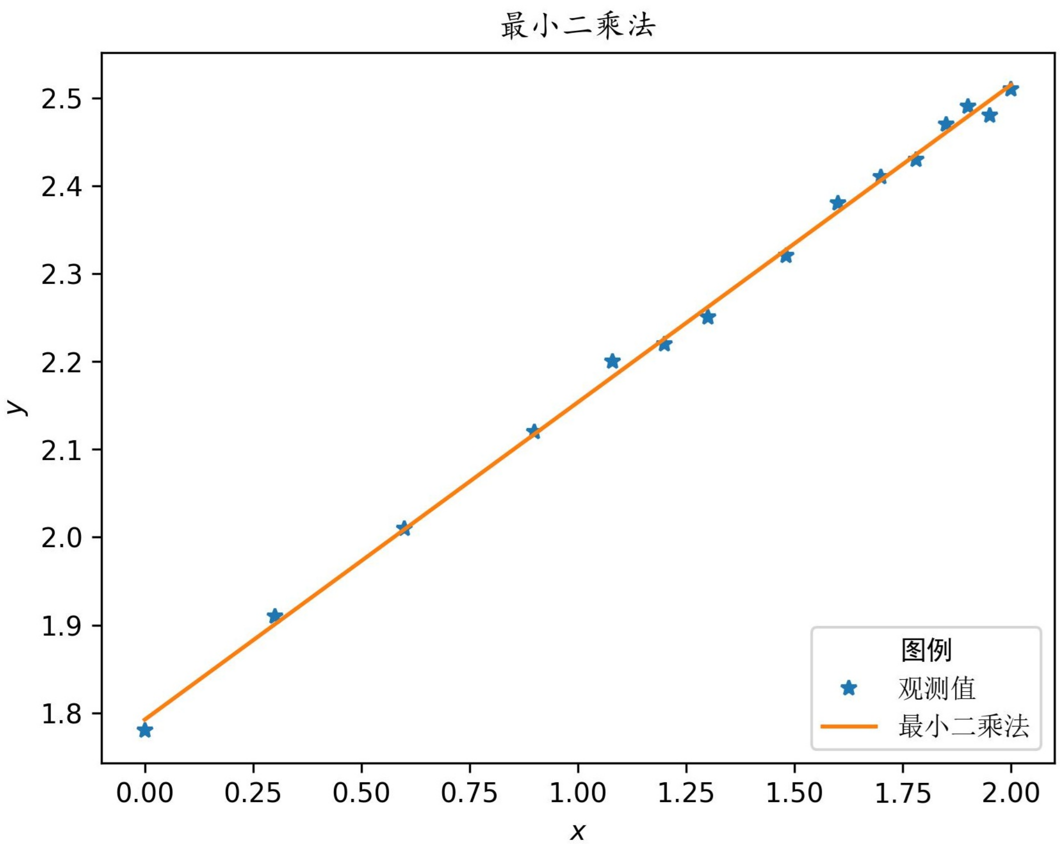
[34]:
```
%matplotlib ipympl

# 导入库
import numpy as np
import matplotlib.pyplot as plt

# 控制小数的显示精度
np.set_printoptions(precision=4)
# 使不用 Unicode 负号，防止出现乱码
plt.rcParams['axes.unicode_minus']=False
```

形成超定方程组并求解：

[35]:
```
# 将原始数据转化为 numpy 的 array 数组
x = np.array([0.00, 0.30, 0.60, 0.90, 1.08, 1.20, 1.30, 1.48, 1.60, 1.70,
              1.78, 1.85, 1.90, 1.95, 2.00])
y = np.array([1.78, 1.91, 2.01, 2.12, 2.20, 2.22, 2.25, 2.32, 2.38, 2.41,
              2.43, 2.47, 2.49, 2.48, 2.51])

# 形成系数
X = np.vstack([np.ones(len(x)), x]).T

# 调用 np.linalg.lstsq, # beta[0] 为最小二乘解,
# beta[1][0] 为误差平方和
beta = np.linalg.lstsq(X, y, rcond=None)
```

输出计算结果：

[36]:
```
# 绘制图形
fig, ax = plt.subplots(dpi=100)

# 绘制散点图
ax.plot(x, y, "*", label="观测值")
# 绘直线图
ax.plot(x, beta[0][0] + beta[0][1]*x, "-", label="最小二乘法")

# 设置数轴标签
plt.xlabel(r"$x$")
plt.ylabel(r"$y$")

# 指定图例标题与内容的字体
plt.legend(prop={'family':'Simsun'}, loc=4, title="图例",
           title_fontproperties={'family':'SimHei'})
# 指定绘图标题为楷体
ax.set_title("最小二乘法", fontproperties={'family':'KaiTi', 'size':'large'})

# 显示图形
plt.show()

# 显示计算结果与误差
print("beta =", beta[0], "residuals =", "{:.4f}".format(beta[1][0]))
```




```
beta = [1.7924 0.3612] residuals = 0.0014
```

[37]:
```
fig.savefig("5.2.jpg")
# plt.close("all")
# widgets.FloatSlider.close_all()
```

6.2 最小二乘法确定 Q - s 曲线类型

常见的 $Q \sim s_w$ 曲线类型

- **直线型：** $Q = q\ s_w$
- **抛物线型：** $s_w = a\ Q + b\ Q^2$
- **幂函数型：** $Q = q_0\ s_w^{\frac{1}{m}}$
- **对数型：** $Q = a + b\ \lg s_w$

将常用的 $Q \sim s$ 类型变换形式：

- 直线型：$Q = b\ s_w$ (不变，要求通过原点）$(b > 0)$
- 抛物线型：$\frac{s_w}{Q} = a + b\ Q$，$(a > 0,\ b > 0)$
- 幂函数型：$\lg Q = a + b\ \lg s_w$，（$b > 0$）
- 对数型：$Q = a + b\ \lg s_w$ (不变)，（$a > 0,\ b > 0$ ）

根据最小二乘法原理构造超定方程组 $X\,\beta = Y$，进一步形成方程组 $A\,\beta = b$，用 `numpy.linalg.solve` 求解（方法 1）。

式中 $A = X^T\,X, b = X^T\,Y$。

分别计算降深误差平方和 $\sum(s-\hat{s})^2$，最小者可能就是最优型。

由于以 day 或 hour 为单位的流量较大，建议流量以 min 记。

例 8：在厚度为 16.50 m 的承压含水层中做了 3 次降深的抽水试验，其结果记录在下表中。试求 $Q \sim s$ 曲线。

表稳定流抽水试验数据

$r_w(m)$	$s_1(m)$	$Q_1(m^3/d)$	$s_2(m)$	$Q_2(m^3/d)$	$s_3(m)$	$Q_3(m^3/d)$
0.40	1.16	320.54	1.60	421.63	1.90	536.54

Python 程序

导入库：

[38]:
```
%matplotlib ipympl

# 导入库
import numpy as np
import matplotlib.pyplot as plt

# 控制小数的显示精度
np.set_printoptions(precision=4)
# 使不用 Unicode 负号，防止出现乱码
plt.rcParams['axes.unicode_minus']=False
```

定义一个绘图程序：

[39]:
```
# 定义一个绘图函数，其它程序也可使用
def plot_Qs(s, Q, beta1, beta2, beta3, beta4):

    fig, ax = plt.subplots(figsize=(6, 4), dpi=100)

    ax.plot(s, Q, '*', label="ob")  # 画出观测数据

    ymin = 0.0
    ymax = math.ceil(max(Q*10))/10
    ax.set_ylim(ymin, ymax)
    ax.set_xlim(xmin = 0)

```

```
x = np.linspace(1e-4, np.max(s), 100)

# 绘直线型
ax.plot(x, beta1*x, label="直线型")

# 绘抛物线型，需要反算流量
ax.plot(x, (-beta2[0] + np.sqrt(beta2[0]**2 + 4*beta2[1]*x))/2/beta2[1],
        label="抛物线型")

# 绘幂函数型
ax.plot(x, np.power(10, beta3[0])*np.power(x, beta3[1]), label="幂函数型")

# 绘对数型
ax.plot(x, beta4[0] + np.log10(x)*beta4[1], label="对数型")

# 数轴标签
plt.xlabel(r'$s_w$')
plt.ylabel(r'$Q$')

# 指定图例标题与内容的字体
plt.legend(prop={'family':'Simsun'}, loc=4, title="图例",
           title_fontproperties={'family':'SimHei'})
# 指定绘图标题为楷体
ax.set_title(r"$Q\sim s$ 曲线", fontproperties={'family':'KaiTi', 'size':'large'})

# 计算降深 s 的误差平方和
print('s 误差平方和：')

# 直线型
rss = sum((s - Q/beta1)**2)
print('  直线型:','beta = {:.4f},'.format(beta1),
      ' RSS of s = {:.4e}'.format(rss))
# 抛物型
rss = sum((s - beta2[0]*Q - beta2[1]*Q**2)**2)
print('  抛物型:','beta = {},'.format(beta2),
      ' RSS of s = {:.4e}'.format(rss))
# 幂函数型
rss = sum((s - np.float_power(10, (np.log10(Q) - beta3[0])/beta3[1]))**2)
print('幂函数型:','beta = {},'.format(beta3),
      ' RSS of s = {:.4e}'.format(rss))
# 对数型
```

```
    rss = sum((s - np.float_power(10, (Q - beta4[0])/beta4[1]))**2)
    print('  对数型:','beta = {},'.format(beta4),
          ' RSS of s = {:.4e}'.format(rss))

    return fig
```

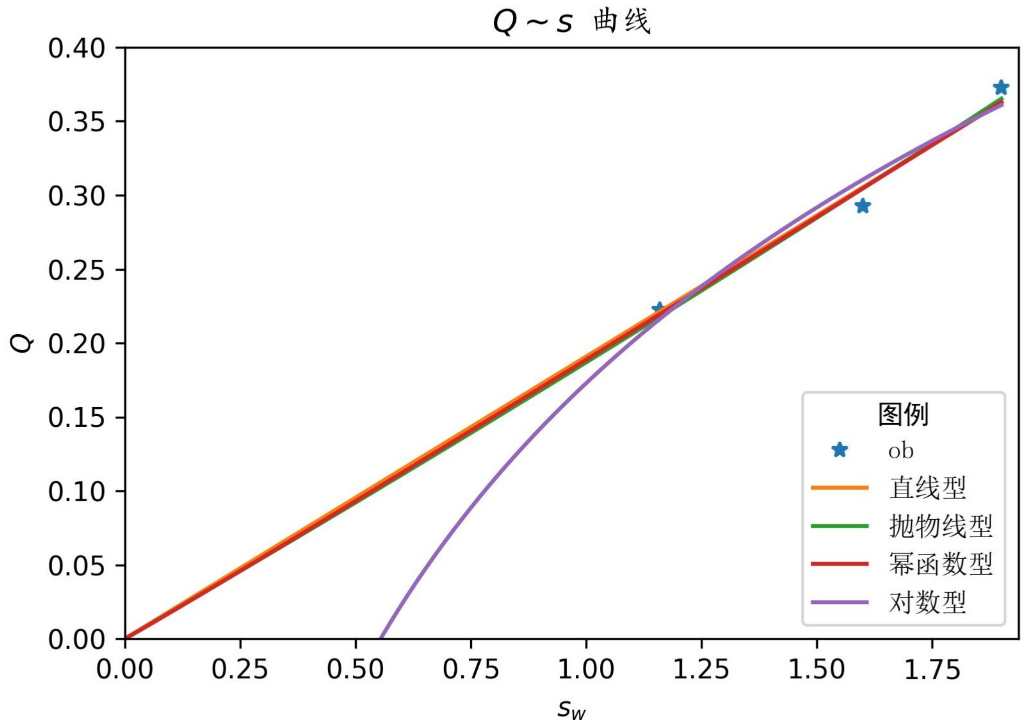
准备数据，形成线性方程组并求解：

[40]:
```
# (Q_i,s_i) 数据
Q = np.array([320.54, 421.63, 536.54])/1440   # m^3/min
s = np.array([1.16, 1.60, 1.90])

# 直线型, 直接求解
beta1 = s.dot(Q.T)/s.dot(s.T)
# 抛物线型
X = np.vstack([np.ones(len(Q)), Q]).T  # 形成系数
beta2 = np.linalg.solve(np.dot(X.T, X), np.dot(X.T, s/Q))
# 幂函数型
X = np.vstack([np.ones(len(Q)), np.log10(s)]).T  # 形成系数
beta3 = np.linalg.solve(np.dot(X.T, X), np.dot(X.T, np.log10(Q)))
# 对数型
X = np.vstack([np.ones(len(Q)), np.log10(s)]).T  # 形成系数
beta4 = np.linalg.solve(np.dot(X.T, X), np.dot(X.T, Q))
```

输出结果：

[41]:
```
# 绘图
fig = plot_Qs(s, Q, beta1, beta2, beta3, beta4)
```

```
s 误差平方和:
  直线型: beta = 0.1909,  RSS of s = 7.1050e-03
  抛物型: beta = [ 5.5048 -0.8326],  RSS of s = 5.3899e-03
幂函数型: beta = [-0.7242  1.0195],  RSS of s = 6.4173e-03
  对数型: beta = [0.1729 0.6747],  RSS of s = 1.5437e-02
```



[42]:
```
fig.savefig("5.3.jpg")
# plt.close("all")
# widgets.FloatSlider.close_all()
```

练习 6：某煤矿对太原组含水层四、五灰进行了简易稳定流抽水，抽水量 Q、抽水井降深 s_w、抽水井半径 r_w、含水层厚度 M 如表所示。试确定抽水孔的 Q~s 曲线类型并计算含水水文地质参数（参考实验 1）。

简易稳定流抽水数据表

孔号	$Q(m^3/d)$	$s_w(m)$	$r_w(m)$	$M(m)$
东 102	133.92	54.21	0.054	26.80
	125.28	44.51	0.054	26.80
	116.64	35.90	0.054	26.80
西 43	429.12	33.50	0.0445	34.90
	495.36	40.40	0.0445	34.90
	567.36	48.50	0.0445	34.90

思考题

(1) 通过比较，抛物线型的 s 误差平方和最小，问该井的 $Q \sim s$ 曲线确定为抛物型是否合适？为什么？

(2) 对比抛物线型的 $Q \sim s$ 与 C. E. Jacob 井损表达式可以看出，它们的形式是一样的。若按上述示例数据计算发现 C 是负值，试分析原因。

(3) 若已知含水层某钻孔的 $Q \sim s$ 类型，问该类型及数据能否用于预测同含水层其它钻孔的降深？

(4) 对于不同的 $Q \sim s$ 类型，如何根据计算结果分析其合理性？

6.3 Jacob 公式的最小二乘法线性拟合

在单对数坐标纸上用实测数据绘制，抽水一定时间后数据点呈直线状态：

$$s = \frac{0.183Q}{T}\lg\frac{2.25T}{r^2S} + \frac{0.183Q}{T}\lg t$$

写成如下形式

$$s = s_0 + i\lg t$$

拟合出直线斜率 i 与降深轴截距 s_0，用公式计算参数：

$$T = \frac{0.183Q}{i}, \quad S = \frac{2.25T}{r^2}10^{-\frac{s_0}{i}}$$

用最小二乘法可以方便计算出直线方程的参数。

例 9：根据 **例 4** 的数据用 Jacob 公式的最小二乘法线性拟合求水文参数。

Python 程序

导入库：

[43]:
```
%matplotlib ipympl

# 导入库
import numpy as np
import matplotlib.pyplot as plt

# 控制小数的显示精度
np.set_printoptions(precision=4)
# 使不用 Unicode 负号，防止出现乱码
plt.rcParams['axes.unicode_minus']=False
```

定义一个绘图程序：

[44]:
```
# 定义一个绘图程序
def plot_jacob(t, s, beta):

    fig, ax = plt.subplots(figsize=(6, 4), dpi=100)

    # y 轴最小-最大值
```

```
    ymin = 0.0
    ymax = math.ceil(max(s*10))/10

    # 对数轴最小-最大值
    imin = math.floor(math.log10(min(t)))
    imax = math.ceil(math.log10(max(t)))
    xmin = 10**imin
    xmax = 10**imax

    plt.style.use('default')  # 默认格式
    ax.set_xlim(xmin, xmax)   # 图形界限
    ax.set_ylim(ymin, ymax)   # 图形界限
    ax.set_xscale("log")      # 设置对数刻度
    plt.xlabel(r'$\log t$')   # 数轴标签
    plt.ylabel(r'$s$')        # 数轴标签
    ax.grid(True)             # 画出网格

    # 画出观测数据
    ax.plot(t, s, '*', label="观测值")
    # 画出拟合直线
    ax.plot(t, beta[0] + beta[1]*np.log10(t), label="Jacob")

    # 指定图例标题与内容的字体
    plt.legend(prop={'family':'Simsun'}, loc=4, title="图例",
               title_fontproperties={'family':'SimHei'})
    # 指定绘图标题为楷体
    ax.set_title("最小二乘法-Jacob 公式", fontproperties={'family':'KaiTi', 'size':
↪'large'})

    return fig
    plt.close('all')
```

准备数据，形成线性方程组并求解：

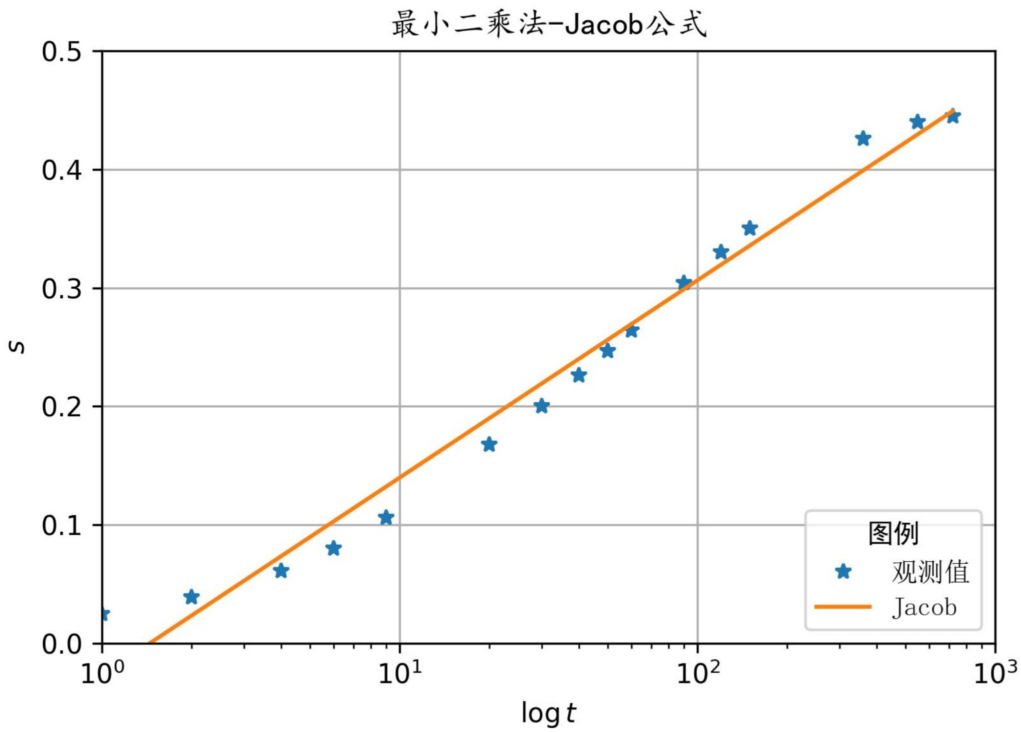
[45]:
```
# 输入数据
Q = 528/1440                  # m^3/min, 抽水量
r = 90                        # m, 观测孔位置
t = np.array([1, 2, 4, 6, 9, 20, 30, 40, 50, 60,
              90, 120, 150, 360, 550, 720])  # min
s = np.array([2.5, 3.9, 6.1, 8.0, 10.6, 16.8, 20.0, 22.6, 24.7, 26.4,
              30.4, 33.0, 35.0, 42.6, 44.0, 44.5])/100    # m
```

```
# 形成方程组
X = np.vstack([np.ones(len(t)), np.log10(t)]).T
A = np.dot(X.T, X)
b = np.dot(X.T, s)

# 求解方程组，beta 返回解
beta = np.linalg.solve(A, b)
```

输出计算结果：

[46]:
```
# 输出拟合系数
print('     beta = ', beta)
# 计算误差平方和
print('residuals = {:.4e}'
      .format(np.sum((s - beta[0] - beta[1]*np.log10(t))**2)))

# 计算 T 与 S
T = 0.183*Q/beta[1]
S = 2.25*T/r**2/np.float_power(10, beta[0]/beta[1])
# 计算 u
u = r**2*S/4/T/t

# 输出结果
print(' T = {:.4f}'.format(T), ', S = {:.4e}'.format(S))
print(' u')
print(u)

# 绘图
fig = plot_jacob(t, s, beta)
```

```
     beta =  [-0.0267  0.1665]
residuals = 6.6459e-03
 T = 0.4030 , S = 1.6203e-04
 u
[0.8142 0.4071 0.2036 0.1357 0.0905 0.0407 0.0271 0.0204 0.0163 0.0136
 0.009  0.0068 0.0054 0.0023 0.0015 0.0011]
```



[47]:
```
fig.savefig("5.4.jpg")
# plt.close("all")
# widgets.FloatSlider.close_all()
```

最小二乘法无法判断该用那些数据拟合直线，若结果不满意需要对数据筛选。

可以看出，前 4 个数据不满足 $u < 0.1$，并且最后 3 组数据没有呈现直线，可能是由于补给使其偏离直线.

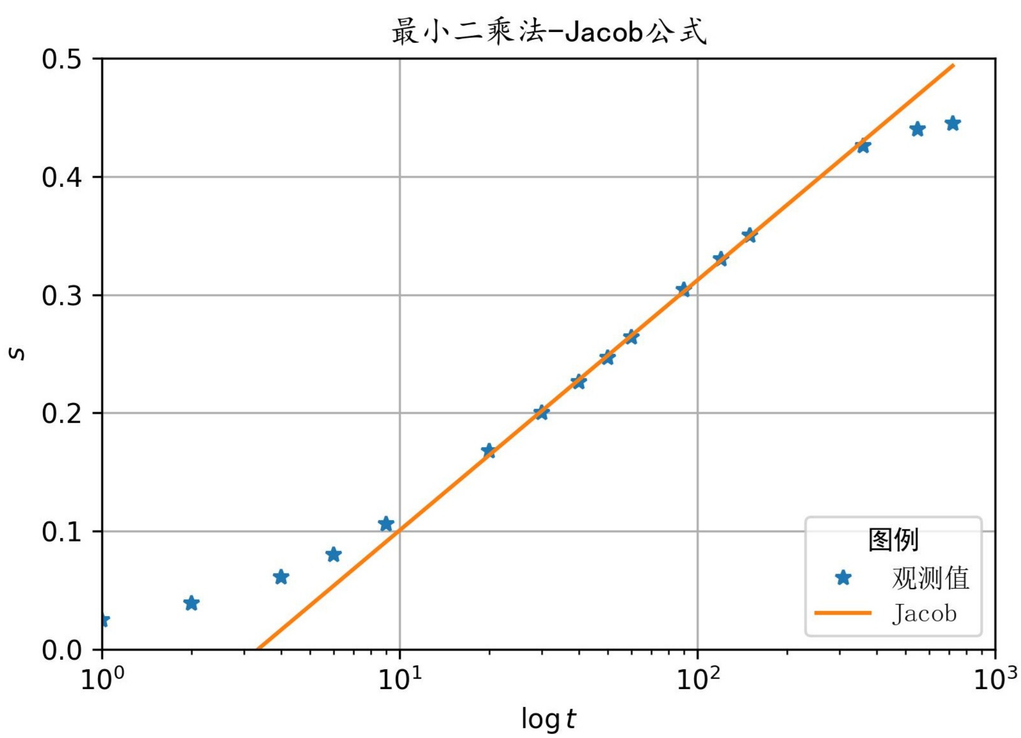
重新筛选数据：

前 5 组与后 3 组数据不用。

[48]:
```
# 重新筛选数据，前 5 组与后 3 组数据不用
# 形成方程组
X = np.vstack([np.ones(len(t[5:-3])), np.log10(t[5:-3])]).T
A = np.dot(X.T, X)
b = np.dot(X.T, s[5:-3])

# 求解方程组，beta 返回解
beta = np.linalg.solve(A, b)
# 输出拟合系数
print('      beta = ', beta)
# 计算误差平方和
print('residuals = {:.4e}'
      .format(np.sum((s - beta[0] - beta[1]*np.log10(t))**2)))
```

```
# 计算 T 与 S
T = 0.183*Q/beta[1]
S = 2.25*T/r**2/np.float_power(10, beta[0]/beta[1])
# 计算 u
u = r**2*S/4/T/t

# 输出结果
print(' T = {:.4f}'.format(T), ', S = {:.4e}'.format(S))
print(' u')
print(u)

# 绘图
fig = plot_jacob(t, s, beta)
```

```
    beta =  [-0.1109  0.2116]
residuals = 3.2021e-02
 T = 0.3172 , S = 2.9447e-04
 u
[1.8801 0.94   0.47   0.3133 0.2089 0.094  0.0627 0.047  0.0376 0.0313
 0.0209 0.0157 0.0125 0.0052 0.0034 0.0026]
```



[49]:
```
fig.savefig("5.5.jpg")
# plt.close("all")
# widgets.FloatSlider.close_all()
```

练习 7：根据 **例 2** 观 1、观 2 孔数据用 Jacob 公式的最小二乘法线性拟合求水文参数。

思考题：

(1) 选择拟合直线数据应该考虑哪些问题？

(2) 试用观 1、观 2 孔等多个观测孔求参，并比较结果。

提示：

数组拼接用 `np.concatenate()` (参考 2.1 Theis 公式的配线法)；

筛选数据要对数组排序，`np.argsort()` 返回排序后的索引，按这个索引从数组中取元素可以保证原有的对应关系不变。

6.4 Theis 公式的最小二乘法非线性拟合

Theis 公式

$$s = \frac{Q}{4\pi T}\int_u^{\infty}\frac{1}{y}e^{-y}dy = \frac{Q}{4\pi T}W(u) \tag{6.3}$$

取初始参数 (T_0, S_0)，降深 s 在 (T_0, S_0) 处的泰勒级数展开为

$$s(T,S) = s(T_0,S_0) + \left.\frac{\partial s}{\partial S}\right|_{S=S_0}\Delta S + \left.\frac{\partial s}{\partial T}\right|_{T=T_0}\Delta T + O(\Delta T^2 + \Delta S^2) \tag{6.4}$$

式中，$\Delta T = T - T_0$, $\Delta S = S - S_0$。因为 $\frac{\partial u}{\partial S} = \frac{u}{S}$, $\frac{\partial u}{\partial T} = -\frac{u}{T}$ ，得

$$\frac{\partial s}{\partial S} = -\frac{Q}{4\pi T}\frac{e^{-u}}{S}, \quad \frac{\partial s}{\partial T} = \frac{Q}{4\pi T^2}\left[-W(u) + e^{-u}\right]$$

记

$$a = \left.\frac{\partial s}{\partial S}\right|_{S=S_0}, \quad b = \left.\frac{\partial s}{\partial T}\right|_{T=T_0}, \quad c = s(T_0,S_0) \tag{6.5}$$

忽略高阶无穷小，(6.4) 式可写为

$$\hat{s} = c + a\cdot\Delta S + b\cdot\Delta T \tag{6.6}$$

式 (6.6) 以 (ΔT, ΔS) 为未知系数，$\hat{s}$ 为 s 的预测值，a, b, c 根据初始参数（T_0, S_0）及观测时间计算，由此式可构造最小二乘问题求参。

例 10：根据 **例 4** 观 2 的数据用 Theis 公式的最小二乘法线性拟合求水文参数。

Python 程序

导入库：

[50]:
```
%matplotlib ipympl

import numpy as np
import matplotlib.pyplot as plt
from scipy.special import exp1

# 控制小数的显示精度
np.set_printoptions(precision=4)
```

定义一些函数，用于计算 u 值、降深、微分项、误差等：

[51]:
```
def calc_u(p, r, t):   # 计算变量 u
    S, T = p
    return r**2*S/4/T/t

# 定义井函数计算方法，可用 scipy.special.exp1 计算.
# 也可用多项式逼近的方法计算井函数
def theis_drawdown(p, t, Q, r):  # 计算降深
    S, T = p
    u = calc_u(p, r, t)
    s_theis = Q/4/np.pi/T*exp1(u)
    return s_theis

def theis_Dfun(p, t, s, Q, r):  # 计算 ds/dT,ds/S
    """Calculate and return ds/dS and ds/dT for the Theis equation.
    The parameters S, T are packed into the tuple p.
    """
    S, T = p
    u = calc_u(p, r, t)
    dsdS = -Q/4/np.pi/T/S*np.exp(-u)
    dsdT = Q/4/np.pi/T**2*(-exp1(u) + np.exp(-u))
    return np.array((-dsdS, -dsdT)).T   # 返回负梯度

def theis_resid(p, t, s, Q, r): # 计算降深预测误差
    S, T = p
```

```
    return s - theis_drawdown(p, t, Q, r)
```

准备数据：

[52]:
```
# 抽水量 Q, 观测孔位置 r
r = 140.0  # m, 观 2 孔位置
Q = 1440   # m^3/d

t = np.array([10., 20., 30., 40., 60., 80., 100., 120., 150., \
              210., 270., 330., 400., 450., 645., 870., 990., 1185.])/1440  # 单位 day
s = np.array([0.16, 0.48, 0.54, 0.65, 0.75, 1., 1.12, 1.22, 1.36, 1.55, 1.7, \
              1.83, 1.89, 1.98, 2.17, 2.38, 2.46, 2.54])  # m
```

计算初始参数：

[53]:
```
# 初始参数怎么取？最简单方法是取中间相邻的两点，用 Jacob 两点公式计算。
# 取数组长度
i1 = int(len(t)/2)
i2 = i1 + 1
slope = (s[i1] - s[i2])/np.log10(t[i1]/t[i2])

T0 = 0.183*Q/slope
S0 = 2.25*T0*t[i1]/r**2/np.float_power(10, s[i1]/slope)

p = S0, T0
S, T = p
```

循环求最优参数：

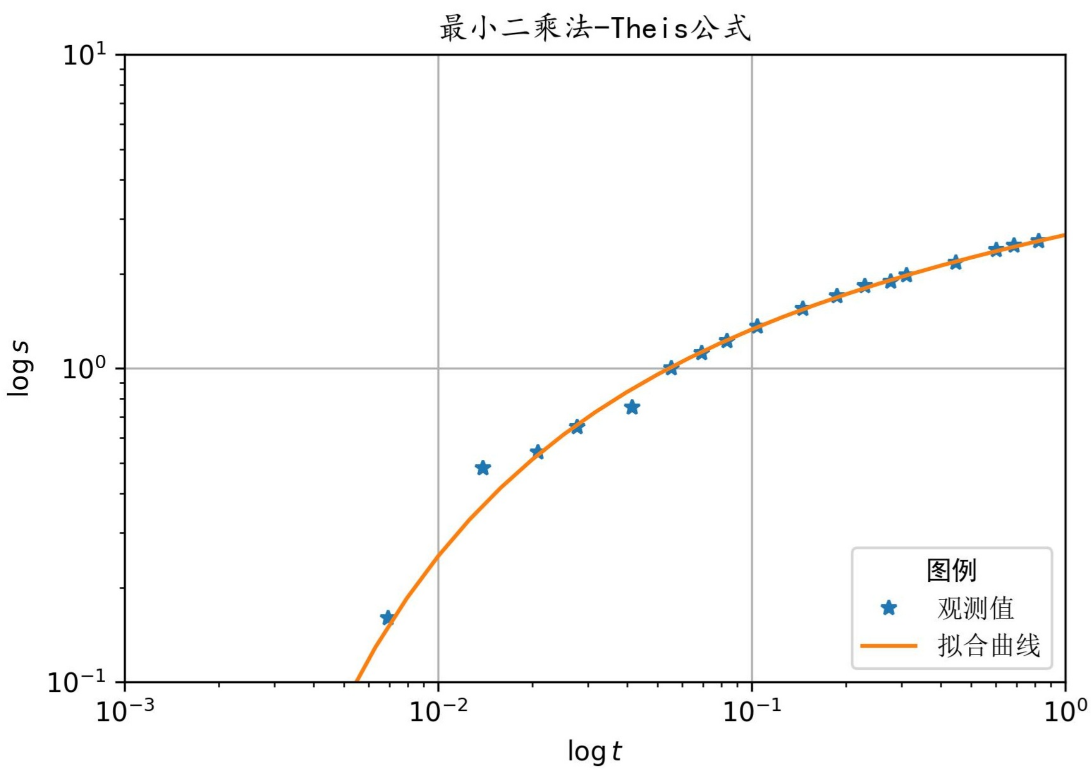
[54]:
```
# 循环计数器
j = 0
while True:
    c = theis_drawdown(p, t, Q, r)
    a = theis_Dfun(p, t, s, Q, r)[:, 0]
    b = theis_Dfun(p, t, s, Q, r)[:, 1]
    X = np.vstack([a, b]).T  # 形成系数矩阵
    # 调用 np.linalg.solve
    beta = np.linalg.solve(np.dot(X.T, X), np.dot(X.T, c - s))
    DS = beta[0]
    DT = beta[1]

    while True:
        if S + DS < 0:    # 步长太大导致参数为负数，不合理！
```

```
            DS = DS/2.0  # 减小步长
        else:
            break
    while True:
        if T + DT < 0:    # 步长太大导致参数为负数，不合理！
            DT = DT/2.0  # 减小步长
        else:
            break
    j = j + 1  # 循环计数器
    if j > 1000:  # 循环次数多，程序可能有错误
        print(' 循环超过 1000 次，请先检查程序再运行！ ')
        break
    if (abs(DT) < 1.0E-6) or (abs(DS) < 1.0e-8):  # 判断计算误差
        break
    # 新参数值
    p = S + DS, T + DT
    S, T = p
```

绘图输出：

[55]:
```
# 生成绘图数据
x0 =[i for i in np.arange(-3, 0.1, 0.1)]
x = np.power(10, x0)
y = theis_drawdown(p, x, Q, r)

# 图形设置
plt.style.use('default')
fig, ax = plt.subplots(dpi=100)
ax.grid(True)
ax.set_xscale("log")
ax.set_yscale("log")
ax.set_xlim(0.001, 1)
ax.set_ylim(0.1, 10)
ax.set_aspect(1)

# 绘制观测数据与拟合曲线
ax.plot(t, s, '*', label='观测值')
ax.plot(x, y, label='拟合曲线')
plt.xlabel(r'$\log t$')
plt.ylabel(r'$\log s$')

# 指定图例标题与内容的字体
```

```
plt.legend(prop={'family':'Simsun'}, loc=4, title="图例",
           title_fontproperties={'family':'SimHei'})
# 指定绘图标题为楷体
ax.set_title("最小二乘法-Theis 公式", fontproperties={'family':'KaiTi', 'size':'large'})

# 显示图形
plt.show()

# 显示最终参数结果
print('循环次数 = {}'.format(j))
print('T = {:.2f} m^2/d,'.format(T), 'S = {:.4e}'.format(S))
print('RSS = {:.4e}'.format(np.sqrt(np.sum((c - s)**2))))
```




```
循环次数 = 3
T = 193.38 m^2/d, S = 2.5011e-04
RSS = 1.6944e-01
```

[56]:
```
fig.savefig("5.6.jpg")
# plt.close("all")
# widgets.FloatSlider.close_all()
```

注意：程序设计需要考虑以下问题：

- 最小二乘法得出的 $(T_0+\Delta T, S_0+\Delta S)$ 有时为不合理的负值，用二分法缩小步长可保证参数为正值；
- 最小二乘法只是在初值的基础上进行了一步优化。为得到最优参数，需用得出的参数作为新的初值重复计算；
- 初值对最终结果有影响。选取两组观测数据按 Jacob 公式可计算出合理的参数初值。

7 附录：井函数计算

7.1 Theis 井函数

$$W(u)=\int_u^\infty \frac{\mathrm{e}^{-y}}{y}\mathrm{d}y=\mathrm{E}_1(u)=-\mathrm{E}_i(-u)$$

scipy.special.exp1 可以计算井函数, 也可用多项式逼近。

[57]:
```python
import numpy as np
import scipy.special as sps

def theis(u):
    """
    scipy.special.exp1 计算 Theis 井函数.
        u = r^2S/(4Tt), u 为数组时返回数组。
    """
    return sps.exp1(u)

u = np.array([10**x for x in range(-4, 1)])
print(theis(u))
```

```
[8.6332 6.3315 4.0379 1.8229 0.2194]
```

$W(u)$ 的多项式逼近

- $0<u\le 1$ 时

$$W(u)=-\ln u+a_0+a_1u+a_2u^2+a_3u^3+a_4u^4+a_5u^5$$

式中

$$\begin{aligned}
a_0 &= -0.57721566 & a_3 &= 0.05519968\\
a_1 &= 0.99999193 & a_4 &= -0.00976004\\
a_2 &= -0.24991055 & a_5 &= 0.00107857
\end{aligned}$$

- $1 < u < \infty$ 时

$$W(u) = \frac{b_0 + b_1 u + b_2 u^2 + b_3 u^3 + u^4}{c_0 + c_1 u + c_2 u^2 + c_3 u^3 + u^4} \cdot \frac{e^{-u}}{u}$$

式中

$$\begin{aligned}
b_0 &= 0.2677737343 & c_0 &= 3.9584969228 \\
b_1 &= 8.6347608925 & c_1 &= 21.0996530827 \\
b_2 &= 18.0590169730 & c_2 &= 25.6329561486 \\
b_3 &= 8.5733287401 & c_3 &= 9.5733223454
\end{aligned}$$

Python 程序

[58]:
```
import numpy as np

def theis1(u):
    """
    多项式逼近方法计算 Theis 井函数.
        u = r^2S/(4Tt), u 为数组时返回数组。
    """

    def wellfunc(u):

        a = [-0.57721566, 0.99999193, -0.24991055, 0.05519968, -0.00976004, 0.00107857]
        b = [0.2677737343, 8.6347608925, 18.059016973, 8.5733287401]
        c = [3.9584969228, 21.0996530827, 25.6329561486, 9.5733223454]

        if u <= 1:
            w = -np.log(u) + a[0] + u*(a[1] + u*(a[2] + u*(a[3] + u*(a[4] + u*a[5]))))
        else:
            w = c[0] + u*(c[1] + u*(c[2] + u*(c[3] + u)))
            w = (b[0] + u*(b[1] + u*(b[2] + u*(b[3] + u))))/w
            w = w*np.exp(-u)/u

        return w

    well = np.vectorize(wellfunc) # 向量化函数

    return 1.0*well(u)

u = np.array([10**x for x in range(-4, 1)])
```

```python
print(theis1(u))
```

```
[8.6332 6.3315 4.0379 1.8229 0.2194]
```

7.2 Hantush-Jacob 井函数

$$W(u,\beta)=\int_u^{\infty}\frac{1}{y}\exp\Big(-y-\frac{\beta^2}{4y}\Big)dy,\quad W(u,\beta)=2K_0(\beta)-W\Big(\frac{\beta^2}{4u},\beta\Big)$$

式中：$u=\frac{r^2S}{4Tt}$，$\beta=\frac{r}{B}$.

级数形式（Hunt，1977）

$$W(u,\beta)=\sum\nolimits_{n=0}^{\infty}\Big(-\frac{\beta^2}{4u}\Big)^n\frac{E_{n+1}(u)}{n!}$$

式中，

$$E_n(u)=\int_1^{\infty}\frac{e^{-uy}}{y^n}dy=u^{n-1}\int_u^{\infty}\frac{e^{-y}}{y^n}dy\quad(n=0,1,2,\cdots;\Re u>0)$$

为指数积分，当 $\frac{\beta^2}{4u}<1$ 时级数快速收敛。

`scipy.special.expn` 计算指数积分，`scipy.special.k0` 计算 0 阶第二类修正 Bessel 函数 $K_0(x)$。

Python 程序

[59]:
```python
import numpy as np
import scipy.special as sps

# 计算 hantush - jacob 井函数程序
def hantush_jacob(u, beta):
    """
    指数积分级数方法计算 hantush - Jacob 井函数.
        u = r^2 S/(4Tt);
        beta = r/B;
        u, beta 可以为数组并返回数组。
    """
    def wellfunc(u, beta):
        if u < 0:
            print('Negative are not allowed')
            return np.nan
        if u == 0:                     # 稳定解
            return 2.0*sps.k0(beta)
```

```python
        r = 1
        t = beta**2/(4*u)
        b = 2*u

        if beta <= b:          # beta < 2u
            W = 0
            n = 0
            term = r*sps.expn(n + 1, u)
            while np.abs(term) > 1e-10:
                W = W + term
                n = n + 1
                r = r*(-t)/n
                term = r*sps.expn(n + 1, u)
        else:
            W = 2.0*sps.k0(beta)
            n = 0
            term = r*sps.expn(n + 1, t)
            while np.abs(term) > 1e-10:
                W = W - term
                n = n + 1
                r = r*(-u)/n
                term = r*sps.expn(n + 1, t)

        return W

    # 向量化函数
    well = np.vectorize(wellfunc)
    # 返回值
    return 1.0*well(u, beta)

beta = 0.05
u = np.array([10**x for x in range(-4, 1)])
print(hantush_jacob(u, beta))
```

```
[6.2282 5.7965 3.9795 1.8184 0.2193]
```