

计算物理作业 7

杨远青 22300190015  CompPhys 24

2024 年 11 月 22 日

在尝试抵御 GPT 的诱惑！

1 题目 1：单摆运动积分

1.1 题目描述

Write a code to numerically solves the motion of a simple pendulum using **Euler's method**, **midpoint method**, **RK4 method** and **Euler-trapezoidal method** (implement these methods by yourself). Plot the angle and total energy as a function of time. Explain the results.

1.2 程序描述

本程序内置了一个 Pendulum 类，具有绳长，质量（小球视作质点），初始角度，初始角速度，重力加速度等属性。通过调用 Pendulum 类的方法，可以使用 Euler's method, midpoint method, RK4 method 和 Euler-trapezoidal method 来求解简单摆的运动，会返回角度与角速度的 numpy 数组。类的方法还包括辅助的导数计算，即演化方程

$$\begin{aligned}\frac{d\theta}{dt} &= \omega, \\ \frac{d\omega}{dt} &= -\frac{g}{L} \sin(\theta),\end{aligned}$$

与总能量采集方法

$$E = T + V = \frac{1}{2}m(\omega L)^2 + mgL(1 - \cos \theta)$$

主程序还有内置的解析解、误差计算与用户输入采集函数，其中解析解借助了 `scipy.special` 的雅可比椭圆积分 `sn, cn`，模数 $k = \sin(\theta_0/2)$ ，固有频率 $\omega_0 = \sqrt{\frac{g}{L}}$ ，所以对大角度的摆动也是精确的。

$$\theta(t) = 2 \arcsin \left(k \operatorname{sn} \left(\omega_0 t + \frac{\pi}{2}, k^2 \right) \right)$$

$$\omega(t) = \frac{2k\omega_0 \operatorname{cn} \left(\omega_0 t + \frac{\pi}{2}, k^2 \right)}{\sqrt{1 - k^2 \operatorname{sn}^2 \left(\omega_0 t + \frac{\pi}{2}, k^2 \right)}}$$

1.2.1 欧拉法 (Euler's Method)

$$\theta_{i+1} = \theta_i + h \cdot \frac{d\theta}{dt} \Big|_{t_i} \quad \omega_{i+1} = \omega_i + h \cdot \frac{d\omega}{dt} \Big|_{t_i}$$

1.2.2 中点法 (Midpoint Method)

$$\text{计算中点值: } \theta_{\text{mid}} = \theta_i + \frac{h}{2} \cdot \frac{d\theta}{dt} \Big|_{t_i}, \quad \omega_{\text{mid}} = \omega_i + \frac{h}{2} \cdot \frac{d\omega}{dt} \Big|_{t_i}$$

$$\text{使用中点斜率更新: } \theta_{i+1} = \theta_i + h \cdot \frac{d\theta}{dt} \Big|_{\text{mid}}, \quad \omega_{i+1} = \omega_i + h \cdot \frac{d\omega}{dt} \Big|_{\text{mid}}$$

1.2.3 四阶龙格-库塔法 (RK4 Method)

$$\text{第一步 } (k_1): k_1^\theta = \frac{d\theta}{dt} \Big|_{t_i, \theta_i, \omega_i}, \quad k_1^\omega = \frac{d\omega}{dt} \Big|_{t_i, \theta_i, \omega_i};$$

$$\text{第二步 } (k_2): k_2^\theta = \frac{d\theta}{dt} \Big|_{t_i + \frac{h}{2}, \theta_i + \frac{h}{2}k_1^\theta, \omega_i + \frac{h}{2}k_1^\omega}, \quad k_2^\omega = \frac{d\omega}{dt} \Big|_{t_i + \frac{h}{2}, \theta_i + \frac{h}{2}k_1^\theta, \omega_i + \frac{h}{2}k_1^\omega};$$

$$\text{第三步 } (k_3): k_3^\theta = \frac{d\theta}{dt} \Big|_{t_i + \frac{h}{2}, \theta_i + \frac{h}{2}k_2^\theta, \omega_i + \frac{h}{2}k_2^\omega}, \quad k_3^\omega = \frac{d\omega}{dt} \Big|_{t_i + \frac{h}{2}, \theta_i + \frac{h}{2}k_2^\theta, \omega_i + \frac{h}{2}k_2^\omega};$$

$$\text{第四步 } (k_4): k_4^\theta = \frac{d\theta}{dt} \Big|_{t_i + h, \theta_i + hk_3^\theta, \omega_i + hk_3^\omega}, \quad k_4^\omega = \frac{d\omega}{dt} \Big|_{t_i + h, \theta_i + hk_3^\theta, \omega_i + hk_3^\omega},;$$

$$\text{更新公式: } \theta_{i+1} = \theta_i + \frac{h}{6} (k_1^\theta + 2k_2^\theta + 2k_3^\theta + k_4^\theta), \quad \omega_{i+1} = \omega_i + \frac{h}{6} (k_1^\omega + 2k_2^\omega + 2k_3^\omega + k_4^\omega).$$

1.2.4 欧拉-梯形法 (Euler-Trapezoidal Method)

$$\text{预测: } \theta_{\text{pred}} = \theta_i + h \cdot \frac{d\theta}{dt} \Big|_{t_i}, \quad \omega_{\text{pred}} = \omega_i + h \cdot \frac{d\omega}{dt} \Big|_{t_i}$$

$$\text{校正: } \theta_{i+1} = \theta_i + \frac{h}{2} \left(\frac{d\theta}{dt} \Big|_{t_i} + \frac{d\theta}{dt} \Big|_{\text{pred}} \right) \quad \omega_{i+1} = \omega_i + \frac{h}{2} \left(\frac{d\omega}{dt} \Big|_{t_i} + \frac{d\omega}{dt} \Big|_{\text{pred}} \right)$$

请在 Problem_1/src 目录下运行 `python -u pendulum.py` 查看结果，需安装辅助计算的 `numpy`, `scipy` 库与绘图的 `matplotlib` 库，其中 `scipy` 库请使用 1.13 以上版本，旧版本的特殊函数可能不在 `scipy.special` 模块中。运行程序后，会提示输入求解参数，用户可以键入回车选择使用默认值，或自定义参数。

1.3 伪代码

Powered by `LATeX pseudocode generator`

Algorithm 1: Euler Method for Simple Harmonic Oscillator

Input: h : Time step size (float), N : Total number of steps (int)

Output: θ : Angle array (rad), ω : Angular velocity array (rad/s)

```

1 Initialize  $\theta[0] \leftarrow \theta_0$ ,  $\omega[0] \leftarrow \omega_0$ ;                                // Set initial conditions
2 for  $i \leftarrow 0$  to  $N - 1$  do
3   Compute  $(\dot{\theta}, \dot{\omega}) \leftarrow \text{Derivatives}(\theta[i], \omega[i])$ ;
4   Update  $\theta[i + 1] \leftarrow \theta[i] + h \cdot \dot{\theta}$ ,  $\omega[i + 1] \leftarrow \omega[i] + h \cdot \dot{\omega}$ ;          // Update values
5 end
6 return  $\theta, \omega$ ;                                         // Return results as arrays

```

Algorithm 2: Midpoint Method for Simple Harmonic Oscillator

Input: h : Time step size (float), N : Total number of steps (int)

Output: θ : Angle array (rad), ω : Angular velocity array (rad/s)

```
1 Initialize  $\theta[0] \leftarrow \theta_0$ ,  $\omega[0] \leftarrow \omega_0$  ; // Set initial conditions
2 for  $i \leftarrow 0$  to  $N - 1$  do
3   Compute  $(\dot{\theta}, \dot{\omega}) \leftarrow \text{Derivatives}(\theta[i], \omega[i])$  ; // Slope at initial point
4   Compute  $\theta_{\text{mid}} \leftarrow \theta[i] + 0.5 \cdot h \cdot \dot{\theta}$ ,  $\omega_{\text{mid}} \leftarrow \omega[i] + 0.5 \cdot h \cdot \dot{\omega}$  ; // Midpoint values
5   Compute  $(\dot{\theta}_{\text{mid}}, \dot{\omega}_{\text{mid}}) \leftarrow \text{Derivatives}(\theta_{\text{mid}}, \omega_{\text{mid}})$  ; // Slope at midpoint
6   Update  $\theta[i + 1] \leftarrow \theta[i] + h \cdot \dot{\theta}_{\text{mid}}$ ,  $\omega[i + 1] \leftarrow \omega[i] + h \cdot \dot{\omega}_{\text{mid}}$  ; // Update values
7 end
8 return  $\theta, \omega$  ; // Return results as arrays
```

Algorithm 3: RK4 Method for Simple Harmonic Oscillator

Input: h : Time step size (float), N : Total number of steps (int)

Output: θ : Angle array (rad), ω : Angular velocity array (rad/s)

```
1 Initialize  $\theta[0] \leftarrow \theta_0$ ,  $\omega[0] \leftarrow \omega_0$  ; // Set initial conditions
2 for  $i \leftarrow 0$  to  $N - 1$  do
3   Compute  $(k_1^\theta, k_1^\omega) \leftarrow \text{Derivatives}(\theta[i], \omega[i])$  ; // Stage 1
4   Compute  $(k_2^\theta, k_2^\omega) \leftarrow \text{Derivatives}(\theta[i] + 0.5 \cdot h \cdot k_1^\theta, \omega[i] + 0.5 \cdot h \cdot k_1^\omega)$  ; // Stage 2
5   Compute  $(k_3^\theta, k_3^\omega) \leftarrow \text{Derivatives}(\theta[i] + 0.5 \cdot h \cdot k_2^\theta, \omega[i] + 0.5 \cdot h \cdot k_2^\omega)$  ; // Stage 3
6   Compute  $(k_4^\theta, k_4^\omega) \leftarrow \text{Derivatives}(\theta[i] + h \cdot k_3^\theta, \omega[i] + h \cdot k_3^\omega)$  ; // Stage 4
7   Update  $\theta[i + 1] \leftarrow \theta[i] + \frac{h}{6} \cdot (k_1^\theta + 2 \cdot k_2^\theta + 2 \cdot k_3^\theta + k_4^\theta)$  ;
8   Update  $\omega[i + 1] \leftarrow \omega[i] + \frac{h}{6} \cdot (k_1^\omega + 2 \cdot k_2^\omega + 2 \cdot k_3^\omega + k_4^\omega)$  ;
9 end
10 return  $\theta, \omega$  ; // Return results as arrays
```

Algorithm 4: Euler-Trapezoidal Method for Simple Harmonic Oscillator

Input: h : Time step size (float), N : Total number of steps (int)

Output: θ : Angle array (rad), ω : Angular velocity array (rad/s)

```
1 Initialize  $\theta[0] \leftarrow \theta_0$ ,  $\omega[0] \leftarrow \omega_0$  ; // Set initial conditions
2 for  $i \leftarrow 0$  to  $N - 1$  do
3   Compute  $(\dot{\theta}, \dot{\omega}) \leftarrow \text{Derivatives}(\theta[i], \omega[i])$  ; // Predictor step slopes
4   Compute  $\theta_{\text{pred}} \leftarrow \theta[i] + h \cdot \dot{\theta}$ ,  $\omega_{\text{pred}} \leftarrow \omega[i] + h \cdot \dot{\omega}$  ; // Euler predictor values
5   Compute  $(\dot{\theta}_{\text{pred}}, \dot{\omega}_{\text{pred}}) \leftarrow \text{Derivatives}(\theta_{\text{pred}}, \omega_{\text{pred}})$  ; // Corrector step slopes
6   Update  $\theta[i + 1] \leftarrow \theta[i] + \frac{h}{2} \cdot (\dot{\theta} + \dot{\theta}_{\text{pred}})$ ,  $\omega[i + 1] \leftarrow \omega[i] + \frac{h}{2} \cdot (\dot{\omega} + \dot{\omega}_{\text{pred}})$  ; // Trapezoidal corrector
7 end
8 return  $\theta, \omega$  ; // Return results as arrays
```

1.4 结果示例

以下结果均使用默认配置，即绳长 $L = 1.0m$, 质量 $m = 1.0kg$, 初始角度 $\theta_0 = 1.0rad$, 初始角速度 $\omega_0 = 0rad/s$, 重力加速度 $g = 9.81m/s^2$, 时间步长 $h = 0.05$, 总步数 $N = 1000$, 总时间 $T = 50.0s$ 。用户可以通过终端输入更改这

些参数。在角速度随角度变化 $\omega - \theta$ 图中, 为凸显效果, 增加时间步长至 $h' = 0.1s$, 总步数不变, 总时间至 $T' = 100.0s$.

```
(base) gilbert@Gilbert-YoungMacBook src % python -u pendulum.py  
请输入摆的参数(直接回车使用默认值):  
摆长 L (m) (默认: 1.0):  
质量 m (kg) (默认: 1.0):  
重力加速度 g (m/s2) (默认: 9.81):  
初始角度 θ0 (rad) (默认: 1.0):  
初始角速度 ω0 (rad/s) (默认: 0.0):  
时间步长 h (s) (默认: 0.05):  
总模拟时间 T (s) (默认: 50.0):
```

图 1: 终端处理用户输入, 此处均采用默认值

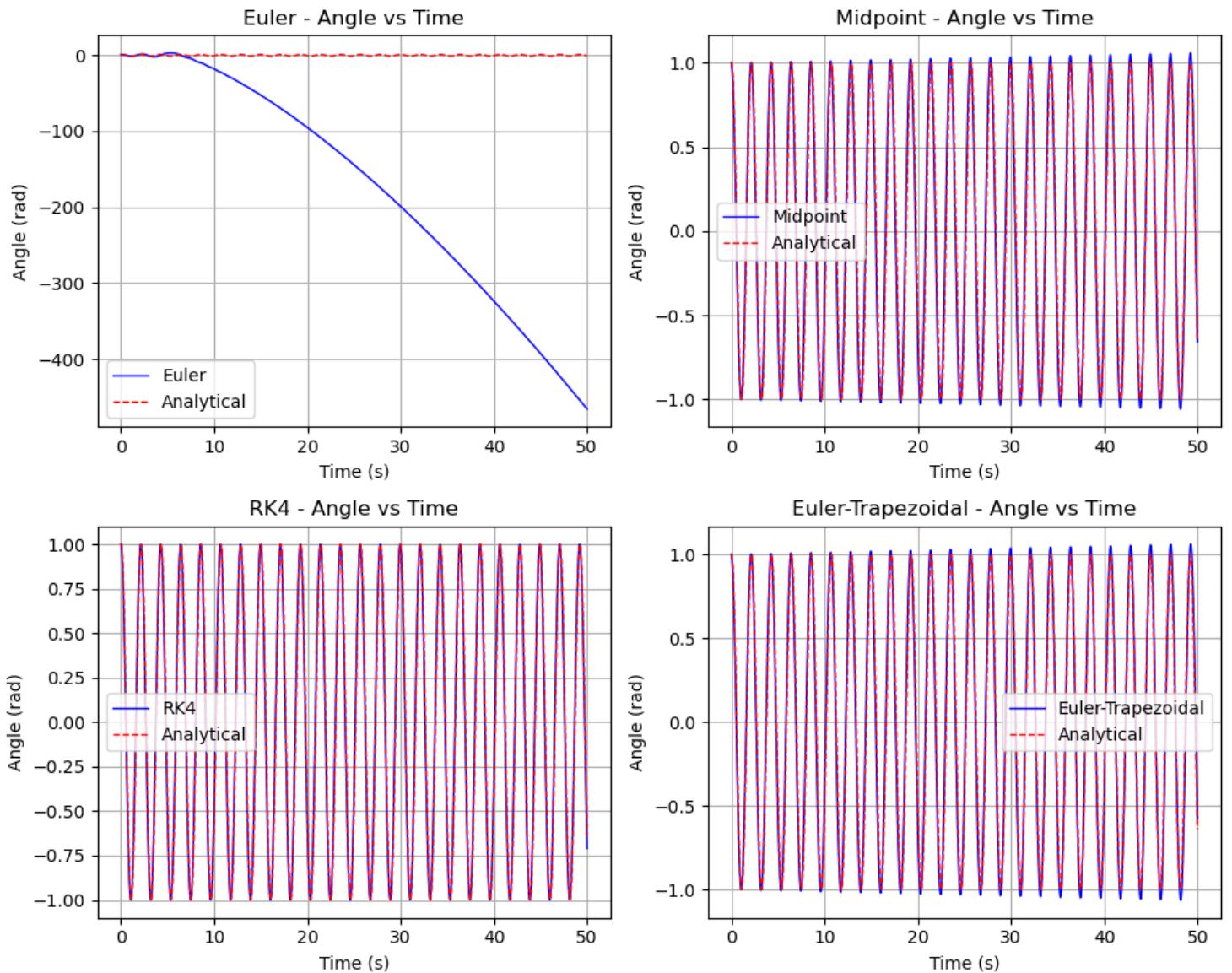


图 2: 角度随时间变化 $\theta - t$ 图

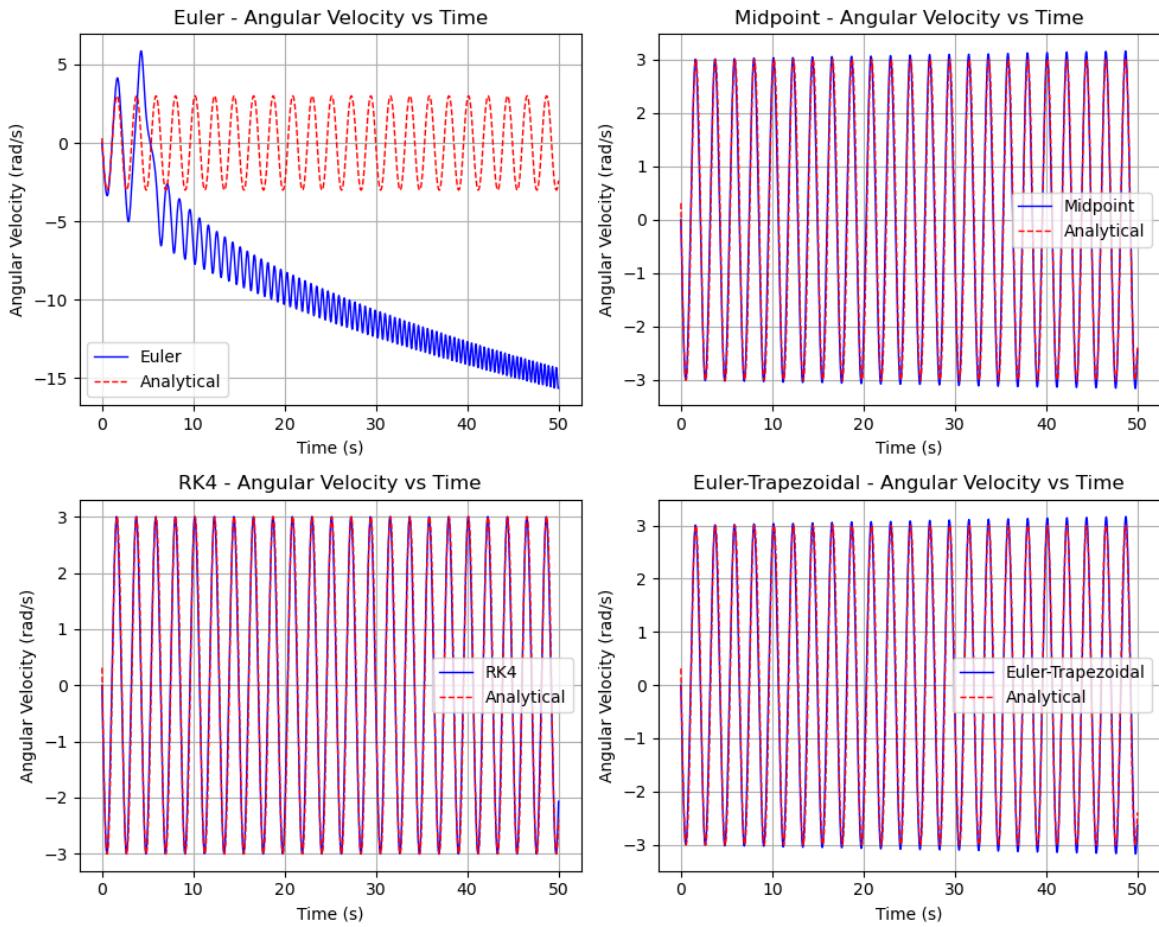


图 3: 角速度随时间变化 $\omega - t$ 图

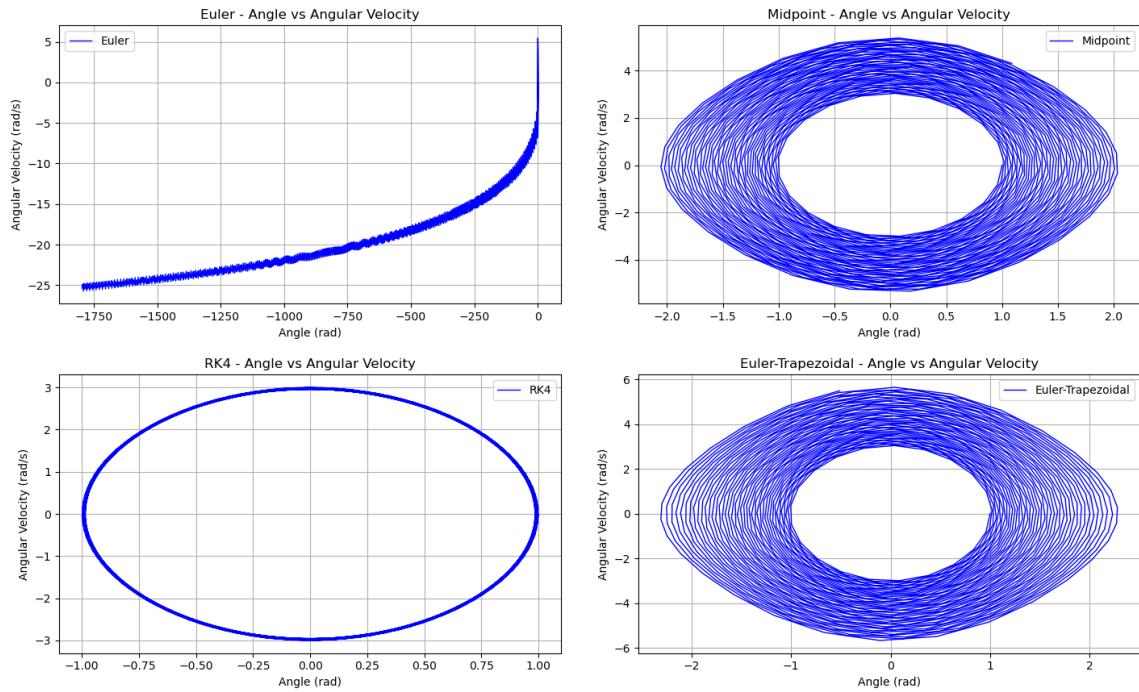


图 4: 角速度随角度变化 $\omega - \theta$ 图

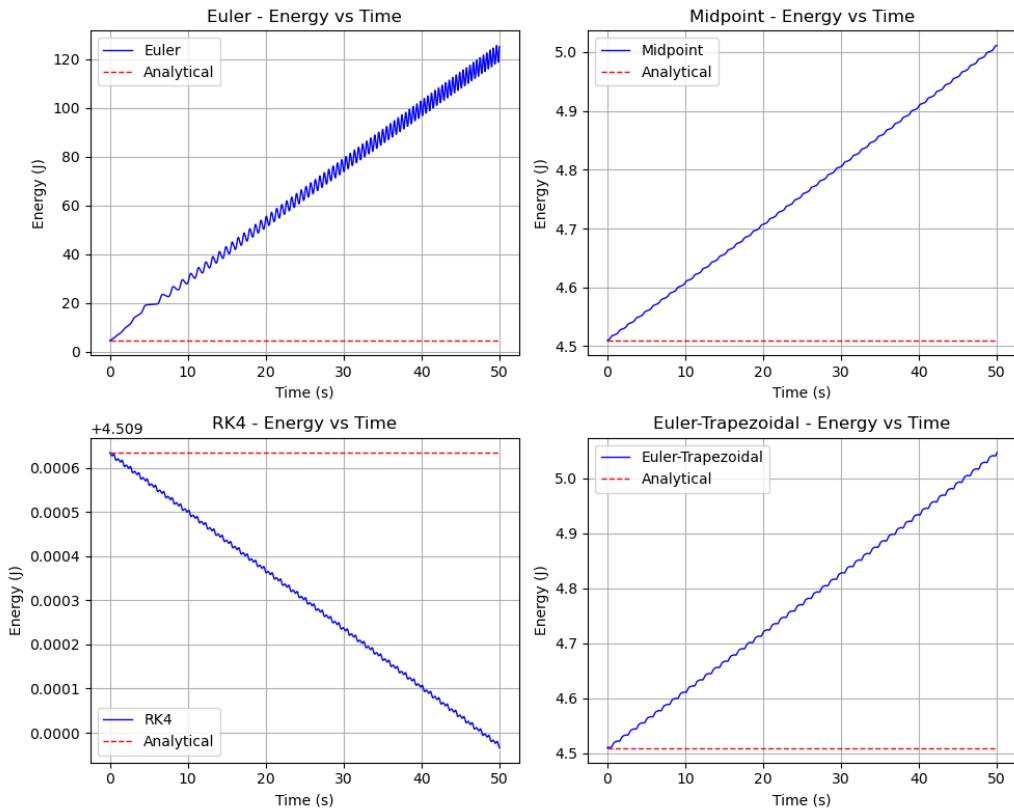


图 5: 能量随时间漂移 $E - t$ 图

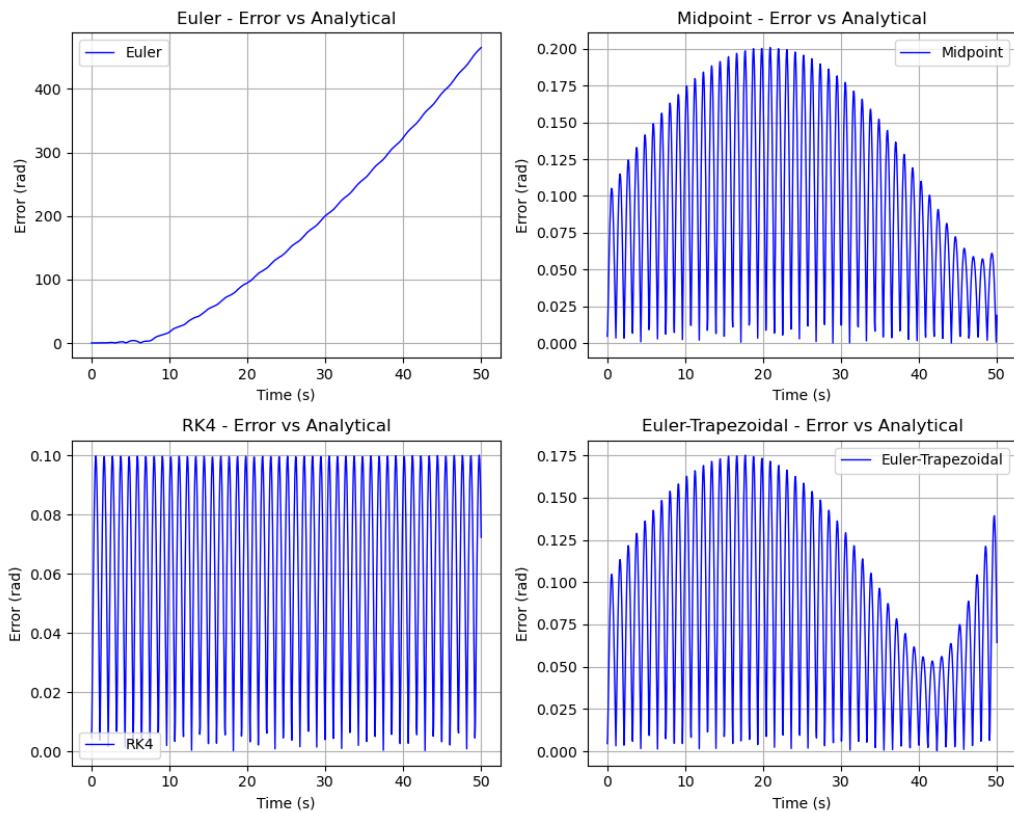


图 6: 角度与解析解误差随时间变化 $\delta\theta - t$ 图

可以看出，四种方法对比中，RK4 方法的能量漂移最小，角度与角速度误差也最小。中点法与欧拉-梯形法次之，虽然角度、角速度误差在长时间后才逐渐显现，但能量漂移较大。欧拉法误差最大，一段时间后就崩溃。综上，RK4 方法最为精确且稳定。能量漂移图与角速度随角度变化的相空间演化图，均证明其保辛性良好。

2 题目 2：径向薛定谔方程求解

2.1 题目描述

Write a code to numerically solve the radial Schrödinger equation for

$$\left[-\frac{1}{2} \nabla^2 + V(\mathbf{r}) \right] \psi(\mathbf{r}) = E\psi(\mathbf{r}), \quad V(\mathbf{r}) = V(r)$$

1. $V(r) = \frac{1}{r}$ (hydrogen atom)
2. Considering the following potential:

$$V(r) = -\frac{Z_{\text{ion}}}{r} \operatorname{erf}\left(\frac{r}{\sqrt{2}r_{\text{loc}}}\right) + \exp\left[-\frac{1}{2}\left(\frac{r}{r_{\text{loc}}}\right)^2\right] \times \left[C_1 + C_2 \left(\frac{r}{r_{\text{loc}}}\right)^2 + C_3 \left(\frac{r}{r_{\text{loc}}}\right)^4 + C_4 \left(\frac{r}{r_{\text{loc}}}\right)^6 \right]$$

where erf is the error function. And for Li, you could set:

- $Z_{\text{ion}} = 3$
- $r_{\text{loc}} = 0.4$
- $C_1 = -14.0093922$
- $C_2 = 9.5099073$
- $C_3 = -1.7532723$
- $C_4 = 0.0834586$

Compute and plot the first three eigenstates. You could find more information about 'how to solve radial Schrödinger equation' and 'use of non-uniform grid (optional)' in the PPT.

Special Note: You may call any library functions for diagonalization.

2.2 程序描述

本程序拥有四个模块：`utils`, `solver`, `analysis`, `visualization`, 分别为工具函数和配置参数模块、数值求解算法模块、结果分析和处理模块与可视化处理模块。主程序入口 `main` 中有主求解器类 `RadialSchrodingerSolver`, 其计算配置与非均匀网格由 `utils` 中的 `SolverConfig` 类与 `RadialGrid` 类指定；势能函数调用 `utils` 中的 `Potential` 基类，其中有本题设定的氢原子库仑势与锂原子局域势，`utils` 中还有一些辅助工具，在此不一一列举。主求解器的核心逻辑依赖绑定的求解器实例 `ShootingSolver` 或 `FiniteDifferenceSolver`¹。主求解器还从 `analysis` 模块绑定了波函数处理器 `WavefunctionProcessor`, 负责计算导数、分析波函数渐进行为、根据不同角量子数由邻域外推原点附近的波函数奇异值、归一化处理等；能量分析器 `EnergyAnalyzer`, 负责比较能级与理论值差异；收

¹Numerov 实在没时间写了 qwq

敛性分析器 `ConvergenceAnalyzer`, 负责分析不同网格下的收敛性。最后, 来自可视化模块 `visualization` 中的 `ResultVisualizer` 负责波函数、概率密度与收敛分析结果的可视化。

采用的非均匀网格为指数网格, 共 $j_{\max} + 1$ 个点, 参数 r_p 满足 $r_{\max} = r(j_{\max})$, 控制 $j_{\max}\delta = 6$

$$r(j) = r_p(\exp(j\delta) - 1) + r_{\min}, \quad j = 0, 1, 2, \dots, j_{\max}$$

在本题单位制 $h = m = a = 1$ 下, 一维径向方程化为 (注意课件上漏了个 2)

$$u''(r) = 2(E - V_{\text{eff}}(r))u(r)$$

此时做变换

$$u(j) = v(j) \exp(j\delta/2)$$

可将方程改写为

$$v''(j) - \frac{\delta^2}{4}v(j) = 2r_p^2\delta^2 \exp(2j\delta)(E - V_{\text{eff}}(r(j)))v(j)$$

于是我们可以愉快地使用 RK4 求解了 (结果示例中验证了 $\mathcal{O}(h^4)$ 的全局误差)。一般的打靶法从原点向外积分, 但此处原点附近势能奇异, 故改从一个较大的 r_{\max} 向内积分。但考虑到在 l 给定的情况下, 不同能级 n 对应的演化方程相同, 很可能出现求解不到指定态, 或者无法收敛的情形。课件最后的参考文献²指出, 可以对 $u(r)$ 施加 $(n - l - 1)$ 的节点约束, 以确保求解到正确的态。但他似乎采取加一个较大的惩罚系数, 如 $1e3 * (\text{nodes} - \text{target})$, 这样会使得一般的求根器容易出错, 我选择改用 $(\text{nodes} - \text{target})^2$, 使得求解更平稳, 并先在求解区间内粗扫描, 再在 r_{\min} 最小的值附近使用 `scipy.optimize.minimize` 的 `l-bfgs-b` 求解, 在测试中发现, 一些边界奇异的态需要该用 `Nelder-Mead` 无导数优化器, 但效率与精度均下降。为了减少奇异解的发生, 我还添加了波函数渐进行为约束与连续性约束, 均要求在接近原点附近的几个点满足

$$\frac{u'(r)}{u(r)} \approx \frac{l+1}{r}$$

不过前者是使用 $[(\frac{d}{dr} \ln u) \cdot r - (l+1)]^2$, 后者直接考虑 $[u'(r) - \frac{l+1}{r}u(r)]^2$ 作为权重函数, 这样可以保证在原点附近的波函数行为符合物理要求。加上节点数要求与目标 $u(r_{\min})$ 的要求, 总共四个目标函数可以加权求和作为最小化的目标, 经其约束的方法大大提高了求解的稳定性与收敛性。

在有限差分法中, 直接将方程两侧除以 $r_p^2\delta^2 \exp(2j\delta)$ 可以按照普通本征值问题求解, 但实际操作发现这样的数值稳定性不太好。本程序进行两处改进, 首先将 $j = 0$ 的近核点从方程移除, 其实保留也能求解, 但因为该变换限制了第一个点与第二个点较近, 且离心势能的 $\frac{1}{r^2}$ 奇异会带来一些麻烦, 如图7所示 而我们求解的 u 总是在第一个点趋于 0 的, 所以去掉可以提高数值稳定性。其次, 我们原先的方程也是可以直接求解的, 属于广义本征值问题 $\mathbf{A}u = \lambda \mathbf{B}u$, 而 `scipy` 内置的库对其迭代有优化, 可加快收敛, 只不过要构造两个系数矩阵。借助本题氢原子与类氢原子的能级 $E_n \approx \frac{1}{n^2}$ 趋势, 我还使用了 shift-invert 模式, 即在基态的 $\frac{1}{n^2}$ 附近找寻激发态³, 提高了求解速度, 但在已知的 3p 态测试中, 经常导致漏掉该态直接找寻 4p 态。

可在 `Problem_2/src/radial_schrodinger` 目录下运行 `pip install -e .` 临时安装本包, 然后导入或直接运行 `python main.py`, 如果不安装, 也可以在 `Problem_2/src` 目录下运行 `python radial_schrodinger.py`, 均支持命令行参数, 请加上 `-h` 查看帮助信息。助教老师还可以借助 `Sphinx.html`⁴ 查看本包的文档。

²Solving The Stationary One Dimensional Schrödinger Equation With The Shooting Method

³令人匪夷所思的是, 早期版本中 $j_{\max} < 1000$ 的网格基本都能收敛求解, 但不知改动了哪里, 现在在一些稍小网格数测试中也经常不收敛。

⁴Github Pages 上的静态版本控制格式有些问题, 本地文档可正常使用

<pre> def construct_hamiltonian(self): """构建哈密顿量矩阵, 对应方程: -[v''(j) - v(j)\delta^2/4]/(2\delta^2 r p^2 e^{(2\delta j)}) + v(j) V_eff(j) = Ev(j) """ N = len(self.j) - 1 # jmax+1个点, 去掉最后一个点 H = lil_matrix((N, N)) # 为了数值稳定性, 每个方程同乘 2\delta^2 r p^2 e^{(2\delta j)} # 这样方程变成: -[v''(j) - v(j)\delta^2/4] + 2\delta^2 r p^2 e^{(2\delta j)}v(j) = E[2\delta^2 r p^2 e^{(2\delta j)}]v(j) for i in range(N): exp_factor = (2 * self.delta**2 * self.r_p**2 * np. exp(2 * self.delta * self.j[i])) # 动能项系数 if i > 0: H[i, i - 1] = -1 # v(j-1)系数 H[i, i] = 2 + self.delta**2 / 4 # v(j)系数 if i < N - 1: H[i, i + 1] = -1 # v(j+1)系数 # 势能项 H[i, i] += exp_factor * self.V_eff(self.j[i]) return H.tocsr() def fd_solve(self, n_states: int) -> Tuple[np.ndarray, np.ndarray]: </pre>	<pre> print(H_test) [21] ✓ 0.0s Python (0, 0) -252.1280372360556 ... (0, 1) -1.0 ... (1, 0) -1.0 ... (1, 1) 1.9280926160509535 ... (1, 2) -1.0 ... (2, 1) -1.0 ... (2, 2) 1.96372079826546 ... (2, 3) -1.0 ... (3, 2) -1.0 ... (3, 3) 1.9755971954334994 ... (3, 4) -1.0 ... (4, 3) -1.0 ... (4, 4) 1.9815343852906082 ... (4, 5) -1.0 ... (5, 4) -1.0 ... (5, 5) 1.9850956827960027 ... (5, 6) -1.0 ... (6, 5) -1.0 ... (6, 6) 1.9874689706056279 ... (6, 7) -1.0 ... (7, 6) -1.0 ... (7, 7) 1.989163368434465 ... (7, 8) -1.0 ... (8, 7) -1.0 ... (8, 8) 1.9904334450184151 (298, 298) 1.9969384267577557 ... (298, 299) -1.0 ... (299, 298) -1.0 ... (299, 299) 1.9969236559544192 </pre>
---	--

图 7: 去掉第一个点前的病态矩阵

2.3 伪代码

Powered by [LATEX pseudocode generator](#)

2.3.1 打靶法

Algorithm 5: Inward Integration with RK4

Input: E : Energy (float)

Output: u : Normalized wavefunction array (radial grid)

- 1 Initialize $v[j_{\max}] \leftarrow 0$, $v'[j_{\max}] \leftarrow -1$; $\text{// Boundary conditions}$
- 2 Set step size $h \leftarrow -1$; $\text{// Negative for inward integration}$
- 3 **for** $j \leftarrow j_{\max} - 1$ **to** 0 **do**
- 4 Compute $(k_1^v, k_1^{v'}) \leftarrow \text{Derivative}(j + 1, v[j + 1], v'[j + 1])$;
- 5 Compute $(k_2^v, k_2^{v'}) \leftarrow \text{Derivative}(j + 0.5, v[j + 1] + 0.5 \cdot h \cdot k_1^v, v'[j + 1] + 0.5 \cdot h \cdot k_1^{v'})$;
- 6 Compute $(k_3^v, k_3^{v'}) \leftarrow \text{Derivative}(j + 0.5, v[j + 1] + 0.5 \cdot h \cdot k_2^v, v'[j + 1] + 0.5 \cdot h \cdot k_2^{v'})$;
- 7 Compute $(k_4^v, k_4^{v'}) \leftarrow \text{Derivative}(j, v[j + 1] + h \cdot k_3^v, v'[j + 1] + h \cdot k_3^{v'})$;
- 8 Update $v[j] \leftarrow v[j + 1] + \frac{h}{6} \cdot (k_1^v + 2 \cdot k_2^v + 2 \cdot k_3^v + k_4^v)$;
- 9 Update $v'[j] \leftarrow v'[j + 1] + \frac{h}{6} \cdot (k_1^{v'} + 2 \cdot k_2^{v'} + 2 \cdot k_3^{v'} + k_4^{v'})$;
- 10 **end**
- 11 Transform $u \leftarrow v \cdot \exp(\delta \cdot j/2)$; $\text{// Transform to radial wavefunction}$
- 12 Normalize u if $\|u\| > 0$; $\text{// Ensure normalization}$
- 13 **return** u

Algorithm 6: Objective Function for Energy Optimization

Input: E : Energy (float)

Output: Error: Objective function value

1 Compute $u \leftarrow \text{IntegrateInward}(E)$; $\text{// Solve radial Schrödinger equation}$

2 Compute r, u' ; $\text{// Radial positions and derivatives}$

3 **Step 1: Logarithmic Derivative Error**; $\text{// Near } r_{\min}$

4 Compute LogDerError based on u and u' ; $\text{// Match theoretical values}$

5 **Step 2: Node Count Error**; $\text{// Compare nodes with target}$

6 Compute NodesError based on node difference;

7 **Step 3: Amplitude Error at r_{\min}** ;

8 Compute AmpError $\leftarrow u[0]^2$;

9 **Step 4: Continuity Error**;

10 Compute ContError $\leftarrow (u'[0] - \text{TheoreticalDerivative})^2$;

11 Compute TotalError $\leftarrow w_1 \cdot \text{LogDerError} + w_2 \cdot \text{NodesError} + w_3 \cdot \text{AmpError} + w_4 \cdot \text{ContError}$;

12 **return** $TotalError$

Algorithm 7: Shooting Method for Energy Eigenvalue and Wavefunction

Input: E_{\min}, E_{\max} : Energy bounds, N_{target} : Target node count

Output: E_{optimal} : Optimal energy, u : Normalized wavefunction

1 Set $N_{\text{target}} \leftarrow n - l - 1$ if not specified;

2 Define weights w_1, w_2, w_3, w_4 for multi-objective terms;

3 **Step 1: Coarse Grid Search**;

4 Generate energy grid $E_{\text{grid}} \in [E_{\min}, E_{\max}]$;

5 Compute errors Errors[i] $\leftarrow \text{Objective}(E_{\text{grid}}[i])$;

6 Set $E_{\text{coarse}} \leftarrow \arg \min(\text{Errors})$;

7 **Step 2: Fine Optimization**;

8 Perform optimization Result $\leftarrow \text{Minimize}(\text{Objective}, \text{initial guess } E_{\text{coarse}})$;

9 **if** Result.success **then**

10 | Compute $u_{\text{optimal}} \leftarrow \text{IntegrateInward}(E_{\text{optimal}})$;

11 | **return** $E_{\text{optimal}}, u_{\text{optimal}}$;

12 **end**

13 **else**

14 | Raise(*RuntimeError*: Optimization did not converge)

15 **end**

2.3.2 有限差分法

Algorithm 8: Construct Hamiltonian Matrix

Output: H : Sparse Hamiltonian matrix

```
1 Initialize  $H$  as a sparse matrix with size  $(N_{\text{reduced}}, N_{\text{reduced}})$  ; // Reduced grid size
2 for  $i \leftarrow 0$  to  $N_{\text{reduced}} - 1$  do
3   Compute  $j_{\text{actual}} \leftarrow i + 1$  ; // Actual grid index
4   Compute  $\exp\_factor \leftarrow 2 \cdot \delta^2 \cdot r_p^2 \cdot \exp(2 \cdot \delta \cdot j_{\text{actual}})$ ;
5   if  $i > 0$  then
6     Set  $H[i, i - 1] \leftarrow -1$  ; // Kinetic term for  $j - 1$ 
7   end
8   Set  $H[i, i] \leftarrow 2 + \frac{\delta^2}{4} + \exp\_factor \cdot V_{\text{eff}}(j_{\text{actual}})$ ;
9   if  $i < N_{\text{reduced}} - 1$  then
10    Set  $H[i, i + 1] \leftarrow -1$  ; // Kinetic term for  $j + 1$ 
11  end
12 end
13 return  $H$ 
```

Algorithm 9: Construct B Matrix

Output: B : Sparse matrix for scaling factors

```
1 Initialize  $B$  as a sparse matrix with size  $(N_{\text{reduced}}, N_{\text{reduced}})$ ;
2 for  $i \leftarrow 0$  to  $N_{\text{reduced}} - 1$  do
3   Compute  $j_{\text{actual}} \leftarrow i + 1$ ;
4   Set  $B[i, i] \leftarrow 2 \cdot \delta^2 \cdot r_p^2 \cdot \exp(2 \cdot \delta \cdot j_{\text{actual}})$ ;
5 end
6 return  $B$ 
```

Algorithm 10: Finite Difference Solver for Eigenvalues and Eigenstates

Input: n_{states} : Number of eigenstates to compute

Output: (energies, u_states): Eigenvalues and normalized wavefunctions

1 Set $H \leftarrow \text{ConstructHamiltonian}();$

2 Set $B \leftarrow \text{ConstructB}(H.\text{shape}[0]);$

3 **Step 1: Compute Ground State;**

4 **Try**

5 | Compute $(e_{\text{ground}}, v_{\text{ground}}) \leftarrow \text{linalg.eigsh}(H, k = 1, M = B, \text{which} = "SA");$

6 **EndTry**

7 **Catch**

8 | Exception e

9 **EndCatch**

10 **Raise**(*RuntimeError*: *Ground state computation failed*);

11 Set energies $\leftarrow [e_{\text{ground}}]$, states $\leftarrow [v_{\text{ground}}];$

12 **Step 2: Compute Excited States (if $n_{\text{states}} > 1$);**

13 **for** $n \leftarrow 2$ **to** n_{states} **do**

14 | Compute estimated_e $\leftarrow e_{\text{ground}}/n^2$; // Estimate using $1/n^2$ scaling

15 | Set window $\leftarrow |e_{\text{ground}}| \cdot 0.1;$

16 | **foreach** shift $\in \{ \text{estimated_e}, \text{estimated_e} \pm \text{window} \}$ **do**

17 | **Try**

18 | | Compute $(e, v) \leftarrow \text{linalg.eigsh}(H, k = 1, M = B, \sigma = \text{shift}, \text{which} = "LM");$

19 | | **if** $|e - e_{\text{prev}}| > 1e^{-6}$ **for all** $e_{\text{prev}} \in \text{energies}$ **then**

20 | | Append e to energies, v to states;

21 | | Break;

22 | | end

23 | **EndTry**

24 | **Catch**

25 | | Exception e

26 | **EndCatch**

27 | Continue to the next shift ; // Skip this shift on failure

28 | **end**

29 **end**

30 **Step 3: Normalize Wavefunctions and Transform to $u(r)$;**

31 Set u_states $\leftarrow \text{Transform and normalize states};$

32 **return** (energies, u_states)

2.4 结果示例

2.4.1 打靶法计算结果

原子	量子数 n, l	能级 (单位: hartree)	原子	量子数 n, l	能级 (单位: hartree)
氢原子	$n = 1, l = 0, 1s$	-0.500000	锂原子	$n = 1, l = 0, 1s$	-4.458247
	$n = 2, l = 0, 2s$	-0.125000		$n = 2, l = 0, 2s$	-1.115432
	$n = 2, l = 1, 2p$	-0.125000		$n = 2, l = 1, 2p$	-1.122278
	$n = 3, l = 0, 3s$	-0.055556		$n = 3, l = 0, 3s$	-0.496256 ^b
	$n = 3, l = 1, 3p$	-0.055556 ^a		$n = 3, l = 1, 3p$	-0.498679
	$n = 3, l = 2, 3d$	-0.055556		$n = 3, l = 2, 3d$	-0.500262

^a 对于 $n = 3$ 的三个态，指定了 `r_Max=60`，精度大幅提高，但下面图片为展现边界效应未替换

^b 对于 $n = 3$ 的三个态，指定了 `r_Max=60`，否则无法收敛，甚至边界发散

2.4.2 有限差分法计算结果

原子	量子数 n, l	能级 (单位: hartree)	原子	量子数 n, l	能级 (单位: hartree)
氢原子	$n = 1, l = 0, 1s$	-0.499993	锂原子	$n = 1, l = 0, 1s$	-4.458266 ^d
	$n = 2, l = 0, 2s$	-0.125007		$n = 2, l = 0, 2s$	-1.115469
	$n = 2, l = 1, 2p$	-0.125003		$n = 2, l = 1, 2p$	-1.122296
	$n = 3, l = 0, 3s$	-0.055556 ^a		$n = 3, l = 0, 3s$	-0.496337
	$n = 3, l = 1, 3p$	-0.031255 ^b		$n = 3, l = 1, 3p$	-0.280662 ^e
	$n = 3, l = 2, 3d$	-0.055529 ^c		$n = 3, l = 2, 3d$	-0.500269 ^f

^a 在 `r_Max=30` 偏差过大，提升至 60

^b 实际上是 4p 态，但始终找不到正确的 3p，可能是窗口设置太小

^c 3d 态在 `r_Max=30` 表现尚好，因其极大值离核，边界效应不显著

^d 使用 `r_Max=30, j_Max=300`，表现良好

^e 实际上应该是 4p 态，但也始终找不到正确的 3p

^f 提升至 `j_Max=400` 后与打靶法结果差距更小

2.4.3 输入与异常处理

```
(base) gilbert@Gilbert-YoungMacBook radial_schrodinger % python main.py -h
usage: main.py [-h] [--V-type {hydrogen,lithium}] [--n N] [--l L] [--method {shooting,fd}] [--j-max J_MAX] [--example] [--convergence]

径向薛定谔方程求解器

options:
-h, --help            show this help message and exit
--V-type {hydrogen,lithium}
--n N                主量子数
--l L                角量子数
--method {shooting,fd}
                    求解方法
--j-max J_MAX        网格点数(可选)
--example            运行示例计算
--convergence        进行网格收敛性分析

(base) gilbert@Gilbert-YoungMacBook radial_schrodinger % python main.py --n 1 --l 1
2024-11-22 09:27:55,496 - ERROR - 程序运行失败: 角量子数l必须小于主量子数n
(base) gilbert@Gilbert-YoungMacBook radial_schrodinger % python main.py --n 2 --l -1
2024-11-22 09:29:23,510 - ERROR - 程序运行失败: 角量子数l必须为非负整数
(base) gilbert@Gilbert-YoungMacBook radial_schrodinger % python main.py --n 0 --l -1
2024-11-22 09:29:33,958 - ERROR - 程序运行失败: 主量子数n必须为正整数
(base) gilbert@Gilbert-YoungMacBook radial_schrodinger % python main.py --V-type silicon
usage: main.py [-h] [--V-type {hydrogen,lithium}] [--n N] [--l L] [--method {shooting,fd}] [--j-max J_MAX] [--example] [--convergence]
main.py: error: argument --V-type: invalid choice: 'silicon' (choose from 'hydrogen', 'lithium')
(base) gilbert@Gilbert-YoungMacBook radial_schrodinger % python main.py --method numerov
usage: main.py [-h] [--V-type {hydrogen,lithium}] [--n N] [--l L] [--method {shooting,fd}] [--j-max J_MAX] [--example] [--convergence]
main.py: error: argument --method: invalid choice: 'numerov' (choose from 'shooting', 'fd')
(base) gilbert@Gilbert-YoungMacBook radial_schrodinger % python main.py --method fd --j-max 800
2024-11-22 09:32:47,477 - ERROR - 程序运行失败: 有限差分法的j_max不能超过640
(base) gilbert@Gilbert-YoungMacBook radial_schrodinger % python main.py --method fd --j-max 40
2024-11-22 09:32:55,657 - ERROR - 程序运行失败: 网格点数过少, 建议至少50个点
```

图 8: 自定义参数输入与异常处理, 新版本支持网格起止点 `r_min` 与 `r_Max` 的指定

2.4.4 使用 example 选项运行的示例

```
(base) gilbert@Gilbert-YoungMacBook radial_schrodinger % python main.py --example
=====
求解hydrogen原子: n=1, l=0
使用shooting方法
=====
2024-11-22 09:37:58,656 - INFO - NumExpr defaulting to 8 threads.
2024-11-22 09:37:59,247 - INFO - 初始化hydrogen原子求解器: n=1, l=0, 方法=shooting
2024-11-22 09:38:03.102 python[3324:37418] +[IMKClient subclass]: chose IMKClient_Modern
2024-11-22 09:38:03.102 python[3324:37418] +[IMKInputSession subclass]: chose IMKInputSession_Modern
能量本征值: -0.500000 Hartree
理论值: -0.500000 Hartree
相对误差: 0.000001%
=====
求解hydrogen原子: n=2, l=0
使用shooting方法
=====
2024-11-22 09:38:04,020 - INFO - 初始化hydrogen原子求解器: n=2, l=0, 方法=shooting
能量本征值: -0.125000 Hartree
理论值: -0.125000 Hartree
相对误差: 0.000003%
=====
求解lithium原子: n=1, l=0
使用shooting方法
=====
2024-11-22 09:38:12,891 - INFO - 初始化lithium原子求解器: n=1, l=0, 方法=shooting
能量本征值: -4.458247 Hartree
理论值: -4.458247 Hartree
相对误差: 0.000000%
=====
进行氢原子1s态 shooting方法 网格收敛性分析
=====
2024-11-22 09:38:14,392 - INFO - 初始化hydrogen原子求解器: n=1, l=0, 方法=shooting
=====
进行氢原子1s态 fd方法 网格收敛性分析
```

图 9: 运行示例的终端输出

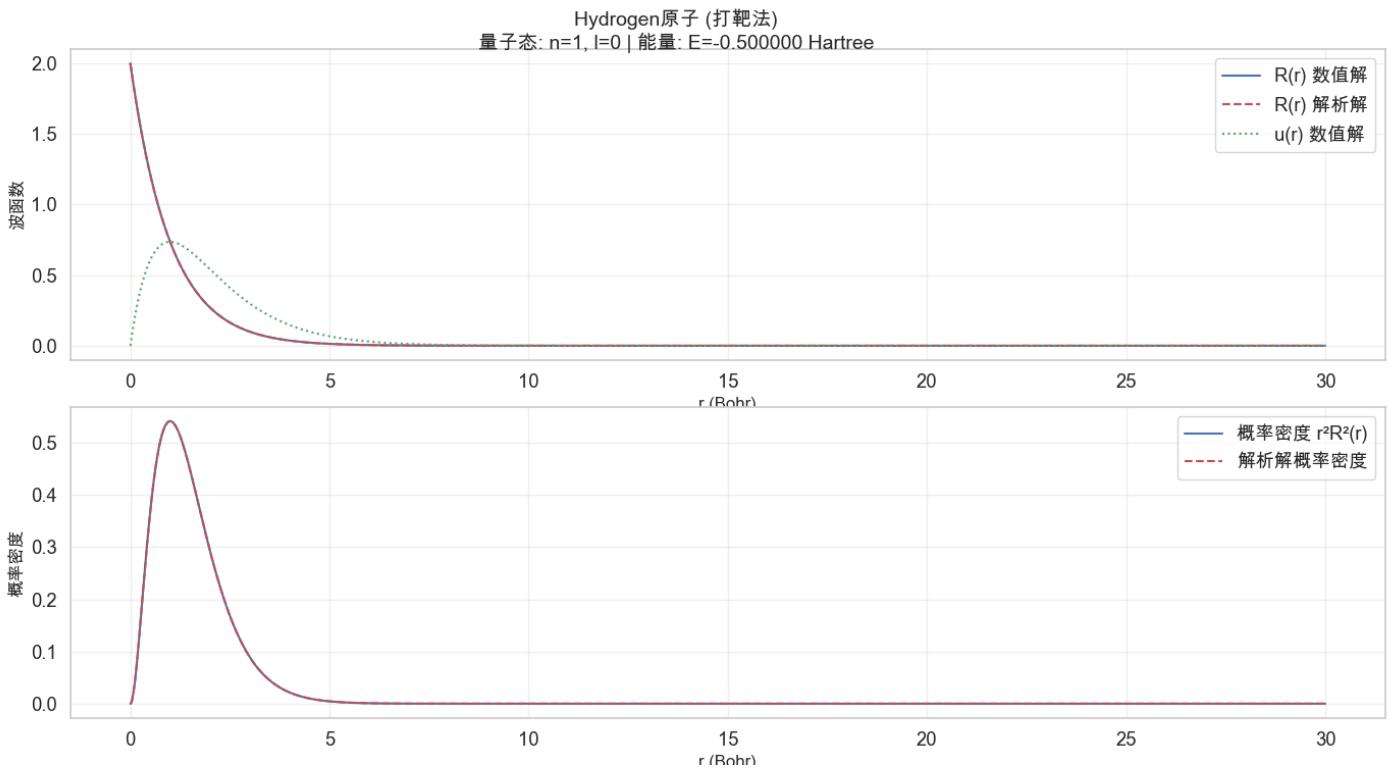


图 10: 使用打靶法求解氢原子库仑势的 1s 态

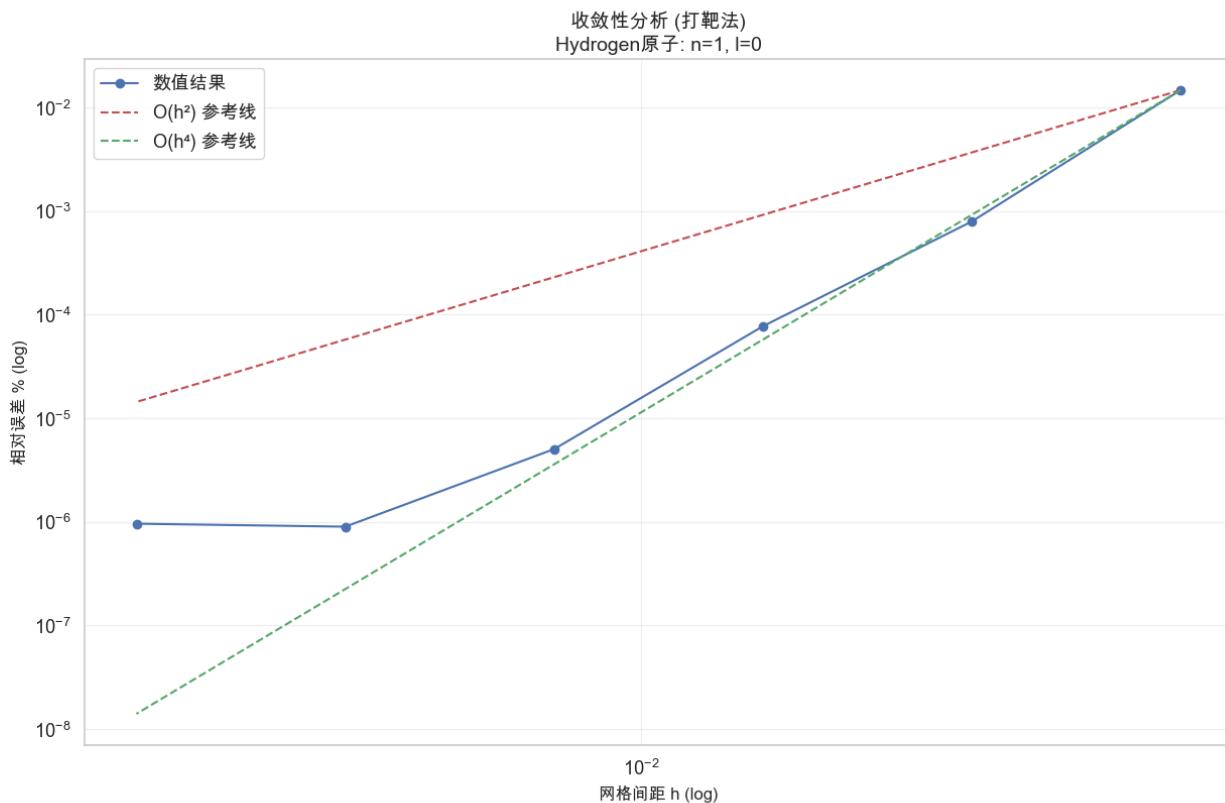
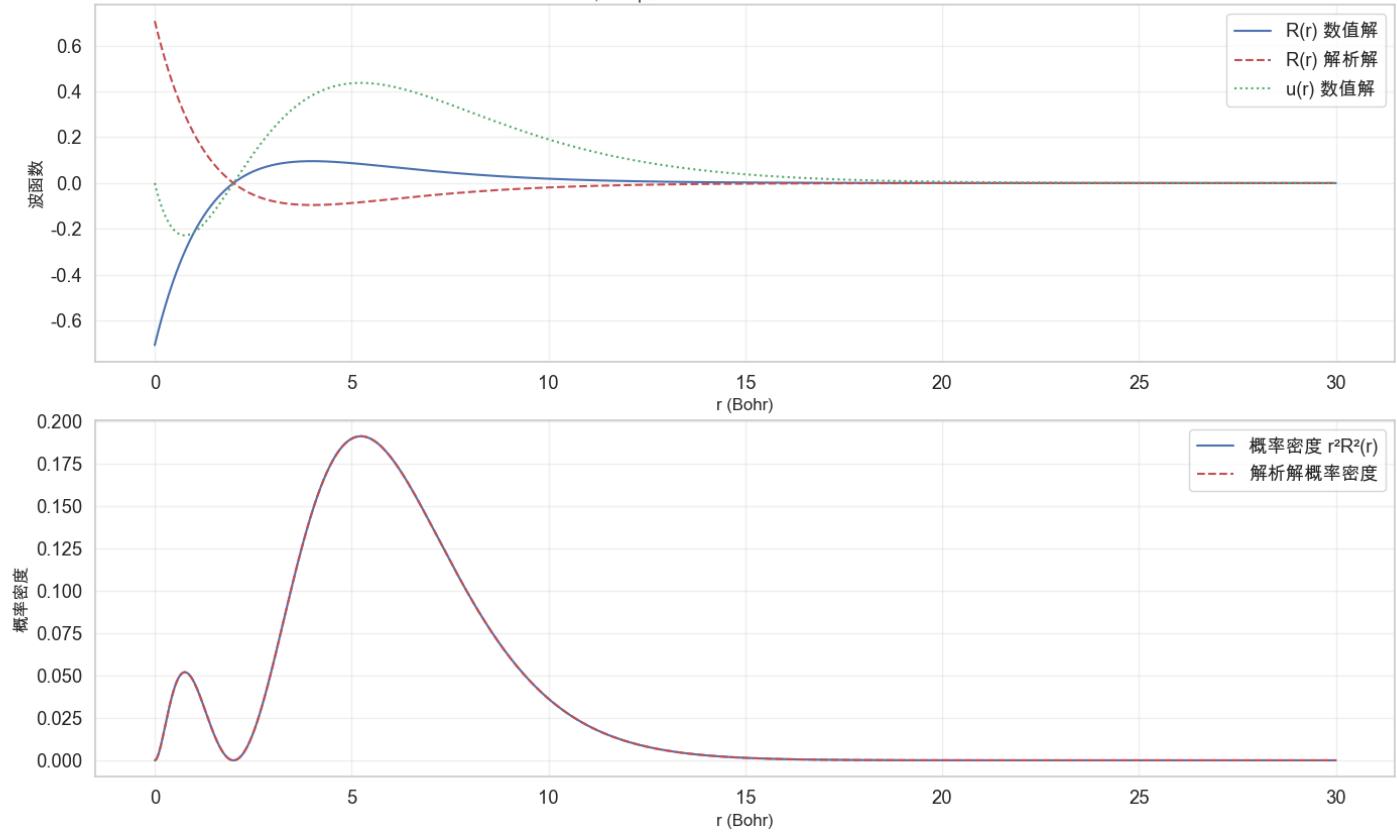


图 11: 使用打靶法求解氢原子库仑势的 1s 态的收敛性分析, 验证了 RK4 全局误差是四阶精度的

Hydrogen原子 (打靶法)
量子态: n=2, l=0 | 能量: E=-0.125000 Hartree



Hydrogen原子 (打靶法)
量子态: n=2, l=1 | 能量: E=-0.125000 Hartree

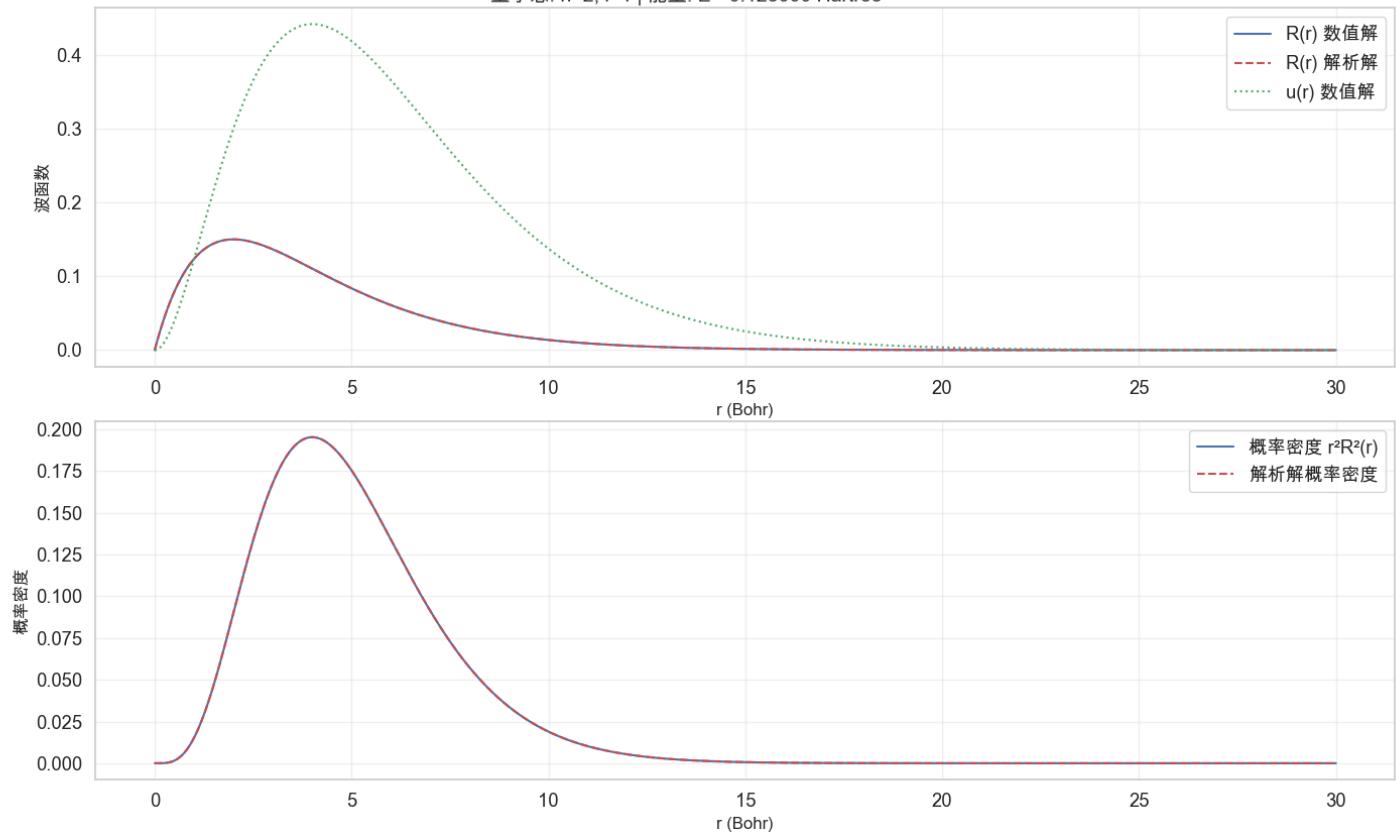


图 12: 使用打靶法求解氢原子库仑势的 2s,2p 态

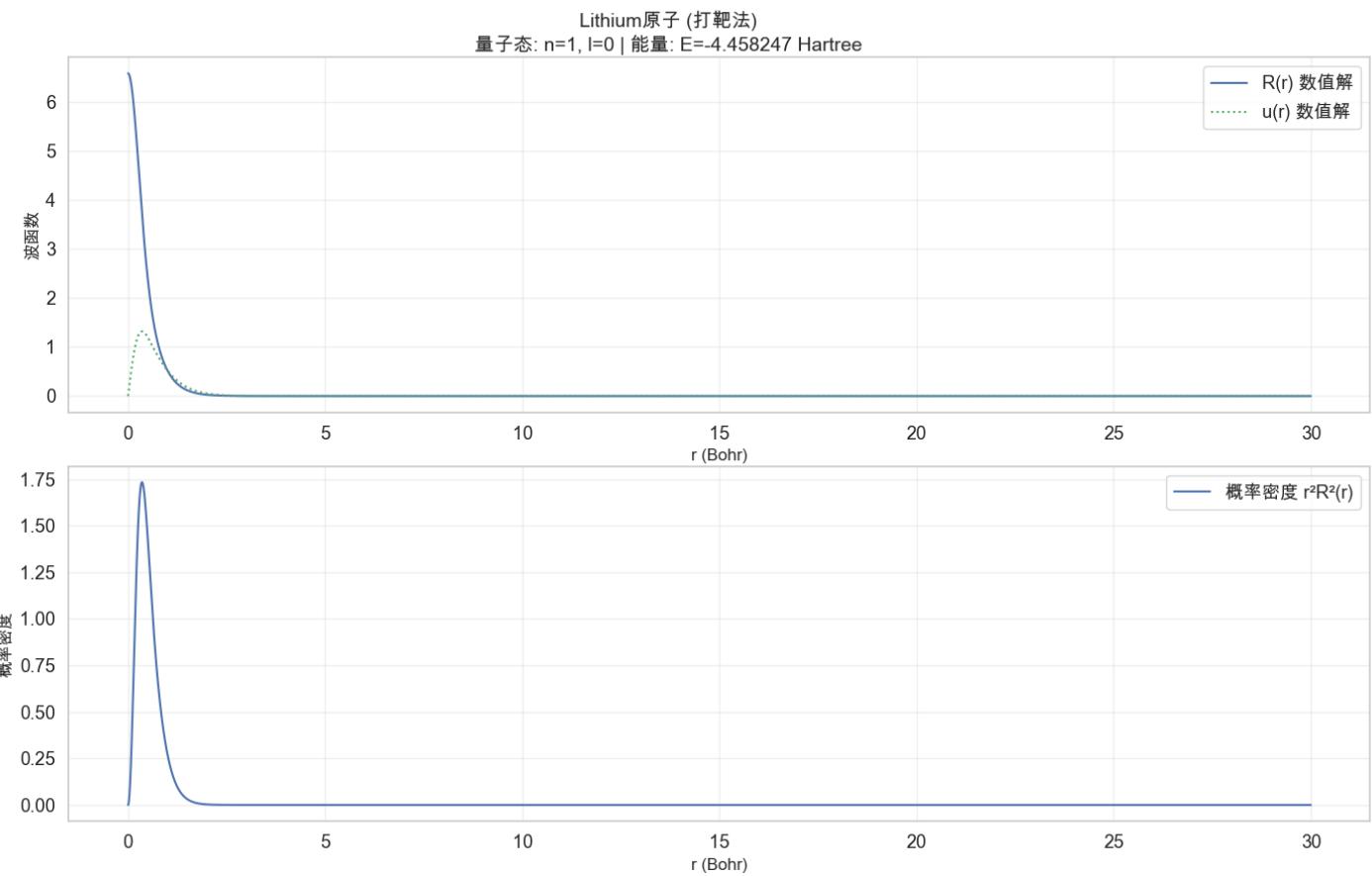


图 13: 使用打靶法求解锂原子局域势的 1s 态

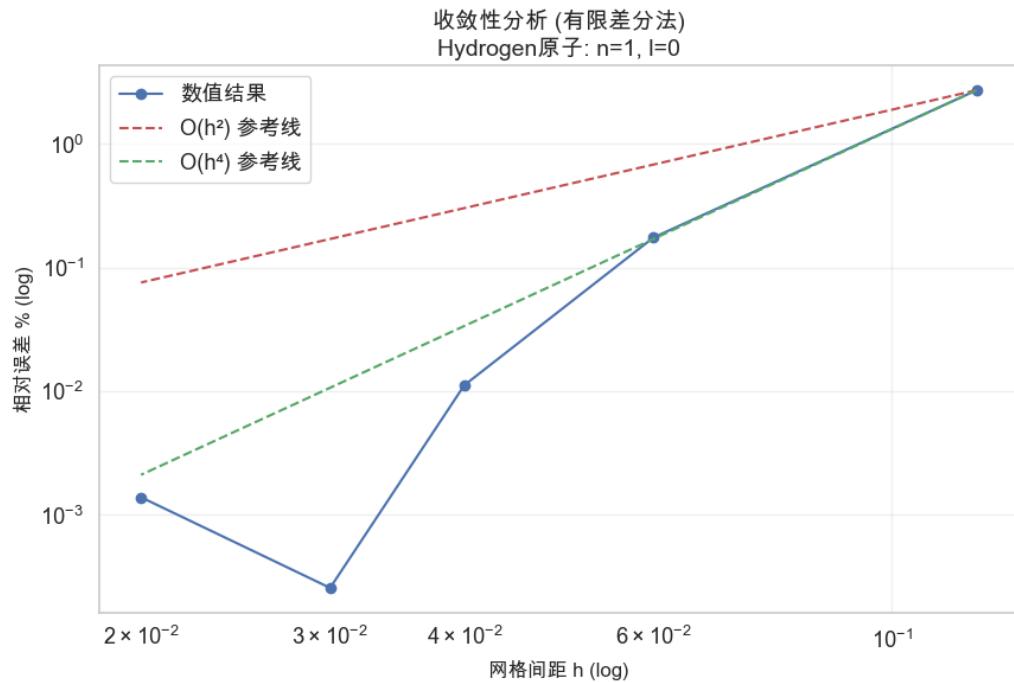


图 14: 使用有限差分法求解氢原子库仑势的 1s 态的收敛性分析, 新版本非均匀细网格的数值稳定性有所下降

2.4.5 氢原子其它示例

```
● (base) gilbert@Gilbert-YoungMacBook radial_schrodinger % python main.py --n 3
```

```
=====  
进行单次求解计算  
=====
```

求解配置：

原子类型：hydrogen

量子数：n=3, l=0

求解方法：shooting

网格点数：1000

网格范围：[1.0e-05, 30.0] Bohr

2024-11-22 09:52:29,590 - INFO - NumExpr defaulting to 8 threads.

2024-11-22 09:52:30,244 - INFO - 初始化hydrogen原子求解器: n=3, l=0, 方法=shooting

2024-11-22 09:52:31.457 python[4095:52152] +[IMKClient subclass]: chose IMKClient_Modern

2024-11-22 09:52:31.457 python[4095:52152] +[IMKInputSession subclass]: chose IMKInputSession_Modern

2024-11-22 09:52:37.332 python[4095:52152] The class 'NSSavePanel' overrides the method identifier.

能量本征值：-0.055424 Hartree

理论值：-0.055556 Hartree

相对误差：0.237385%

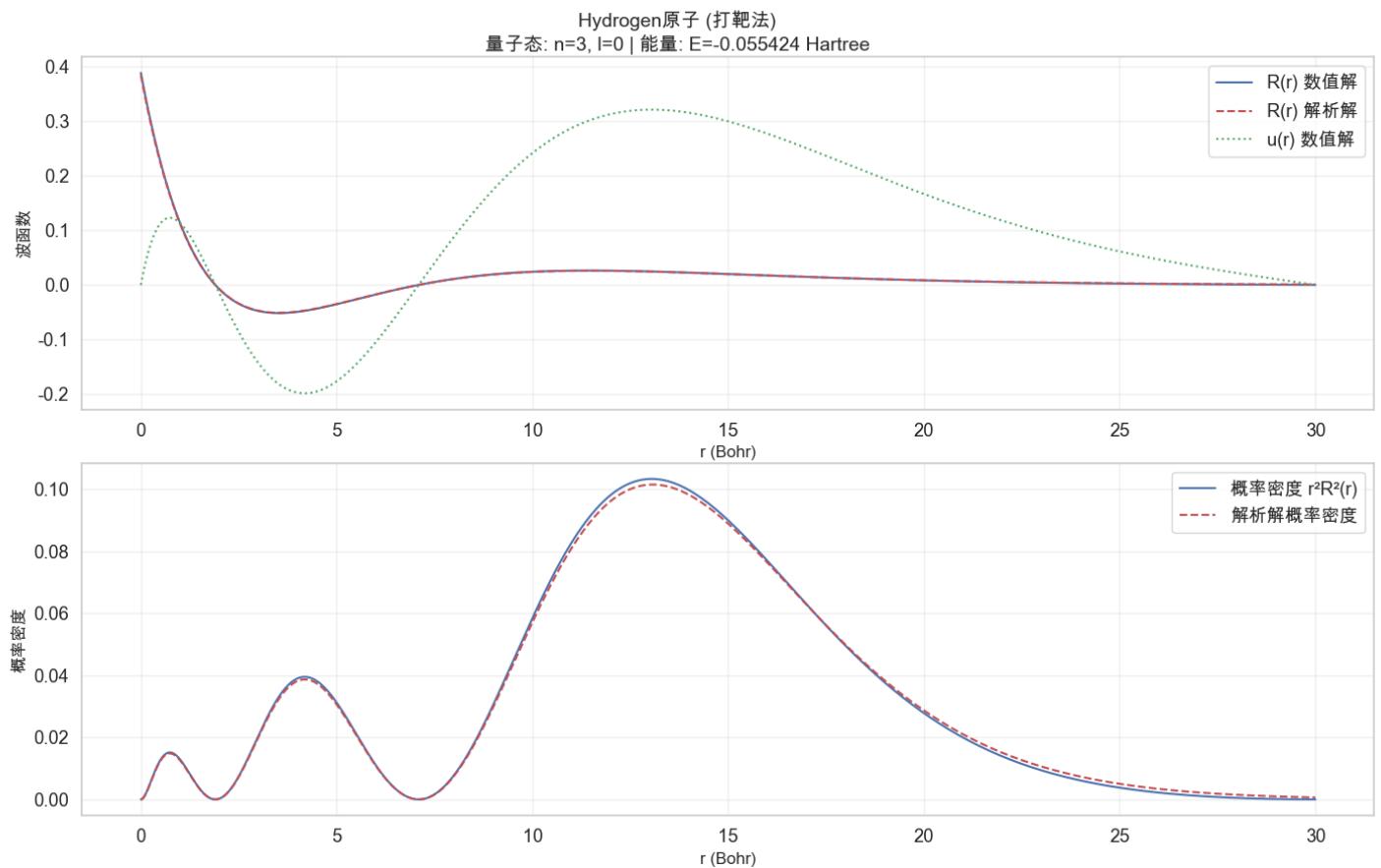


图 15: 使用打靶法求解氢原子库仑势的 3s 态, $r_{\text{Max}}=30$ 的选项已经有些捉襟见肘

```
(base) gilbert@Gilbert-YoungMacBook radial_schrodinger % python main.py --method shooting --j-max 1600 --r-max 60 --V-type hydrogen --n 4 --l 0
=====
进行单次求解计算
=====
求解配置：
    原子类型: hydrogen
    量子数: n=4, l=0
    求解方法: shooting
    网格点数: 1600
    网格范围: [0, 6e-05, 60.0] Bohr
2024-11-22 18:19:59.779 - INFO - NumExpr defaulting to 8 threads.
2024-11-22 18:19:59.711 - INFO - 初始化hydrogen原子求解器: n=4, l=0, 方法=shooting
2024-11-22 18:20:01.110 python[6104:80671] +[IMKClient subclass]: chose IMKClient.Modern
2024-11-22 18:20:01.110 python[6104:80671] +[IMKInputSession subclass]: chose IMKInputSession.Modern
2024-11-22 18:20:16.971 python[6104:80671] The class 'NSSavePanel' overrides the method identifier. This method is implemented by class 'NSWindow'
能级本征值: -0.031248 Hartree
```

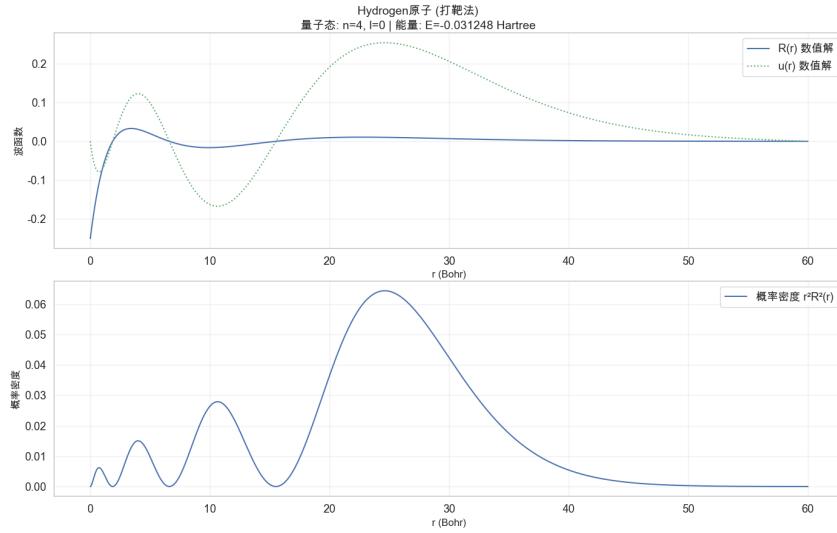


图 16: 使用打靶法求解氢原子库仑势的 4s 态, 指定 $r_Max=60$

```
(base) gilbert@Gilbert-YoungMacBook radial_schrodinger % python main.py --method fd --j-max 400 --r-max 60 --V-type hydrogen --n 4 --l 3
=====
进行单次求解计算
=====
求解配置：
    原子类型: hydrogen
    量子数: n=4, l=3
    求解方法: fd
    网格点数: 400
    网格范围: [0, 6e-05, 60.0] Bohr
2024-11-22 18:21:24.486 - INFO - NumExpr defaulting to 8 threads.
2024-11-22 18:21:24.952 - INFO - 初始化hydrogen原子求解器: n=4, l=3, 方法=fd
2024-11-22 18:23:00.507 python[6223:82431] +[IMKClient subclass]: chose IMKClient.Modern
2024-11-22 18:23:19.487 python[6223:82431] +[IMKInputSession subclass]: chose IMKInputSession.Modern
2024-11-22 18:23:19.487 python[6223:82431] The class 'NSSavePanel' overrides the method identifier. This method is implemented by class 'NSWindow'
能级本征值: -0.031250 Hartree
```

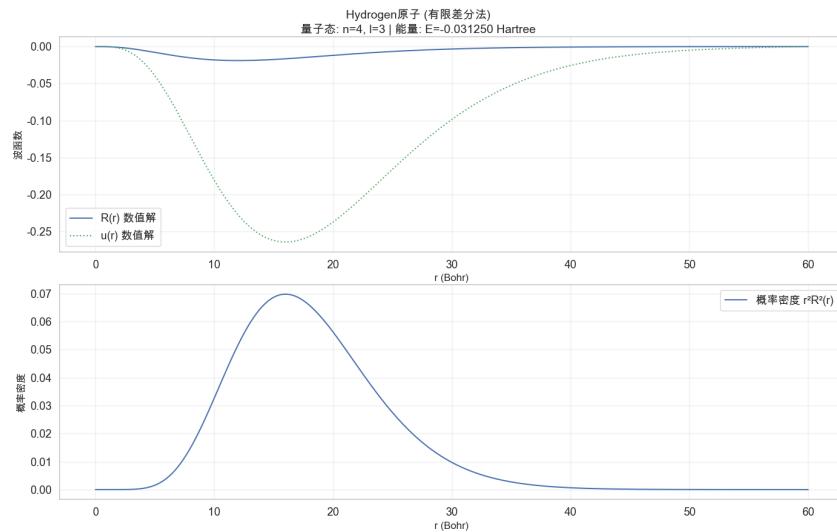


图 17: 使用有限差分法求解氢原子库仑势的 4f 态, 指定 $r_Max=60$

2.4.6 锂原子其它示例

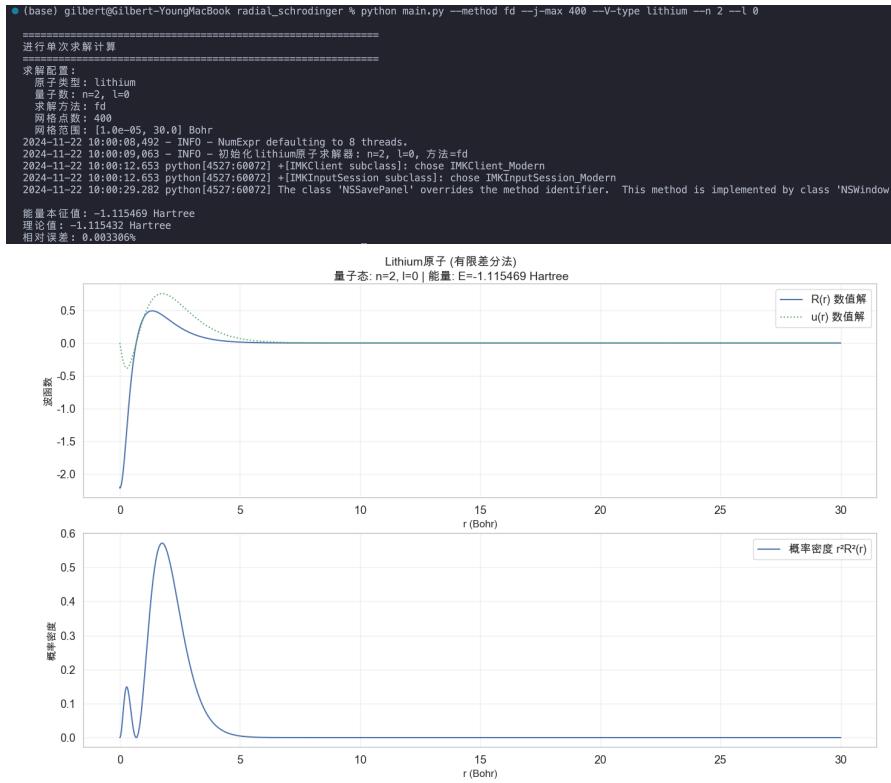


图 18: 使用有限差分法求解锂原子局域势的 2s 态

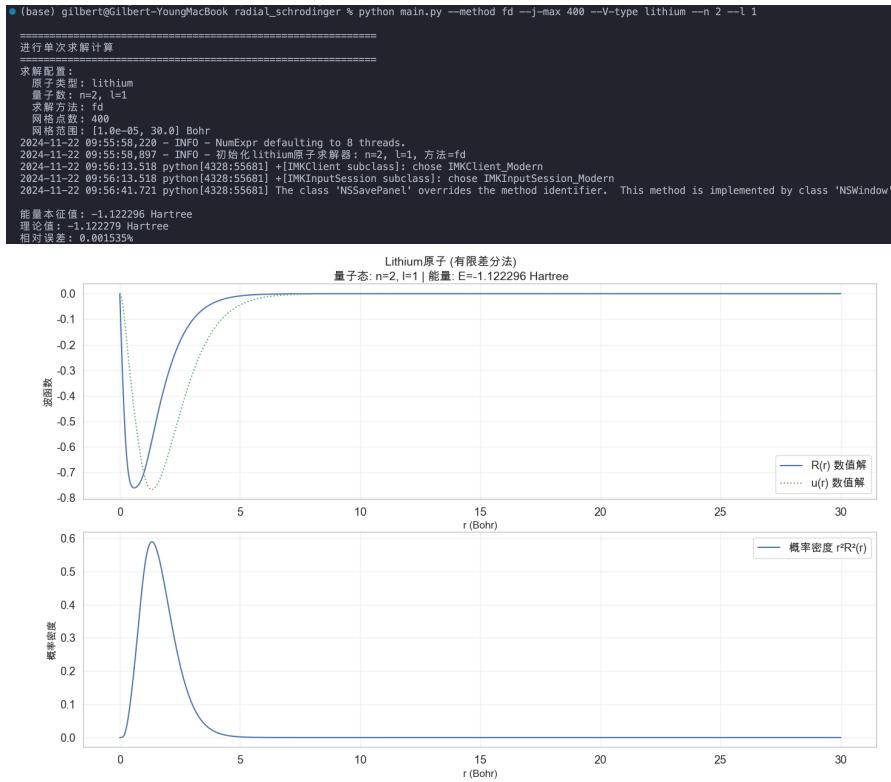


图 19: 使用有限差分法求解锂原子局域势的 2p 态

```

(base) gilbert@Gilbert-YoungMacBook radial_schrodinger % python main.py --n 3 --l 2 --V-type lithium --r-max 60 --method shooting
=====
进行单次求解计算
求解配置:
原子类型: lithium
量子数: n=3, l=2
求解方法: shooting
网格点数: 1000
网格范围: [1.0e-05, 60.0] Bohr
2024-11-22 17:15:57,651 - INFO - NumExpr defaulting to 8 threads.
2024-11-22 17:15:58,193 - INFO - 初始化 lithium 原子求解器: n=3, l=2, 方法=shooting
2024-11-22 17:16:01,244 python[27045:337910]+[IMKClient subclass]: chose IMKClient_Modern
2024-11-22 17:16:01,244 python[27045:337910]+[IMKInputSession subclass]: chose IMKInputSession_Modern
能量本征值: -0.500262 Hartree
=====

求解配置:
量子数: n=3, l=2
求解方法: fd
网格点数: 400
网格范围: [1.0e-05, 30.0] Bohr
2024-11-22 17:15:57,823 - INFO - NumExpr defaulting to 8 threads.
2024-11-22 17:15:58,263 - INFO - 初始化 lithium 原子求解器: n=3, l=2, 方法=fd
2024-11-22 17:16:36,258 python[27047:337926]+[IMKClient subclass]: chose IMKClient_Modern
2024-11-22 17:16:36,258 python[27047:337926]+[IMKInputSession subclass]: chose IMKInputSession_Modern
2024-11-22 17:16:45,457 python[27047:337926] The class 'NSSavePanel' overrides the method identifier. This method is implemented by class 'NSWindow'
能量本征值: -0.500269 Hartree
=====
```

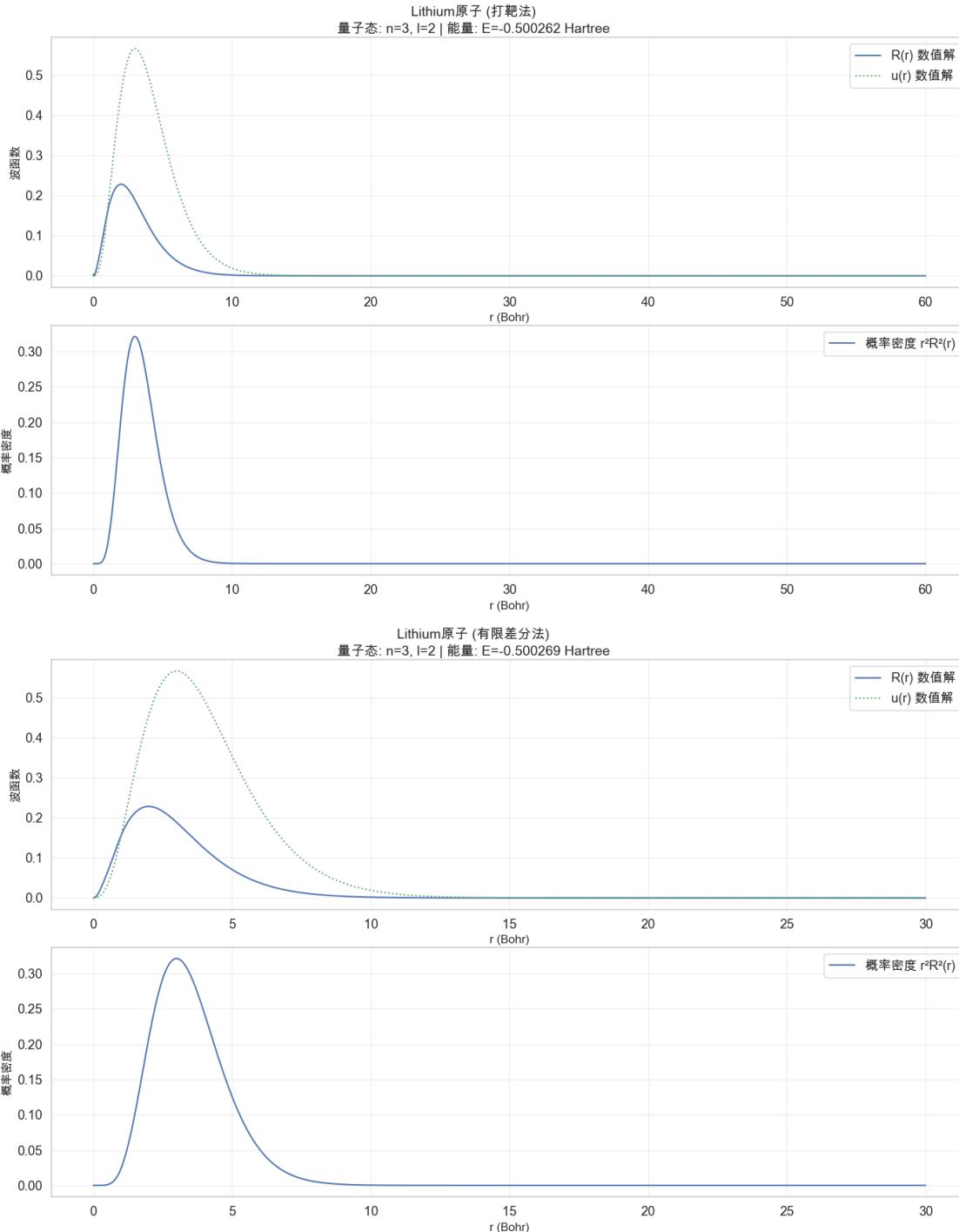


图 20: 分别使用打靶法与有限差分法求解锂原子局域势的 3d 态, 其中打靶法必须拓展 `r_Max`, 否则无法收敛