双原子分子振动能级半量子化理论的数值计算

王治平

2020 级 物理学二班
320200908151

指导老师：吴枝喜、关剑月

2021 年 10 月 28 日

摘要

量子系统的能级问题通常都是通过对于 $Schrdinger$ 方程来求解进行处理。这种常规方法，可以同时给出系统的束缚态能级和相应的本征函数，然而对于一些系统能直接求解较为困难，常采用数值法或各种不同的近似方法. 本文采用了以 $Lennard-Jones$ 势和 $Morse$ 势两种位势函数近似描述双原子分子振动过程的势能，从而求解。文章给出了两种位势函数作为理论支持下数值计算的初步结果，对其分别分析了的误差结果差异与初步原因，并指出了两种函数各自的问题。

课题程序实现已上传至 https://github.com/YingD0/level2-work1

目录

1 课题介绍

1.1 分子振动能级

分子能级指的是分子内部各种运动状态所形成的能级结构, 分子内部各种运动状态所形成的能级结构, 其内部的运动有电子运动、分子振动和分子转动, 它们的能量都是量子化的, 故可形成电子能级、振动能级和转动能级。其中振动能级能级跃迁产生振动光谱, 有一定的宏观表现。

1.2 课题内容

本课题旨在对模式分子[1]的振动能级进行数值计算, 期间用到 $Bohr$ 等人的半量子理论进行分析架构, 故为双原子分子振动能级半量子化理论的数值计算。

[1]本文中模式分子指具有典型性质便于数值计算的简单上双原子分子, 如 H_2, DH, O_2 等

2 理论分析

2.1 原理介绍

2.1.1 *Bohr* 的定态能级假设

1913 年，为了解释氢原子的线状光谱，*Bohr* 作出以下三个假设：

• **定态假设:** 电子仅沿特殊轨道圆周运动, 此时其处于不辐射电磁波的具有确定能量 E_n 的稳定状态, 称作定态。

• **跃迁假设:** 当电子在两个定态 E_m 和 E_n 跃迁时, 才伴有频率为 $\nu = |En-Em|/h$ 的电磁波的吸收或发射。

• **量子化假设:** 运动轨道由量子化条件 $J = \oint pdq = nh$(n 为非负整数) 确定, 其中 J 为电子角动量, p 和 q 分别为其广义动量和广义坐标

本次课题主要利用 *Bohr* 的定态能级假设中的量子化假设作为理论基础，结合基础物理学定律进行数值计算。结合本体具体情况量子化假设可以表示为在相空间中闭合轨道积分满足闭合积分（相空间面积）为 $\hbar/2$ 的奇数倍，即 $\oint pdr = n + \frac{1}{2}\hbar$ 具体理论会在 2.1.3 部分讨论。

2.1.2 原子核势能

考虑一维空间中孤立双原子分子的运动，由于原子核对于核运动的贡献远大于核外电子，故而在本题目的讨论中可以仅讨论原子核的能量。由于其运动过程中所受的作用力的大小以及种类与其核间距（r）有关（近距离主要为电磁作用和量子作用，远距离主要为范德瓦耳斯作用以及电磁作用），故而可以用一个关于核间距的函数表示这种相互之间的位势。通常用来表示上述关系的势函数有 *Lennard-Jones* 位势，*Morse* 位势，以及 *Varshni* 经验势, 其中本文采用前两种势函数进行数值计算与结果比较。

•***Lennard-Jones* 势**

最早由数学家 John Lennard-Jones 于 1924 年提出。由于其解析形式简单形象而被广泛使用，其表达式为

$$V(r) = V_0[(\frac{a}{r})^{12} - (\frac{a}{r})^6] \tag{1}$$

其中 r 为两原子核之间的距离，V_0 为势井深度，a 为波尔半径。绘制出的无量纲示意图见 图 *1.*

•***Morse* 势**

以物理学家 *Hilip M. Morse* 的名字命名的 Morse 势是一种对于双原子分子间势能的简易解析模型。一方面，对 *Morse* 势求解薛定谔方程具有解析解，方便分析问题；另

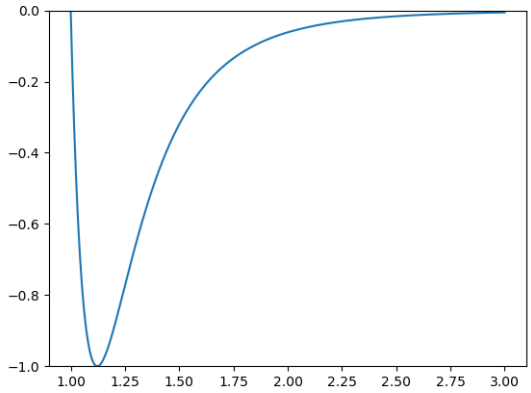


图 1: $Lennard-Jones$ 势-核间距无量纲化图像

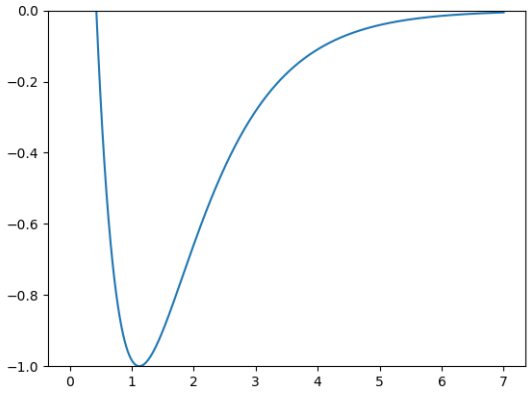


图 2: $Morse$ 势-核间距无量纲化图像

一方面，由于它隐含地包括了键断裂这种现象，对于分子振动的微细结构的具有良好的近似。Morse 势包含有谐振子模型所缺乏的特性，那就是非成键态。相对量子谐振子模型，$Morse$ 势更加真实，因为它能够描述非谐效应，倍频，以及组合频率。

$Morse$ 势具有如下的形式

$$V(r) = -D_e + D_e(1 - e^{-a(r-r_e)})^2 \tag{2}$$

在这里，r 是核间距（或键长）；r_e 是平衡键长（$\frac{dV(r)}{dr}|_{r=r_e} = 0$, 即最低点对应 r 值）；$D_e$ 是 Morse 势的阱深（势能零点可任意选取，在此将解离极限设为势能零点，即两核间距趋于无穷远时令系势能为零），$V(\infty) = 0$）；a 则控制了势井的“宽度”，然而在本题中只考虑纵向能级，故可暂且不与讨论。对于 $Morse$ 势绘制出的无量纲示意图见 图 *2.*

2.2 理论生成

基于上述两个前置介绍，我们可以得到分子振动的势能以及总能量，仅仅讨论振动能级时我们可以对于能级计算进行简化，并得出简化模型的计算方法。

2.2.1 简化模型

- **只考虑原子核的运动**

同 2.1.1 介绍，在本题目尺度要求下，可以省却电子运动的影响故而将势能简化为原子核势能，便于后续的分析与讨论。

- **只考虑原子核的振动**

原子核占有原子绝大多数的质量，同时原子核的转动比振动要慢的多，故而对于振动能级的贡献可以忽略，故而可以作出只考虑原子核的振动。

- **振动方程的讨论**

原子核尺度的粒子运动可以用量子力学理论进行解释，设第 n 能级的振动能量为 E_n，波函数为 $\Psi_n(r,t)$, 简化后的核的振动可以用一维定态薛定谔方程描述

$$[-\frac{\hbar^2}{2m}\frac{d^2}{dr^2}+V(r)]\Psi_n=E_n\Psi_n \tag{3}$$

按照量子理论只需要解出 $[-\frac{\hbar^2}{2m}\frac{d^2}{dr^2}+V(r)]$ 在波函数空间中的本征值，这样就得出了各能级以及对应能量。本题我们把问题看作原子核在 *V(r)* 势场的经典运动结合 *Bohr* 的定态能级假设来求解量子化的振动能级。

2.2.2 理论生成

有了上述的简化模型，我们可以进行一些简单的理论分析。

从能量角度，系统具有的总能量为

$$E=\frac{p^2}{2m}-V(r) \tag{4}$$

p 为原子核相对运动的动量, $m=\frac{m_0^2}{m_0+m_0}=\frac{1}{2}m_0$ 为约化质量。显然前一项为动能项，后一项为势能项。

根据（4）式 p 可以写作如下形式。

$$p(r)=\pm\sqrt{[2m(E-V(r))]} \tag{5}$$

显然的，在确定能级下（给定 E）这是一条在相空间的闭合轨道。由 *Bohr* 的定态能级假设的限制我们可以得到

$$\oint p(r)dr=(2n+1)\pi\hbar \tag{6}$$

将（5）式代入，同时用 E_n 表示第 n 级轨道对应的能量，考虑到 p 的方向性每个确定态具有两个相反的取值，得到如下方程

$$\int_{r_{in}}^{r_{out}}\sqrt{2m(E_n-V(r))}dr=(n+\frac{1}{2})\pi\hbar \tag{7}$$

其中 r_{in},r_{out} 表示相空间左右两个端点，根据相空间物理意义可知其为运动的两个极限点，此刻动能全部转化为势能，即 $E_n-V(r)=0$。

为了减少计算工作量，防止数据溢出，同时也我们对上述提到的物理量进行无量纲

化:

$$\varepsilon_n = \frac{E_n}{V_0},\quad x = \frac{a}{c},\quad \gamma = \sqrt{\frac{2ma^2V_0}{\hbar^2}}$$

(7) 式可以改写为

$$\gamma \int_{x_{in}}^{x_{out}} \sqrt{\varepsilon_n - v(x)}\, dx = (n + \frac{1}{2})\pi \tag{8}$$

其中

$$v(x) = 4(x^{-12} - x^{-6}) \quad (Lennard{-}Jones \text{ 势})$$

或

$$v(x) = -1 + (1 - e^{-a(x-x_e)})^2 \qquad (Morse \text{ 势})$$

在最终得到的（8）式中，γ 是唯一一个与分子本身有关的量。即进行无量纲化操作之后只需要改变 γ 的数值就可以切换分子种类，达到了上述的便于计算。γ 中所包含的 a 和 V_0 可以通过对于分子转动能量和电离能的测定实验得出。

x_{in} 和 x_{out} 的给出和无量纲化之前的操作类似方程

$$\varepsilon_n - v(x) = 0 \tag{9}$$

的零点即是给定 ε_n 对应的 x_{in} 和 x_{out}.

上述操作等价于寻找函数

$$f(n, \varepsilon_n, x_{in}, x_{out}) = \gamma \int_{x_{in}}^{x_{out}} \sqrt{\varepsilon_n - v(x)}\, dx - (n + \frac{1}{2})\pi \tag{10}$$

的零点，寻找本函数零点的方法会在数值计算部分具体讨论。

3 数值计算

理论分析过程已经得出了若干关于 $n, \varepsilon_n, x_{in}, x_{out}$ 四个无量纲量的方程，本部分将讨论寻找零点的方法。

3.1 计算思路

只有式 (9)(10) 是关于 $n, \varepsilon_n, x_{in}, x_{out}$ 四个无量纲量的方程或函数，为寻找式 (10) 零点，可以建立联立上式形成如下方程组

$$\begin{cases} \gamma \int_{x_{in}}^{x_{out}} \sqrt{\varepsilon_n - v(x)}\, dx - (n + \frac{1}{2})\pi = 0 \\ \varepsilon_n - v(x_{in}) = 0 \\ \varepsilon_n - v(x_{out}) = 0 \end{cases}$$

同时根据量子化约束还有 n 为非负整数的约束条件。

由于连立结果是未知数个数大于方程个数的不定方程，我们只能通过量子化约束结合计算机循环给定 ε_n 的数值进行不定方程的数值求解。

问题的关键转化为在给定 ε_n 的条件下求出 x_{in}, x_{out}，以及 $\int_{x_{in}}^{x_{out}} \sqrt{\varepsilon_n - v(x)}\, dx$.

3.2 计算实现

3.2.1 *Newton* 迭代法解方程

首先，选择一个接近函数 $f(x)$ 零点的 x_0，计算相应的 $f(x_0)$ 和切线斜率 $f'(x_0)$（这里 f' 表示函数 f 的导数）。然后我们计算穿过点 $(x_0, f(x_0))$ 并且斜率为 $f'(x_0)$ 的直线和 x 轴的交点的 x 坐标，也就是求如下方程的解：

$$0 = (x - x_0) \cdot f'(x_0) + f(x_0)$$

我们将新求得的点的 x 坐标命名为 x_1，通常 x_1 会比 x_0 更接近方程 $f(x) = 0$ 的解。因此我们现在可以利用 x_1 开始下一轮迭代。迭代公式可化简为如下所示：

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

已有证明牛顿迭代法的二次收敛必须满足以下条件：$f'(x) \neq 0$; 对于所有 $x \in I$，其中 I 为区间 $[\alpha - r, \alpha + r]$，且 x_0 在区间其中 I 内，即 $r \geqslant |a - x_0|$ 的; 对于所有 $x \in I$, $f''(x)$ 是连续的; x_0 足够接近根 。

显然的，在合适的构造下式 (9) 可以满足收敛条件，下面分别计算两种位势函数满足 $\varepsilon_n - v(x) = 0$ 的解。

· 方程 (9) 在 $Lennard-Jones$ 势的计算

$Lennard-Jones$ 势无量纲化的形式如下

$$v(x)=4(x^{-12}-x^{-6})$$

方程 (9) 符合题目要求的解即方程

$$\varepsilon_n=v(x)=4(x^{-12}-x^{-6})$$

的实解, 对于本方程不妨换元降次为

$$\varepsilon_n=4(y^2-y)$$
$$y=x^{-6}$$

可解得

$$y=\frac{1\pm\sqrt{1+\varepsilon_n}}{2}$$

显然，两根均为实根，故而用 $Newton$ 迭代法求 x 即可, 迭代公式如下

$$f(x)=x^{-6}-y=0$$
$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}=x_n-\frac{x_n^{-6}-y}{-6x_n^{-7}}$$

经验证循环 1000 次后，精度已经达到 10^{-12} 量级，完全符合要求，可不进行判断。程序实现如下

运行环境

$Ubuntu9.3.0-17\ Ubuntu1\ 20.04LTS$

$Python3.8.8\quad Anaconda4.10.3\quad AMD-4800u$

```
import numpy as np
```

```
solve=[]
```

```
#Lennard-Jones
def inout(epsilon):
    epsilon=epsilon/4
    y2=(1-np.math.sqrt(1+4*epsilon))/2
    y1=(1+np.math.sqrt(1+4*epsilon))/2
    x1,x2=1.1,1.3
    for i in range(1,1000):
        x1=x1-(x1**-6-y1)/(-6*x1**-7)
        x2=x2-(x2**-6-y2)/(-6*x2**-7)
```

```
    solve.append(x1)
    solve.append(x2)
    return solve[0],solve[1]
```

· 方程 (9) 在 $Morse$ 势的计算

$Morse$ 势无量纲化的形式如下

$$v(x) = -1 + (1 - e^{-a(x-x_e)})$$

方程 (9) 符合题目要求的解即方程

$$\varepsilon_n = v(x) = -1 + (1 - e^{-a(x-x_e)})$$

可解得

$$x = x_e - \frac{\ln(1 \pm \sqrt{\varepsilon_n + 1})}{a}$$
$$x_e = \sqrt[6]{2}$$

本题中可取 $a = 1$, 程序实现如下

运行环境

$Ubuntu9.3.0 - 17\ Ubuntu1\ 20.04LTS$

$Python3.8.8 \quad Anaconda4.10.3 \quad AMD - 4800u$

```
import numpy as np
```

```
solve=[]
```

```
#Morse
def inout(epslion):
    e=epslion
    x1=2**(1/6)-(np.math.log(1-np.math.sqrt(e+1)))/kk
    x2=2**(1/6)-(np.math.log(1+np.math.sqrt(e+1)))/kk
    if (x1>x2):
        x1,x2=x2,x1
    solve.append(x1)
    solve.append(x2)
    return solve[0],solve[1]
```

3.2.2 *Romberg* 积分

•*Richardson* 外推方法

假设某个所求量（例如某个定积分的值）可以写成积分步长 h 的函数如下

$$A = A(h) + Ch^k + C'h^{k+1} + C''h^{k+2}\cdots$$

其中 k 为刻画误差阶数的未知常数，$C', C'', C'''\cdots$ 为未知系数。例如，梯形积分中 $k = 2$，其精确解可以写作

$$A = A(h) + Ch^2 + C'h^3 + O(h^3)$$

如果忽略高阶小量可以得到

$$A = A(h) + Ch^k$$

$A(h)$ 可以通过数值积分得到，如果我们将步长减半我们可以得到

$$A = A(\tfrac{h}{2}) + C(\tfrac{h}{2})^k + O(h^{k+1})$$

联立,

$$A = A(h) + Ch^k + C'h^{k+1} + O(h^{k+1})$$

可以得到:

$$A = \tfrac{2^k A(h/2) - A(h)}{2^k - 1} + O(h^{k+1})$$

令,

$$B(h) = \tfrac{2^k A(h/2) - A(h)}{2^k - 1}$$

有:

$$A = B(h) + O(h^{k+1})$$

对比两式

$$A = A(h) + Ch^k + C'h^{k+1} + O(h^{k+1})$$
$$A = B(h) + O(h^{k+1})$$

可见 $B(h)$ 的精度比 $A(h)$ 高出一阶，这种从低阶精度格式的截断误差的渐进展开出发,通过改变积分步长构造简单线性组合,从而得到高阶精度格式的方法称为 *Richardson* 外推加速收敛技术

•*Romberg* 求积方法

将 *Richardson* 外推方法应用于复合梯形求积公式，得到 *Romberg* 求积方法

梯形积分的误差形式如下

$$I - T_n = Ch^2 + C'h^4 + C''h^5 \cdots$$
$$I - T_n = O(h^2)$$

步长减半时

$$I - T_{2n} = O(\tfrac{h^2}{4})$$

可以得到

$$\tfrac{1-T_{2n}}{1-T_n} \approx \tfrac{1}{2^2}$$
$$1 - T_{2n} \approx \tfrac{1}{3}(T_{2n} - T_n)$$

即将积分区间数量翻倍后，定积分值近似为

$$I = \tfrac{2^2 T_{2n} - T_n}{2^2 - 1} + O(h^4)$$

可见精度极大提高，对于梯形积分公式进行本操作可以得到 $Simpson$ 积分公式。

同理，继续对于 T_{2n} 使用 $Richardson$ 外推, 可以得到更精确的结果，以此类推，在增加外推次数的过程中，我们还可以得到 $Newton - Cotes$ 公式等更加精确的积分公式，不断增加外推数目得到符合精度要求的结果即可。

将学过的积分公式可以做出如下外推

$$T_n = h_n \cdot \left[\tfrac{1}{2}f(a) + \sum_{i=1}^{n-1} f(x_i) + \tfrac{1}{2}f(b)\right]$$

$$S_n = T_{2n} + \tfrac{1}{2^2-1}\left(T_{2n} - T_n\right) + O\left(h^4\right)$$

$$C_n = S_{2n} + \tfrac{1}{2^4-1}\left(S_{2n} - S_n\right) + O\left(h^6\right)$$

$$R_n = C_{2n} + \tfrac{1}{2^6-1}\left(C_{2n} - C_n\right) + O\left(h^8\right)$$

对于任意外推次数的积分公式均可用如下公式多次迭代得到满足精度的结果

$$R_{k,j} = R_{k,j-1} + \frac{R_{k,j-1} - R_{k-1,j-1}}{4^{j-1} - 1}, k = 2, 3$$

$$\begin{array}{ccccccc}
R_{1,1} & & & & & & \\
 & \searrow & & & & & \\
R_{2,1} & & \longrightarrow R_{2,2} & & & & \\
 & \searrow & & \searrow & & & \\
R_{3,1} & \longrightarrow & R_{3,2} & \longrightarrow & R_{3,3} & & \\
\vdots & & \vdots & & \vdots & \ddots & \\
R_{n,1} & \longrightarrow & R_{n,2} & \longrightarrow & R_{n,3} & \ldots & R_{n,n}
\end{array}$$

程序实现如下

运行环境

$Ubuntu9.3.0-17\ Ubuntu1\ 20.04LTS$

$Python3.8.8\quad Anaconda4.10.3\quad AMD-4800u$

```
import numpy as np
```

```
def fun(x):
    f = np.math.sqrt(round((epsilon)-(vx(x)),10))
    return f
```

```
def T_2n(a, b, n, T_n):
    if n<1:
        print('n should larger than 1')
    h = (b - a)/n
    sum_f = 0.
    for k in range(0, n):
        sum_f = sum_f + fun(a + (k + 0.5)*h)
    T_2n = T_n/2. + sum_f*h/2.
    return T_2n
```

```
def Romberg(a, b, err_min):
    kmax = 10
    tm = np.zeros(kmax,dtype = float)
    tm1 = np.zeros(kmax,dtype = float)
    tm[0] = 0.5*(b-a)*(fun(a) + fun(b))
    #print(tm)
    err = 1.
    k = 0
    np.set_printoptions(precision = 9)
    while(err>err_min and k <kmax-2):
        n = 2**k
        m = 1
        tm1[0] = T_2n(a, b, n, tm[0])
        while(err>err_min and m <= (k+1)):
            tm1[m] = tm1[m-1]+(tm1[m-1]-(tm[m-1]))/(4.**m-1)
            result = tm1[m]
            err1 = abs(tm1[m]-tm[m-1])
            err2 = abs(tm1[m]-tm1[m-1])
            err = min(err1,err2)
            m = m+1
```

```
        tm = np.copy(tm1)
        k = k+1
        #print(tm)
    return result
```

3.2.3 猜解枚举法解不定方程

由 3.1，我们可以得到下述关于 n 的不定方程

$$\gamma \int_{x_{in}}^{x_{out}} \sqrt{\varepsilon_n - v(x)}\, dx - (n + \tfrac{1}{2})\pi = 0$$

建立关于 n, 与 ε_n 的双重循环进行梯度穷举，将达到 $f(n)$ 符合精度要求的零点的 ε_n 值进行存储并输出，即可得到一个关于 ε_n 的集合，即为无量纲化的能级集合。需要注意的是由于 $\varepsilon_n \to 0$ 时各个能级差距过小，对于 ε_n 的梯度选择应该有倾向性的非均匀。方便起见本次计算直接写入主函数, 程序实现如下

运行环境

$Ubuntu9.3.0-17\ Ubuntu1\ 20.04LTS$

$Python3.8.8\quad Anaconda4.10.3\quad AMD-4800u$

```
import numpy as np
```

```
if __name__=="__main__":
    epsl=[]
    fnnn=[]
    result=[]
    for n in range(0,N):
        epsl=[]
        fnnn=[]
        while (1):

            solve=[]

            inout(epsilon)
            if (abs(f(n))<1.e-2):
                print(f(n),n,epsilon,solve)
                fnnn.append(f(n))
                epsl.append(epsilon)
            epsilon=epsilon+-epsilon/1000
            if epsilon>-0.0001:
                break
        print(fnnn)
```

```
        print(n,epsl)
        if (len(epsl)==0):
            print("MAX=",n-1)
            break
        mi=fnnn[0]
        for i in range(0,len(fnnn)):
            if abs(mi)>abs(fnnn[i]):
                mi=fnnn[i]
        for i in range(0,len(epsl)):
            if mi==fnnn[i]:
                result.append(epsl[i])
        print("r",result)
        epsilon=result[-1]
    print(result)
    #势 能 图
```

3.3 计算结果

根据两种势能计算出的氢分子振动能如下 (在这里只展示氢分子振动能级其他计算结果见附录):

表 1: H_2 $Lennard-Jones$ 势 能级能量列表

能级	0	1	2	3	411111
无量纲能量	-0.77246	-0.42254	-0.19557	-0.06772	-0.01251

表 2: H_2 $Morse$ 势 能级能量列表

能级	0	1	2	3	4
无量纲能量	-0.95402	-0.86579	-0.78179	-0.70172	-0.62608
能级	5	6	7	8	9
无量纲能量	-0.55525	-0.48802	-0.42551	-0.36694	-0.31298
能级	10	11	12	13	14
无量纲能量	-0.26297	-0.21745	-0.17624	-0.13931	-0.10665
能级	15	16	17	18	19
无量纲能量	-0.07829	-0.05423	-0.03450	-0.01910	-0.00810

为了直观表述, 可以把能级在位势图中标注, 同时对齐 $p-x$ 相图, 对于两种位势能函数, 可以分别得到以下两张图:

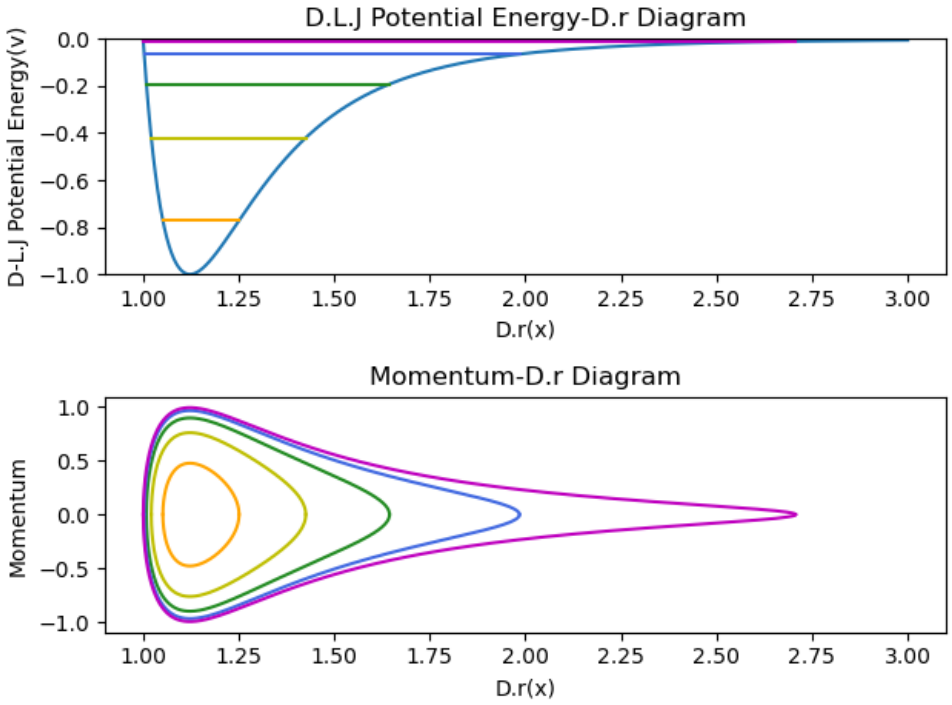


图 3: $Lennard-Jones$ 势的能级图以及相图

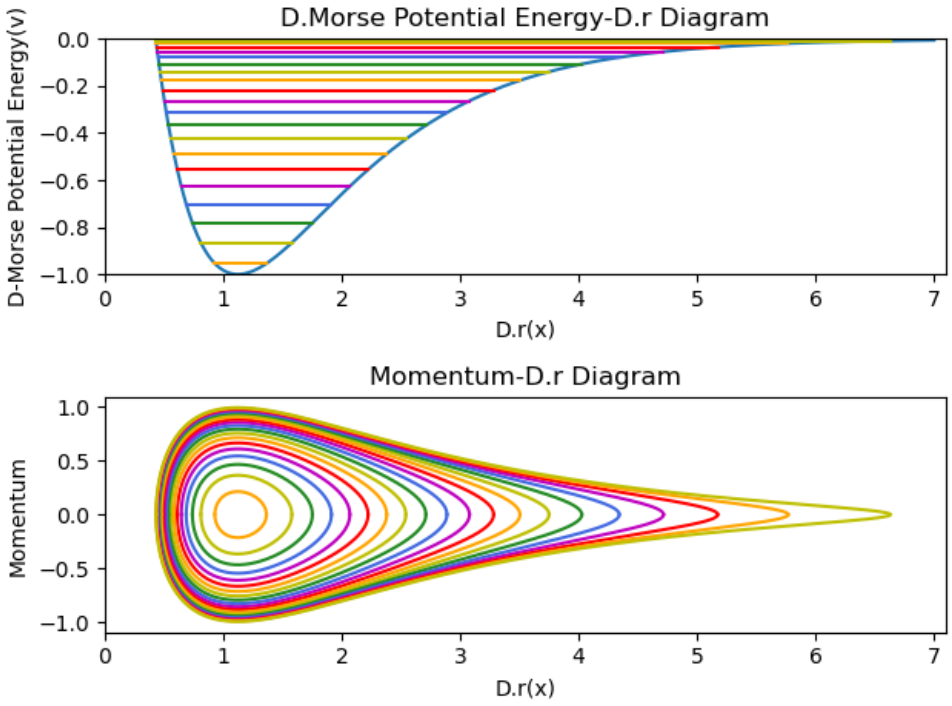


图 4: $Morse$ 势的能级图以及相图

其中, 上半部分是势能图, 下半部分为 $p-x$ 相图, 可以观察到相图和势能图对应能

级的 x_in, x_out 对应相等.

程序实现见下 (篇幅所限, 程序展示只有最终结果, 完整内容见附录)

运行环境

$Ubuntu9.3.0-17\ Ubuntu1\ 20.04LTS$

$Python3.8.8\quad Anaconda4.10.3\quad AMD-4800u$

```
import numpy as np
import matplotlib.pyplot as plt
epsilon=-0.99
N=100
gamma=21.7
solve=[]
kk=1
col=['orange','y','forestgreen','royalblue','m','r']
ee=np.math.e
```

```
def paint(ii):

    x1=inout(result[ii])
    n1=[vx(x1[0]),vx(x1[1])]
    plt.plot(x1,n1,color=col[i%6])
    solve.clear()
```

```
def paint2(i):
    e=result[i]/4
    #Lennard -Jones 势
    y2=(1-np.math.sqrt(1+4*e))/2
    y1=(1+np.math.sqrt(1+4*e))/2
    x1,x2=1.1,1.1
    for o in range(1,1000):
        x1=x1-(x1**-6-y1)/(-6*x1**-7)
        x2=x2-(x2**-6-y2)/(-6*x2**-7)
    xx=np.linspace(x1,x2,10000)
    #p=np.sqrt(np.abs(result[i]-vx(xx)))
    p=(np.abs(result[i]-vx(xx))**0.5)
    plt.plot(xx,p,color=col[i%6])
    plt.plot(xx,-p,color=col[i%6])
```

```
def paint2(i):
    #Morse 势
    e=result[i]
    x1=2**(1/6)-(np.math.log(1-np.math.sqrt(e+1)))/kk
```

```
    x2=2**(1/6)-(np.math.log(1+np.math.sqrt(e+1)))/kk
    xx=np.linspace(x1,x2,1000000)
    #p=(result[i]-(np.round(vx(xx))**0.5,4))
    p=(np.abs(result[i]-vx(xx))**0.5)
    plt.plot(xx,-p,color=col[i%6])
    plt.plot(xx,p,color=col[i%6])
```

```
if __name__=="__main__":
```

```
    plt.xlabel("D.r(x)")
    plt.xlim(0.9,3.1)
    plt.ylim(-1,0)
    # 将第二个 subplot 激活，并绘制第二个图像
    plt.subplot(2,  1,  2)
    for i in range(0,len(result)):
        paint2(i)
    plt.title("Momentum-D.r Diagram")
    plt.ylabel("Momentum")
    plt.xlabel("D.r(x)")
    plt.xlim(0.9,3.1)
    # 展示图像
    plt.show()
```

3.4 结果比较

查找资料可得, 氢分子振动能级势井深度为 4.6eV, 能级实验值如下表:

表 3: 氢分子振动能级实验值

能级	0	1	2	3	4
能量 (eV)	-4.477	-3.962	-3.475	-3.017	-2.587
能级	5	6	7	8	9
能量 (eV)	-2.185	-1.811	-1.466	-1.151	-0.867
能级	10	11	12	13	14
能量 (eV)	-0.615	-0.400	-0.225	-0.094	-0.017

根据势井深度还原无量量纲能量进行误差处理结果如下:

表 4: $Lennard-Jones$ 势及 $Morse$ 势误差表

$Lennard-Jones$ 势				能级实验值	$Morse$ 势			
能级	无量纲	理论值	误差		误差	理论值	无量纲	能级
0				-4.477	1.97%	-4.411	-0.954	0
1				-3.962	0.52%	-4.003	-0.866	1
2	-0.772	-3.553	2.25%	-3.475	3.49%	-3.614	-0.782	2
3				-3.017	6.99%	-3.244	-0.702	3
						-2.894	-0.626	
4				-2.587	1.27%	-2.567	-0.555	4
5				-2.185	2.74%	-2.256	-0.488	5
6	-0.423	-1.944	7.33%	-1.811	8.08%	-1.967	-0.426	6
						-1.696	-0.367	
7				-1.466	1.79%	-1.447	-0.313	7
8				-1.151	5.10%	-1.216	-0.263	8
						-1.005	-0.217	
9	-0.196	-0.900	3.76%	-0.867	6.49%	-0.815	-0.176	9
10				-0.615	4.20%	-0.644	-0.139	10
						-0.493	-0.107	
11				-0.400	9.97%	-0.362	-0.078	11
						-0.251	-0.054	
12	-0.068	-0.312	38.45%	-0.225	29.46%	-0.160	-0.035	12
13				-0.094	6.63%	-0.088	-0.019	13
14	-0.013	-0.058	238.48%	-0.017	119.05%	-0.037	-0.008	14

从表 *4*, 我们可以看出, 对于氢分子, 依据 $Lennard-Jones$ 势和 $Morse$ 势作为理论基础计算出来的能级大都可以在实验数值中找到对应，但是在本课题的运行环境以及程序实现中,$Lennard-Jones$ 势出现了明显的缺级, 反而 $Morse$ 势出现了明显的多级; 同时, 两种位势函数在较高能级处均出现了较大误差, 目前遇到的问题都和文献中出现或者提到的问题相同，并且已经相对文献减少了存在的问题与误差（如缺级情况有所改善）。目前推测问题原因可能是能级差过小同时计算精度不足等，具体细节会在后续讨论。

4 问题分析

在进行双原子分子振动能级半量子化理论的数值计算时，我们主要遇到了如下问题

· $Lennard-Jones$ 势的缺级

· $Morse$ 势的多级

· 计算时间过长

· 在较高能级存在较大误差

· 求根迭代方法

· 最优筛选问题

下面对各个问题的进行讨论。

4.1 问题产生

•$Lennard-Jones$ 势的缺级

$Lennard-Jones$ 势的缺级是比较全面的, 如表 *4* 所示其在不同能级均有缺级, 根据引文 [1][2][3] , 可以知道在常见计算精度之下, 由于 $Lennard-Jones$ 势不够平滑, 缺级常有发生。在计算精度不足和 $Lennard-Jones$ 势本身缺陷下造成了能级的缺失。

•$Morse$ 势的缺级

$Lennard-Jones$ 势的多级，主要在高能级出现频繁，原因大概同样是精度的缺陷，但是仅限于猜测，具体原因为查阅到相关文献，准备日后进行理论分析。

• 计算时间过长

由于多次迭代，判断，循环，调用函数，计算量较大，因此计算时间过程，我们可以同过对上述内容的优化以及算法改进减少计算量，从而减少计算时间。

• 在较高能级存在较大误差

较高能级由于能级差异过小，从而容易由于收敛条件产上较大误差，同时由于本身数目极小，误差容易被充分展现，因此应该通过迭代步长和更佳算法的选取减少误差。

• 求根迭代方法

由于各种求解方程的迭代法存在收敛条件，因而我们需要进行方程构造以及算法选取，$Newton$ 迭代法已经是一种收敛条件较弱的求根方法，但是仍然对构造出的函数有所限制，因为需要合理构造函数实现收敛求根。

• 最优筛选问题

对于给定的 n，可能有不止一个 ε_n 满足精度要求，在循环中我们需要考虑如何筛选出最优解并进行储存与输出。

4.2 解决方案

•$Lennard-Jones$ **势的缺级**

由于问题产生原因是势函数的缺陷以及精度不足，因此我们在计算中应该提升精度，或者改变为更准确优秀的位势函数，如本题中用 $Morse$ 势代替 $Lennard-Jones$ 势便可解决缺级问题。

•$Morse$ **势的缺级**

同样的，多级问题也可以通过改变计算精度，取更好的 a 值，以及改进位势函数得到，查阅文献 [1]，了解到 $Varshni$ 或许可以更好的描述本题目描述运动情况的势能，如有需要可以尝试用此修正。

• **计算时间过长**

减少迭代循环次数，优化算法等都可以加快计算减少循环次数可以通过构造合理函数，合理优化算法减少 ε_n 增加次数（不重复计算不可能的数值）等方法，优化算法可以通过改进计算方法，调用更好的函数表达形式，用更稳定的数学包（例如用 $numpy.math$ 而不是 $math$），画图取合适的上下限等等。

• **在较高能级存在较大误差**

较高能级由于能级差异过小，因此我们需要大量提升精度，常规的均匀梯度精度明显不足，因此需要动态梯度在 $\varepsilon_0 \to 0$ 时减少步长以此增加精度。

• **求根迭代方法**

选用收敛条件较弱的牛顿迭代法，构造合适的迭代函数，（用低幂次的迭代函数为好）可以极大的提升收敛速度，如有需要，可以采用 $Aitken$ 加速法再行加速。

• **最优筛选问题**

对于多个 ε_n 满足精度的情况，可以用一个列表进行收纳，本次循环结束之后，再行寻找最优数值并输出到结果列表，便可得到对于每个 n, 最合适的 ε_n 值。

参考文献

[1] 熊志斌. H_2 分子振动能级的数值模拟.*pdf*[M]. 湖南师范大学数学与计算机科学学院，湖南长沙，湖南文理学院学报 (自然科学版) 第 17 卷第三期 2005 年 9 月

[2] 张磬兰; 王燕昌. 双原子分子振动能级半量子化及量子理论的数值计算 [M]. 宁夏大学物理系; 宁夏大学物理系 来源: 大学物理 ISSN:1000-0712 年:1997 卷:016 期:001

[3] 彭芳麟. 计算物理基础 [M]. 高等教育出版社

A 全部程序实现

A.1 *Lennard-Jones* 势理论程序实现

运行环境

$Ubuntu9.3.0-17\ Ubuntu1\ 20.04LTS$

$Python3.8.8\quad Anaconda4.10.3\quad AMD-4800u$

```
import numpy as np
import matplotlib.pyplot as plt
epsilon=-0.99
N=100
gamma=21.7
solve=[]
kk=1
col=['orange','y','forestgreen','royalblue','m','r']
ee=np.math.e
def fun(x):
    f = np.math.sqrt(round((epsilon)-(vx(x)),10))
    return f

def T_2n(a, b, n, T_n):
    if n<1:
        print('n should larger than 1')
    h = (b - a)/n
    sum_f = 0.
    for k in range(0, n):
        sum_f = sum_f + fun(a + (k + 0.5)*h)
    T_2n = T_n/2. + sum_f*h/2.
    return T_2n

#Lennard-Jones
def inout(epsilon):
    epsilon=epsilon/4
    y2=(1-np.math.sqrt(1+4*epsilon))/2
    y1=(1+np.math.sqrt(1+4*epsilon))/2
    x1,x2=1.1,1.3
    for i in range(1,1000):
        x1=x1-(x1**-6-y1)/(-6*x1**-7)
        x2=x2-(x2**-6-y2)/(-6*x2**-7)

    solve.append(x1)
    solve.append(x2)
```

```
    return solve[0],solve[1]


def Romberg(a, b, err_min):
    kmax = 10
    tm = np.zeros(kmax,dtype = float)
    tm1 = np.zeros(kmax,dtype = float)
    tm[0] = 0.5*(b-a)*(fun(a) + fun(b))
    #print(tm)
    err = 1.
    k = 0
    np.set_printoptions(precision = 9)
    while(err>err_min and k <kmax-2):
        n = 2**k
        m = 1
        tm1[0] = T_2n(a, b, n, tm[0])
        while(err>err_min and m <= (k+1)):
            tm1[m] = tm1[m-1]+(tm1[m-1]-(tm[m-1]))/(4.**m-1)
            result = tm1[m]
            err1 = abs(tm1[m]-tm[m-1])
            err2 = abs(tm1[m]-tm1[m-1])
            err = min(err1,err2)
            m = m+1
        tm = np.copy(tm1)
        k = k+1
        #print(tm)
    return result

#fx=0
def f(n):
    fx=-(n+0.5)*np.math.pi
    fx=fx+Romberg(solve[0], solve[1], 1.e-12)*gamma
    return fx

def vx(x):
    return (x**(-12)-x**(-6))*4

def paint(ii):

    x1=inout(result[ii])
    n1=[vx(x1[0]),vx(x1[1])]
```

```
    plt.plot(x1,n1,color=col[i%6])
    solve.clear()

def paint2(i):
    e=result[i]/4
    #Lennard -Jones 势
    y2=(1-np.math.sqrt(1+4*e))/2
    y1=(1+np.math.sqrt(1+4*e))/2
    x1,x2=1.1,1.1
    for o in range(1,1000):
        x1=x1-(x1**-6-y1)/(-6*x1**-7)
        x2=x2-(x2**-6-y2)/(-6*x2**-7)
    xx=np.linspace(x1,x2,10000)
    #p=np.sqrt(np.abs(result[i]-vx(xx)))
    p=(np.abs(result[i]-vx(xx))**0.5)
    plt.plot(xx,p,color=col[i%6])
    plt.plot(xx,-p,color=col[i%6])



if __name__=="__main__":
    epsl=[]
    fnnn=[]
    result=[]
    for n in range(0,N):
        epsl=[]
        fnnn=[]
        while (1):

            solve=[]

            inout(epsilon)
            if (abs(f(n))<1.e-2):
                print(f(n),n,epsilon,solve)
                fnnn.append(f(n))
                epsl.append(epsilon)
            epsilon=epsilon+-epsilon/1000
            if epsilon>-0.0001:
                break
        print(fnnn)
        print(n,epsl)
```

```
        if (len(epsl)==0):
            print("MAX=",n-1)
            break
        mi=fnnn[0]
        for i in range(0,len(fnnn)):
            if abs(mi)>abs(fnnn[i]):
                mi=fnnn[i]
        for i in range(0,len(epsl)):
            if mi==fnnn[i]:
                result.append(epsl[i])
        print("r",result)
        epsilon=result[-1]
    print(result)
    #势能图
    fig = plt.figure(num=1)
    solve.clear()
    x=np.linspace(1,3,1000)
    v=4*(x**(-12)-x**(-6))
    plt.plot(x,v)
    for i in range(0,len(result)):
        paint(i)
    plt.title("D.L.J Potential Energy-D.r Diagram")
    plt.ylabel("D-L.J Potential Energy(v)")
    plt.xlabel("D.r(x)")

    #相图
    fig = plt.figure(num=2)
    for i in range(0,len(result)):
        paint2(i)
    plt.title("Momentum-D.r Diagram")
    plt.ylabel("Momentum")
    plt.xlabel("D.r(x)")



    fig = plt.figure(num=3)
    plt.subplot(2,  1,  1)
    # 绘制第一个图像

    x=np.linspace(1,3,1000)
    v=4*(x**(-12)-x**(-6))
    plt.plot(x,v)
```

```
    for i in range(0,len(result)):
        paint(i)
    plt.title("D.L.J Potential Energy-D.r Diagram")
    plt.ylabel("D-L.J Potential Energy(v)")
    plt.xlabel("D.r(x)")
    plt.xlim(0.9,3.1)
    plt.ylim(-1,0)
    # 将第二个 subplot 激活，并绘制第二个图像
    plt.subplot(2,  1,  2)
    for i in range(0,len(result)):
        paint2(i)
    plt.title("Momentum-D.r Diagram")
    plt.ylabel("Momentum")
    plt.xlabel("D.r(x)")
    plt.xlim(0.9,3.1)
    # 展示图像
    plt.show()
```

A.2 *Morse* 势理论程序实现

运行环境

*Ubuntu*9.3.0 − 17 *Ubuntu*1 20.04*LTS*

*Python*3.8.8 *Anaconda*4.10.3 *AMD* − 4800*u*

```
import numpy as np
import matplotlib.pyplot as plt
epslion=-0.99
N=200
gamma=21.7
solve=[]
kk=1
col=['orange','y','forestgreen','royalblue','m','r']
ee=np.math.e
def fun(x):            #被积函数，  x为自变量
    f = np.math.sqrt(round((epslion)-(vx(x)),10))
    return f

def T_2n(a, b, n, T_n):        #计算梯形积分
    if n<1:
        print('n should larger than 1')
    h = (b - a)/n              #步长
    sum_f = 0.             #初始化，中间变量
```

```
    for k in range(0, n):
        sum_f = sum_f + fun(a + (k + 0.5)*h)
    T_2n = T_n/2. + sum_f*h/2.
    return T_2n

#Morse
def inout(epslion):
    e=epslion
    x1=2**(1/6)-(np.math.log(1-np.math.sqrt(e+1)))/kk
    x2=2**(1/6)-(np.math.log(1+np.math.sqrt(e+1)))/kk
    if (x1>x2):
        x1,x2=x2,x1
    solve.append(x1)
    solve.append(x2)
    return solve[0],solve[1]


def Romberg(a, b, err_min):
    kmax = 6
    tm = np.zeros(kmax,dtype = float)       # 第m行所有的元素
    tm1 = np.zeros(kmax,dtype = float)      #第m+1行所有的元素
    tm[0] = 0.5*(b-a)*(fun(a) + fun(b))  # 初始值
    #print(tm)
    err = 1.
    k = 0
    np.set_printoptions(precision = 9)
    while(err>err_min and k <kmax-2):  #控制循环次数
        n = 2**k                       # n是区间等分数
        m = 1
        tm1[0] = T_2n(a, b, n, tm[0])
        while(err>err_min and m <= (k+1)):  #控制循环次数
            tm1[m] = tm1[m-1]+(tm1[m-1]-(tm[m-1]))/(4.**m-1)
            result = tm1[m]
            err1 = abs(tm1[m]-tm[m-1])
            err2 = abs(tm1[m]-tm1[m-1])
            err = min(err1,err2)
            m = m+1
        tm = np.copy(tm1)
        k = k+1
        #print(tm)
    return result
```

```
#fx=0
def f(n):
    fx=-(n+0.5)*np.math.pi
    fx=fx+Romberg(solve[0], solve[1], 1.e-8)*gamma
    return fx

def vx(x):
    return (1-ee**(-kk*x+kk*2**(1/6)))**2-1

def paint(ii):

    x1=inout(result[ii])
    n1=[vx(x1[0]),vx(x1[1])]
    plt.plot(x1,n1,color=col[i%6])
    solve.clear()

def paint2(i):
    #Morse 势
    e=result[i]
    x1=2**(1/6)-(np.math.log(1-np.math.sqrt(e+1)))/kk
    x2=2**(1/6)-(np.math.log(1+np.math.sqrt(e+1)))/kk
    xx=np.linspace(x1,x2,1000000)
    #p=(result[i]-(np.round(vx(xx))**0.5,4))
    p=(np.abs(result[i]-vx(xx))**0.5)
    plt.plot(xx,-p,color=col[i%6])
    plt.plot(xx,p,color=col[i%6])



if __name__=="__main__":
    epsl=[]
    fnnn=[]
    result=[]
    for n in range(0,N):
        epsl=[]
        fnnn=[]
        while (1):

            solve=[]
```

```
            inout(epslion)
            if (abs(f(n))<1.e-1):
                print(f(n),n,epslion,solve)
            fnnn.append(f(n))
            epsl.append(epslion)
            epslion=epslion+-epslion/1000
            if epslion>-0.0001:
                break
        print(fnnn)
        print(n,epsl)
        if (len(epsl)==0):
            print("MAX=",n-1)
            break
        mi=fnnn[0]
        for i in range(0,len(fnnn)):
            if abs(mi)>abs(fnnn[i]):
                mi=fnnn[i]
        for i in range(0,len(epsl)):
            if mi==fnnn[i]:
                result.append(epsl[i])
        print("r",result)
        epslion=result[-1]
    fig = plt.figure(num=1)
    #势能图
    solve=[]
    x=np.linspace(0,7,3000)
    v=vx(x)
    plt.plot(x,v)
    for i in range(0,len(result)):
        paint(i)
    plt.title("D.Morse Potential Energy-D.r Diagram")
    plt.ylabel("D-Morse Potential Energy(v)")
    plt.xlabel("D.r(x)")
    plt.ylim(-1,0)
    #plt.show()
    fig = plt.figure(num=2)
    #相图
    for i in range(0,len(result)):
        paint2(i)
    plt.title("Momentum-D.r Diagram")
    plt.ylabel("Momentum")
```

```
    plt.xlabel("D.r(x)")
    #plt.show()

    plt.figure(num=3)
    plt.subplot(2,  1,  1)
    # 绘制第一个图像
    x=np.linspace(0,7,1000)
    v=vx(x)
    plt.plot(x,v)
    plt.title("D.Morse Potential Energy-D.r Diagram")
    plt.ylabel("D-Morse Potential Energy(v)")
    plt.xlabel("D.r(x)")
    for i in range(0,len(result)):
        paint(i)
    plt.title("D.Morse Potential Energy-D.r Diagram")
    plt.ylabel("D-Morse Potential Energy(v)")
    plt.xlabel("D.r(x)")
    plt.xlim(0,7.1)
    plt.ylim(-1,0)
    # 将第二个 subplot 激活，并绘制第二个图像
    plt.subplot(2,  1,  2)
    for i in range(0,len(result)):
        paint2(i)
    plt.title("Momentum-D.r Diagram")
    plt.ylabel("Momentum")
    plt.xlabel("D.r(x)")
    plt.xlim(0,7.1)

    # 展示图像
    plt.show()
```

B 图片结果输出

B.1 H_2 分子振动能级

B.1.1 *Lennard-Jones* 势

图 5: $Lennard-Jones$ 势能级图

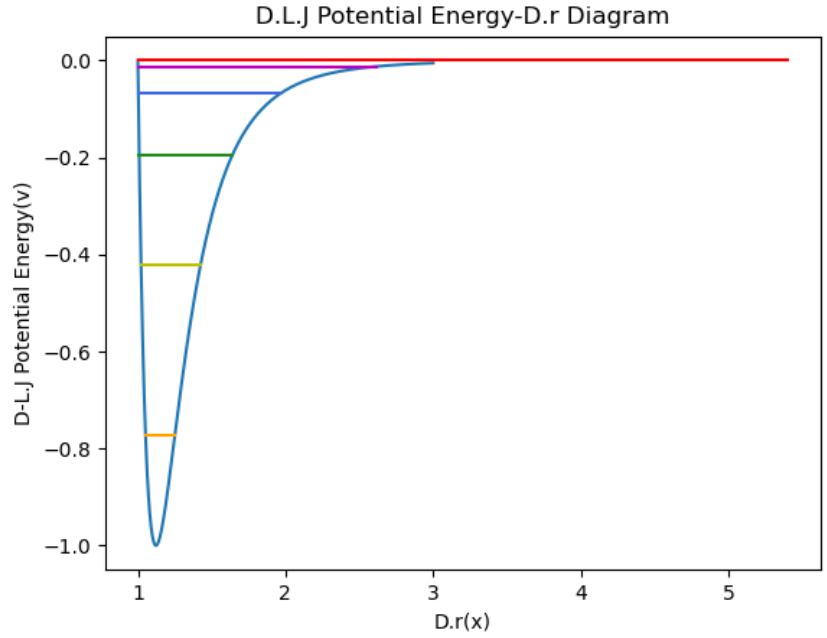


图 6: $Lennard-Jones$ 势相图

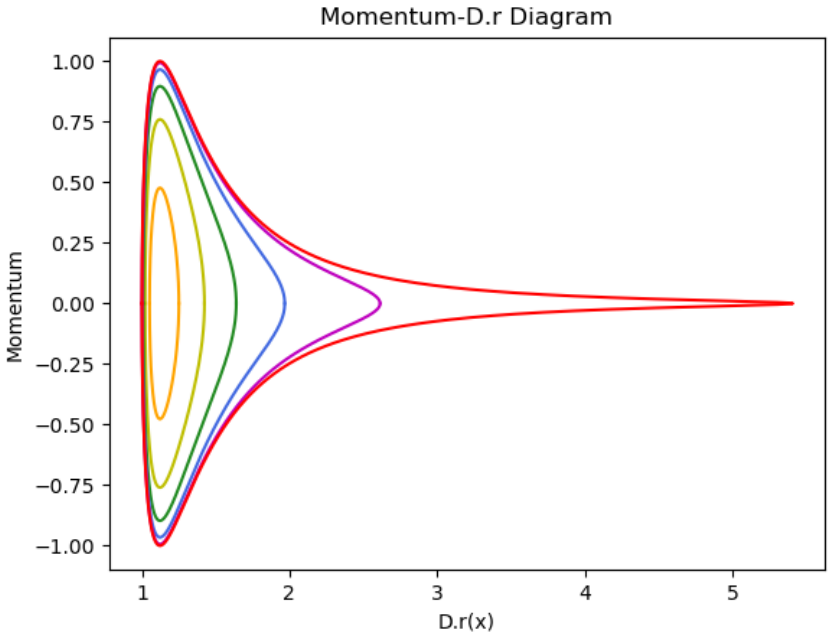
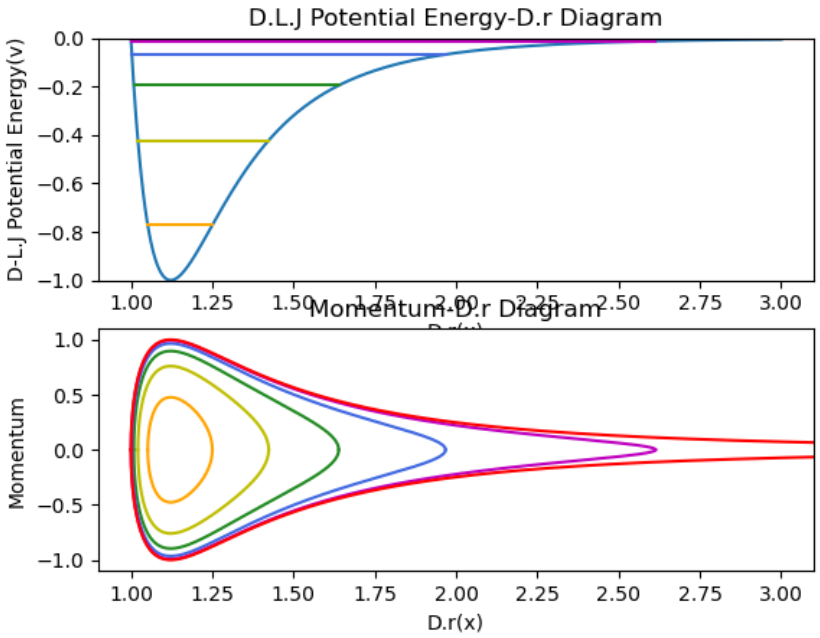


图 7: $Lennard-Jones$ 势能级图及相图



B.1.2 *Morse* 势

图 8: $Morse$ 势能级图

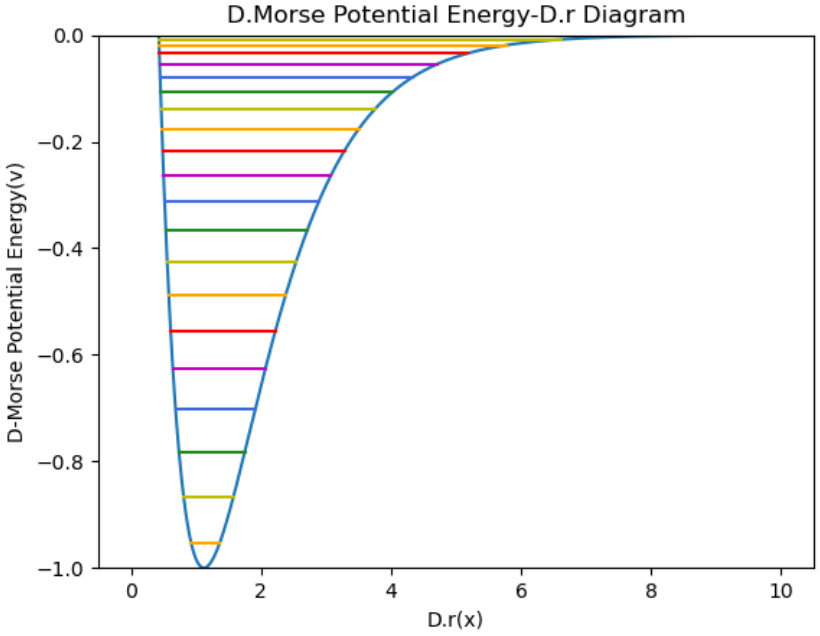


图 9: *Morse* 势相图

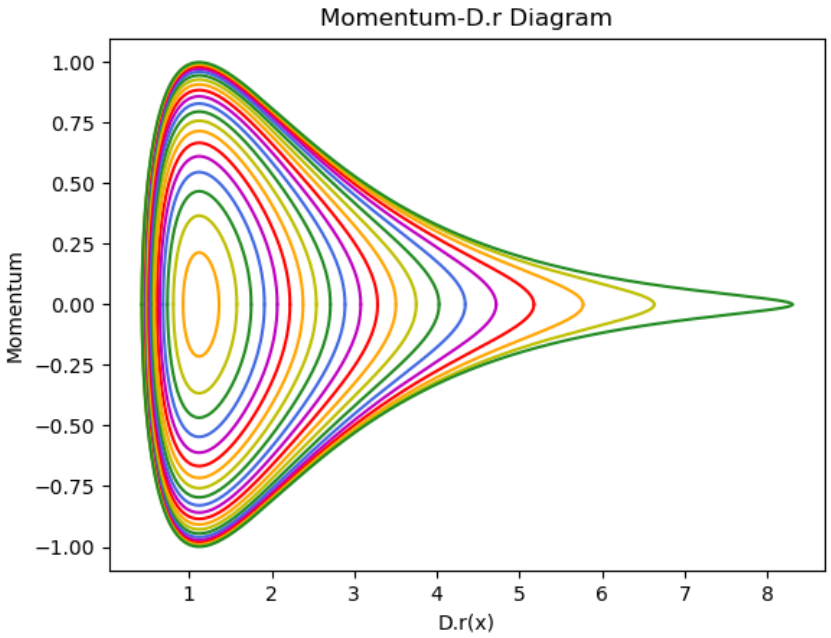
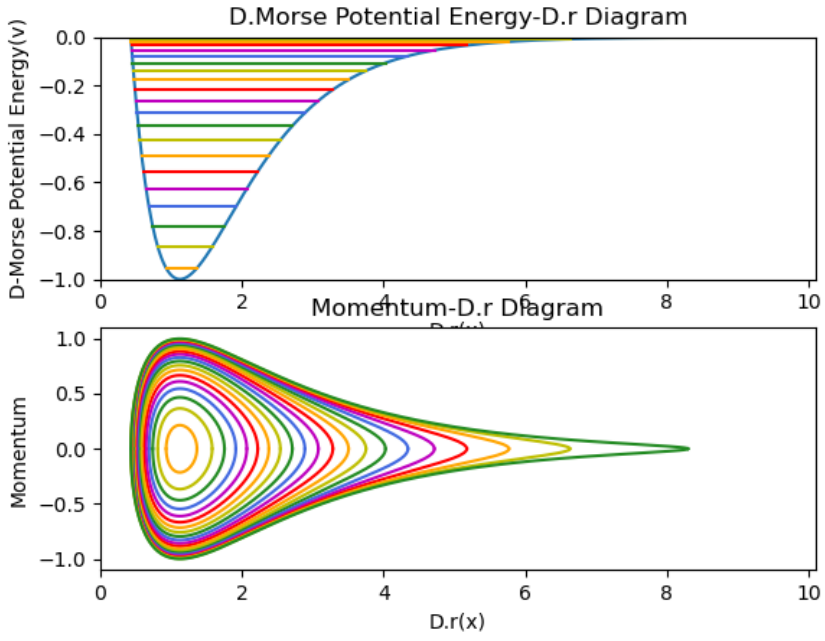


图 10: *Morse* 势能级图及相图



B.2 *DH* 分子振动能级

B.2.1 *Lennard-Jones* 势

图 11: $Lennard-Jones$ 势能级图

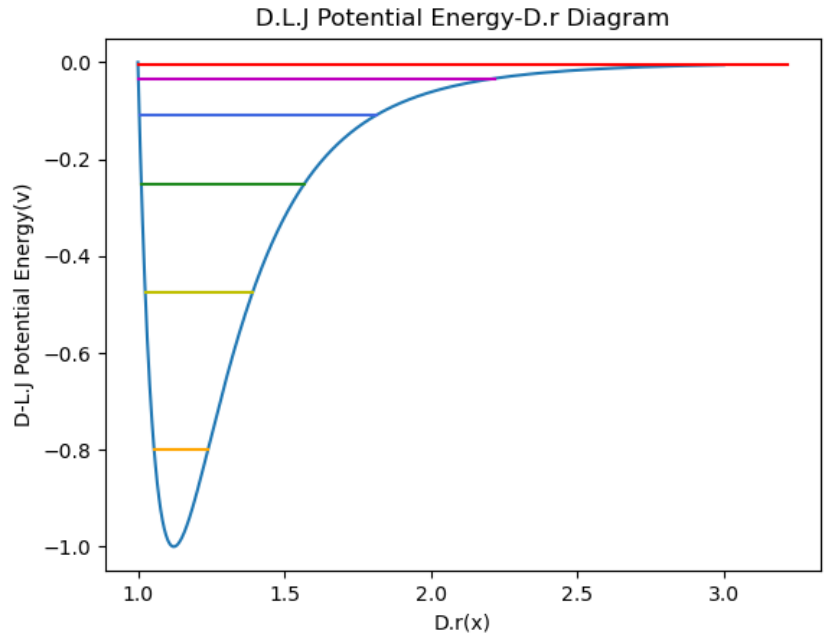


图 12: $Lennard-Jones$ 势相图

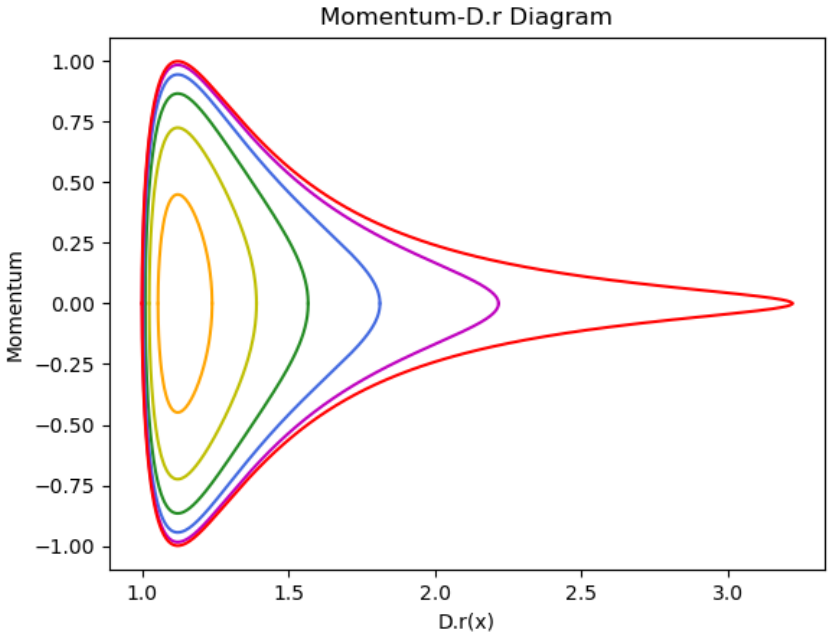
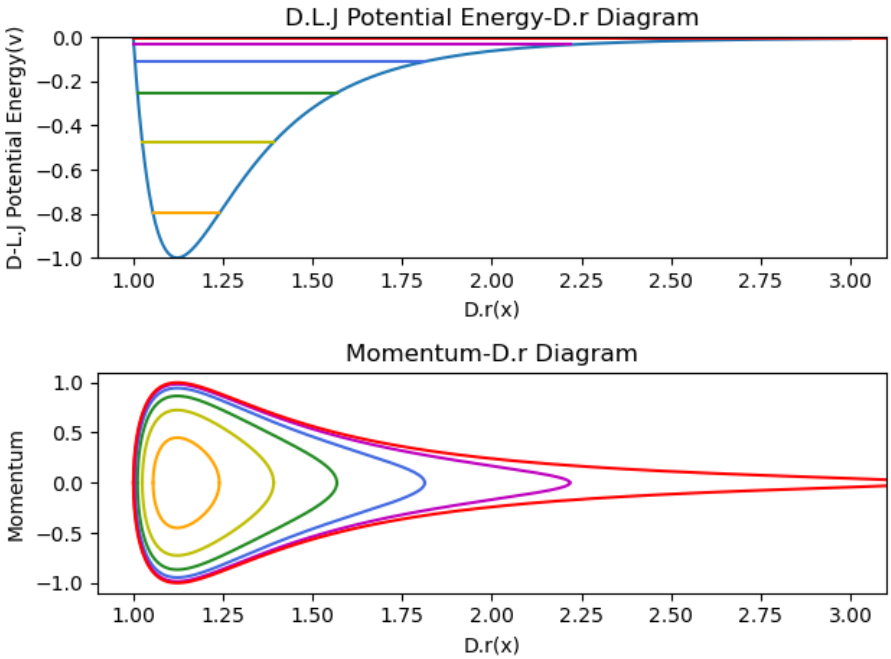


图 13: $Lennard-Jones$ 势能级图及相图



B.2.2 *Morse* 势

图 14: $Morse$ 势能级图

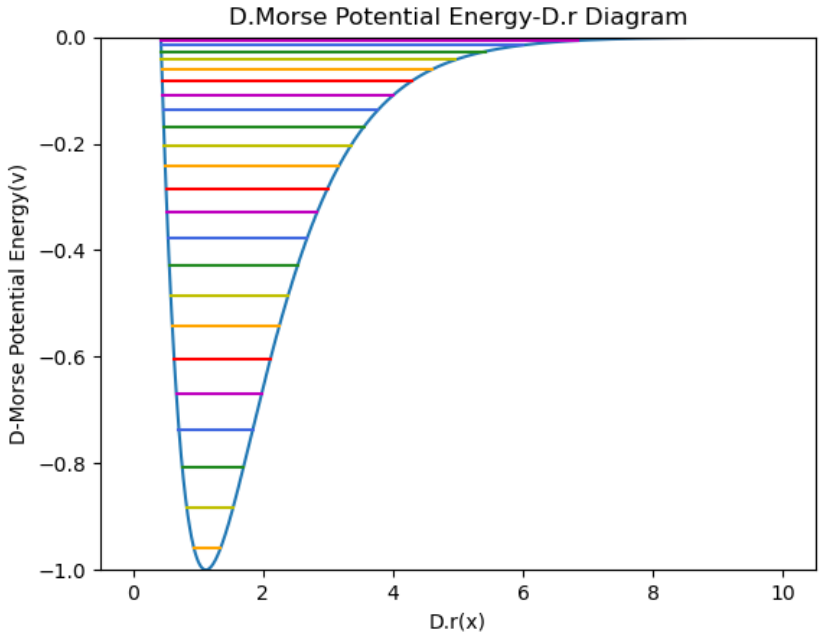


图 15: *Morse* 势相图

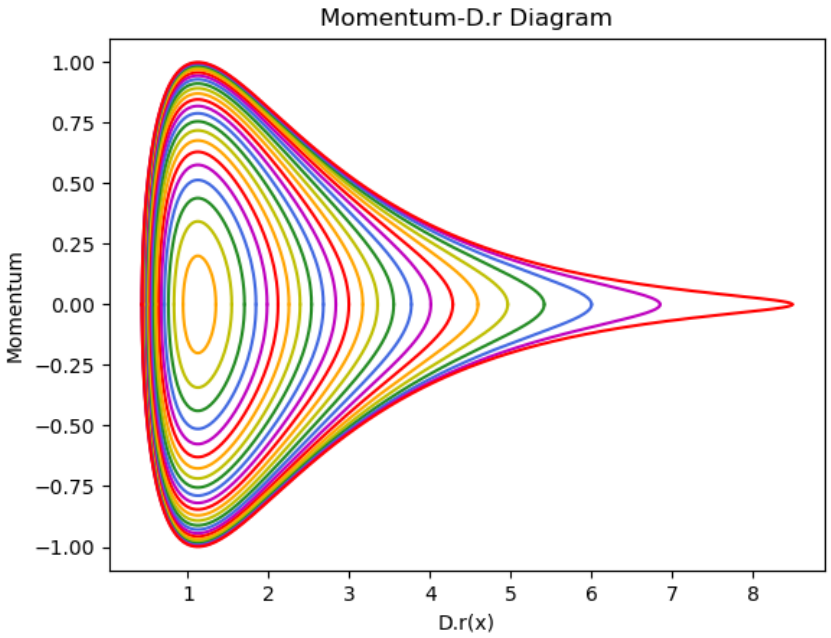
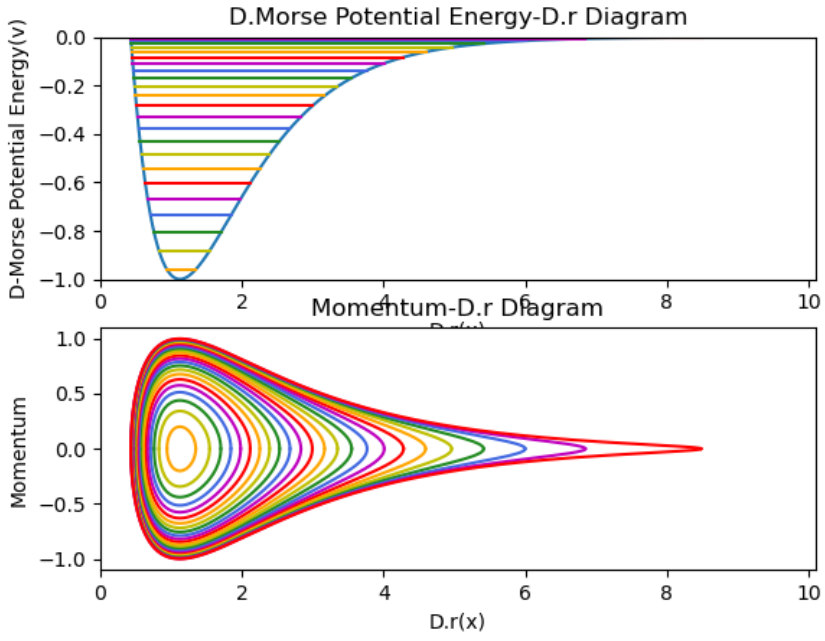


图 16: *Morse* 势能级图及相图



B.3 O_2 分子振动能级

B.3.1 *Lennard-Jones* 势

图 17: $Lennard-Jones$ 势能级图

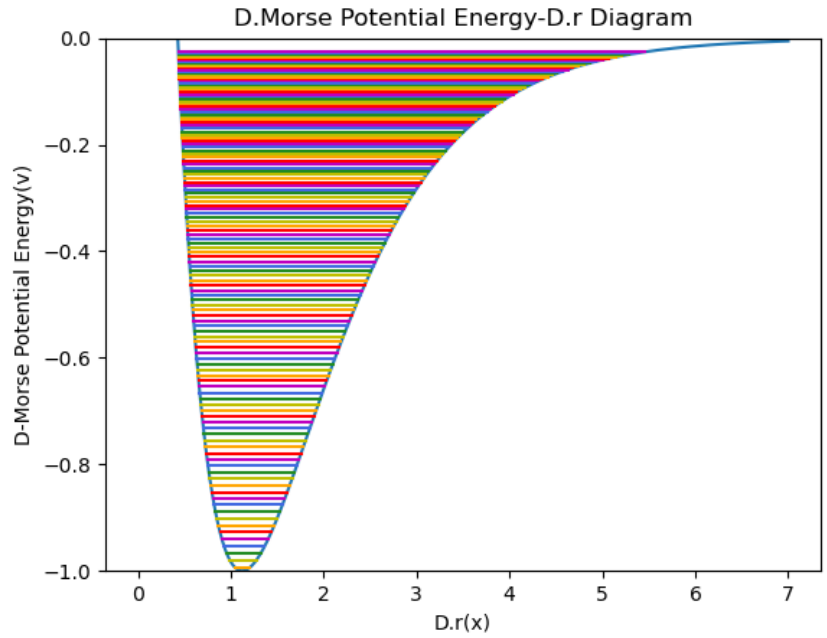


图 18: $Lennard-Jones$ 势相图

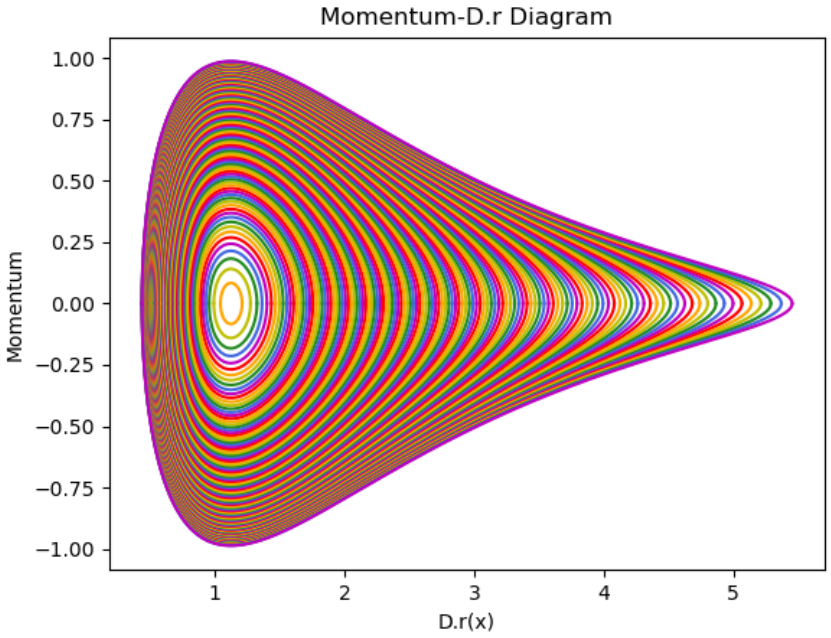
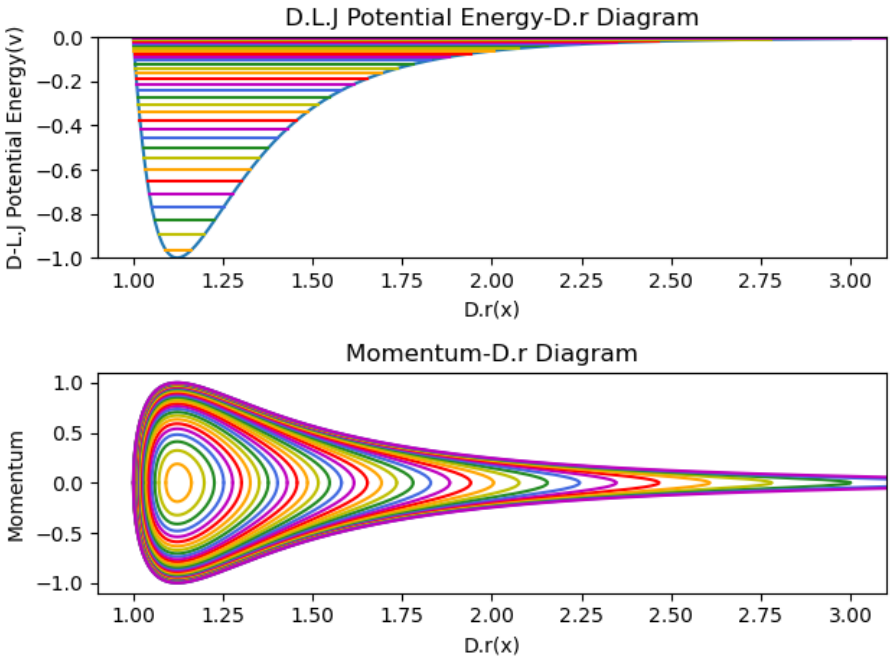


图 19: $Lennard-Jones$ 势能级图及相图



B.3.2 *Morse* 势

图 20: $Morse$ 势能级图

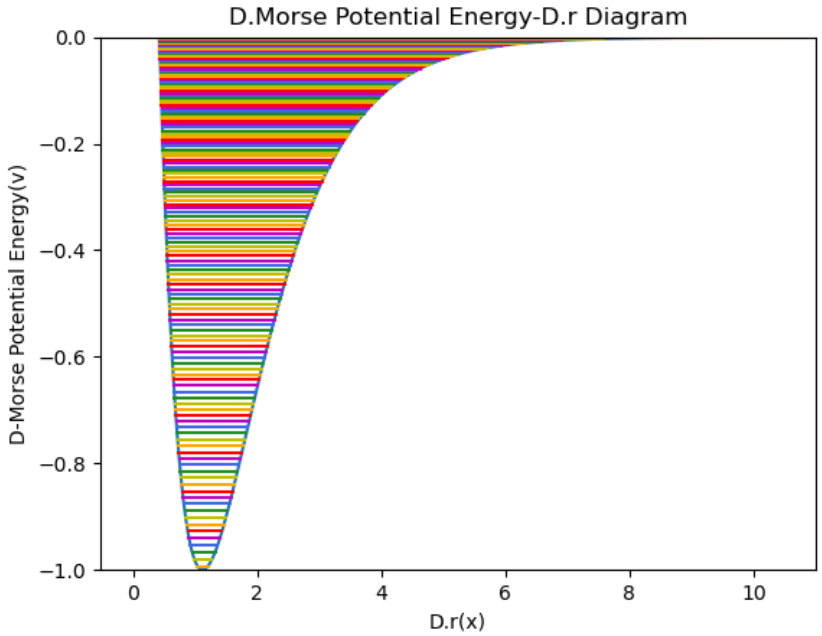


图 21: $Morse$ 势相图

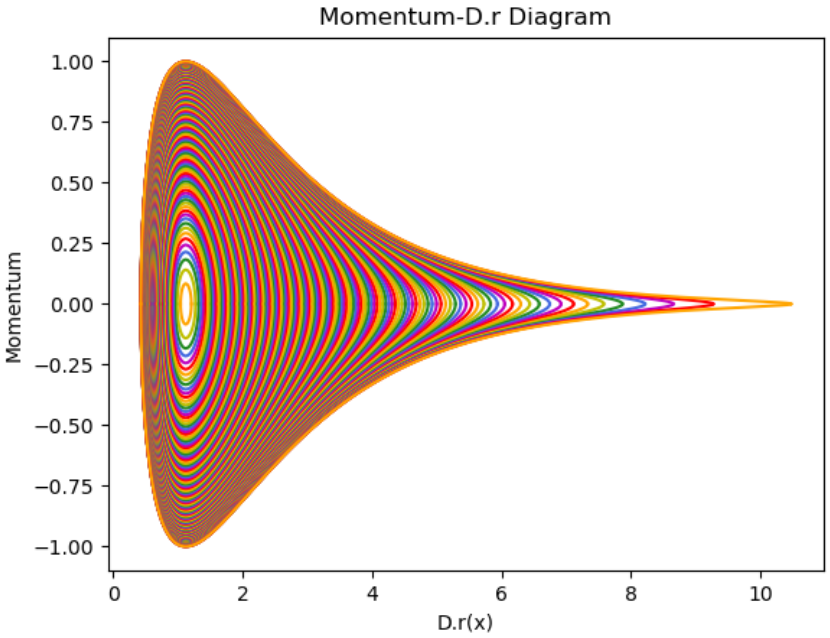


图 22: $Morse$ 势能级图及相图