第二章 物理学中的数值计算：插值、微分与积分

2.1 探测器刻度：插值法

在物理学实验研究中，很多时候需要在实验室自制探测仪器设备。原则上，任何介质，如其某一属性随待测物理量呈单调变化，则都可以用来制作探测仪器。例如，人们可以使用酒精、水银、煤油等液态物质的热胀冷缩效应来制作温度计。在核物理研究中，人们可以利用带电粒子在半导体硅材料中运动产生的电子-空穴对数目随带电粒子能量的单调变化关系，而将半导体硅材料制作成高分辨带电粒子能量探测器。此类探测仪器，在使用前均需要做仪器刻度。而数值插值则是仪器刻度中用到的最主要算法。本节内容，我们将学习两种常用的插值算法，即拉格朗日插值和样条插值。

2.2 从温度计说起

日常生活中最常见的探测器是温度计。在温度计的刻度定标过程，根据摄氏温标的定义，首先把温度计置入冰水混合物（图 2.1a），标记对应的液面位置为 v1，并记温度为 0 摄氏度（图 2.1b）；然后把温度计置入标准大气压下的沸水中，标记对应的液面位置为 v2，并记温度为 100 摄氏度。最后将液面 v1 和 v2 之差等分为 100 份，则每一份为 1.0 摄氏度，完成了温度计的刻度定标。

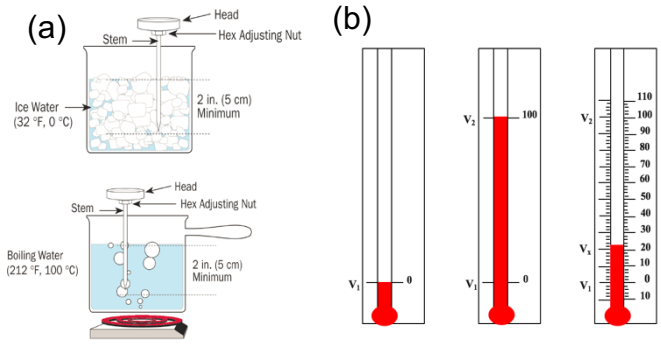

图 2.1. 温度计刻度定标示意图。

以上刻度定标过程，用数学语言可描述为：已知二维平面 v-t 上的(v1,t1)和（v2,t2）两个已知点（其中 t1=0；t2=100），则通过已知数据点的两点式直线方程为

$$t(v)=\frac{v-v_1}{v_2-v_1}\cdot t_2+\frac{v-v_2}{v_1-v_2}\cdot t_1 \qquad (2.1)$$

式（2.1）给出了温度计的刻度关系式，对任何待测物体，若测得的液面高度为 v，可根据（2.1）式计算出对应的温度值。这种构建解析函数表达式，使其通过已知数据点的过程叫做插值，是基本的数值计算方法。通过得到的插值函数，人们可以利用已知的离散数据求得未知数据，达到预测数据的目的。

2.3 拉格朗日插值

更一般地，式（2.1）可以写为

$$y(x)=\frac{x-x_2}{x_1-x_2}\cdot y_1+\frac{x-x_1}{x_2-x_1}\cdot y_2=A_1(x)y_1+A_2(x)y_2 \quad (2.2)$$

插值函数图形见图 2.2a 所示。其中，$A_1(x)=\frac{x-x_2}{x_1-x_2}$和$A_2(x)=\frac{x-x_1}{x_2-x_1}$分别为$x_1$和$x_2$的插值基函数。因此，插值函数 y(x)可以看作是两个已知点x_1和x_2处插值基函数$A_1(x)$和$A_2(x)$的线性组合，组合系数分别为已知点的函数值y_1和y_2。这种将插值函数表示为基函数线形组合的插值方法称为拉格朗日插值法。

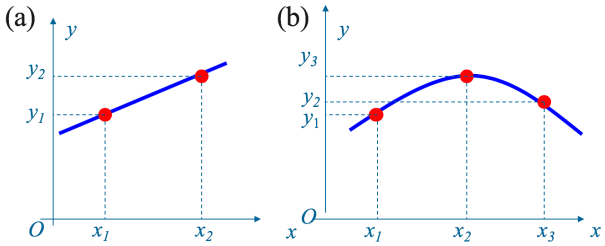

图 2.2 两点(a)和三点(b)插值示意图。

类似地，我们可以写出过二维平面上三个已知点(x_1,y_1)，(x_2,y_2)，和(x_3,y_3)的二次插值函数（图 2.2b）：

$$y(x)=A_1(x)y_1+A_2(x)y_2+A_3(x)y_3 \quad (2.3)$$

其中，插值基函数为：

$$A_1(x)=\frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}$$

$$A_2(x)=\frac{(x-x_3)(x-x_1)}{(x_2-x_3)(x_2-x_1)}$$

$$A_3(x)=\frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$$

以此类推，已知 n 个数据点 $(x_1,y_1),(x_2,y_2),\cdots(x_n,y_n)$，我们可以写出 n-1 次插值多项式：

$$y(x) = \sum_{j=1}^{n-1} A_j(x) y_j \qquad (2.4)$$

其中，插值基函数可以表示为：

$$A_j(x) = \prod_{\substack{i=1 \\ i \neq j}}^{n} \frac{x - x_i}{x_j - x_i} \qquad (2.5)$$

2.4 带电粒子探测器刻度

在粒子物理与核物理研究中，通常需要测量诸如α粒子、高能电子、质子、重离子等带电粒子的能量。通常采用的探测器有半导体探测器、气体探测器、塑料闪烁体探测器等。例如，高纯硅探测器是常用的带电粒子探测器（图 2.3a）。其基本原理是当带电粒子射入探测器介质后，所经之处硅原子会发生电离，产生电子-空穴对（图 2.3b）。在高压电源产生的强电场驱动下，电子向正电极移动并产生感应信号，被后续电子学元件收集并放大后，记录信号幅度。显然，带电粒子在探测器中沉积的能量越高，产生的电子-空穴对越多，信号幅度越大，因此可以将高纯硅作为探测器介质用来测量带电粒子的能量。在粒子探测问题中，直接测量所得到的是信号幅度，如何由信号幅度反推出粒子能量就需要对探测器进行刻度定标，构建信号幅度和粒子能量之间的解析函数关系式。

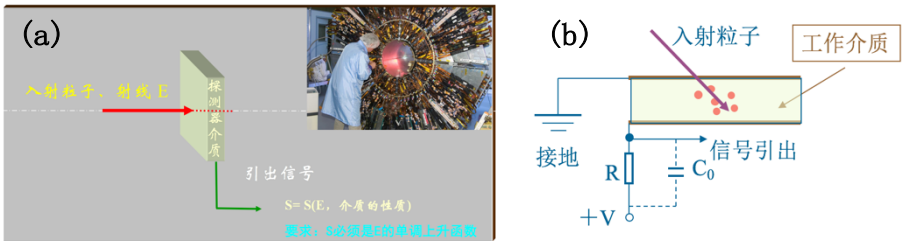

图 2.3 带电粒子探测器工作原理示意图。

问题 2-1：使用包括 Am241， Cm244， Pu239 的三组份α源产生的 5 种能量的α粒子，对高纯硅探测器进行能量刻度，得到的信号幅度见表 2.1 所示。若用该探测器设备探测未知能量的带电粒子，测得的信号幅度为 1588mV，请计算该粒子的能量为多少 keV？

表 2.1. 三组份α源产生的α粒子能量以及其沉积在探测器上所产生的信号幅度。

能量 E（keV）	5156	5440	5486	5763	5805
幅度 S（mV）	1390	1455	1561	1640	1705

对问题 2-1，基本思路是基于已知能量的 5 种α粒子得到的信号幅度值，构建插值函数，然后将待测粒子的信号幅度带入插值函数，估算能量值。以下是根据拉格朗日插值公式编写的 python 程序。

```
1 # -*- coding: utf-8 -*-
2 import numpy as np
3 import matplotlib.pylab as pl
4
5 energy_alpha_source=np.zeros(5,dtype=float)
6 signal_alpha_source=np.zeros(5,dtype=float)
7
8 with open('calibrate.dat','r') as fin:
9     i = 0
10    for lines in fin:
11        words=lines.split()
12        energy_alpha_source[i]=float(words[0])
13        signal_alpha_source[i]=float(words[1])
14        i = i + 1
15 def lagr_poly(x0,y0,n,x):
16    y=0.0
17    for i in range(0,n):
18        p=1.0
19        for j in range(0,n):
20            if(i!=j):
21                p=p*(x-x0[j])/(x0[i]-x0[j])
22        y=y+p*y0[i]
23    return y
24
25 signal=np.arange(1300,1800,1)
26 energy=[]
27 for x in signal:
28    xenergy=lagr_poly(signal_alpha_source,energy_alpha_source,5,x)
29    energy.append(xenergy)
30 print('enegy of particle:%8.3fkeV'%lagr_poly(signal_alpha_source,\
31              energy_alpha_source,5,1588))
32 pl.plot(signal_alpha_source,energy_alpha_source,'kD',label='alpha-source')
33 pl.plot(signal,energy,'r-',label='interpolation')
34 pl.xlabel('Signal(mV)')
35 pl.ylabel('Energy(keV)')
36 pl.xlim(1300,1800)
37 pl.ylim(5000,6000)
38 pl.legend(loc='upper left')
39 pl.show()
40
```

code-2-1

程序 code-2-1 中，第 5-6 行定义了两个包含 5 个元素的浮点类型数组 energy_alpha_source 和 signal_alpha_source，并将数组元素置为 0.这两个数组分别存储α粒子源的四种粒子能量和探测器测得的信号幅度。第 8-14 行将能量和信号幅度数值从数据文件“calibrate.dat”读入。第 15-23 行定义了一个子函数 lagr_poly，构建出拉格朗日插值函数，并返回函数值。第 27-29 行调用子函数 lagr_poly，计算出拉格朗日插值函数在（1300，1800）区间的数值。第 30 行再次调用子函数 lagr_poly，计算出信号幅度为 1588mV 的待测粒子能量。第 32-39 行将结果用图形表现出来。

运行程序 code-2-1.py,得到探测器的能量-信号幅度关系如图 2.4 所示，信号幅度为 1588mV 的待测粒子对应的能量为 5558keV。

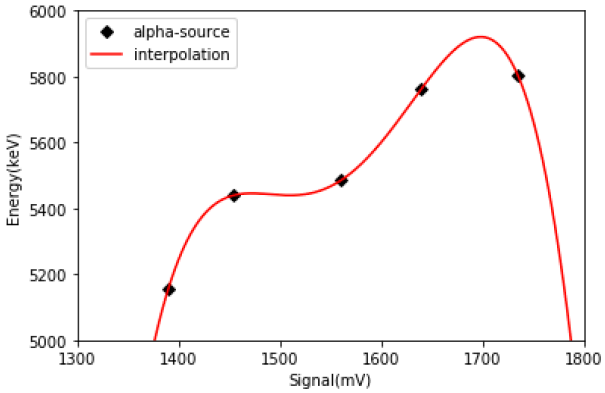

图 2.4. α粒子的能量与探测器测得的信号幅度关系（方实点），以及基于拉格朗日插值得到的能量-幅度函数曲线（红实线）。

在实际拉格朗日插值计算中，通常不需要象 code-2-1 那样自己编写程序来构建拉格朗日插值函数，而是直接调用 scipy 程序库中的 interpolate 模块来实现拉格朗日插值函数的构建。在 code-2-2.py 中，第 4 行加载了 scipy 的 interpolate 模块。第 16 行调用 interpolate 模块中的 lagrange 函数，构建出拉格朗日插值函数 poly。在第 22 行和第 25 行，分别调用 poly 函数计算给定信号幅度对应的能量值。运行 code-2-2.py 得到与图 2.4 相同的结果。

```
1 # -*- coding: utf-8 -*-
2 import numpy as np
3 import matplotlib.pylab as pl
4 from scipy import interpolate
5
6 energy_alpha_source=np.zeros(5,dtype=float)
7 signal_alpha_source=np.zeros(5,dtype=float)
8 with open('calibrate.dat','r') as fin:
9     i = 0
10    for lines in fin:
11        words=lines.split()
12        energy_alpha_source[i]=float(words[0])
13        signal_alpha_source[i]=float(words[1])
14        i = i + 1
15
16 poly = interpolate.lagrange(signal_alpha_source, energy_alpha_source) # from scipy
17
18 signal=np.arange(1300,1800,1)
19 energy=[]
20
21 for x in signal:
22     xenergy=poly(x)
23     energy.append(xenergy)
24
25 print('enegy of particle:%8.3fkeV'%poly(1588))
26 pl.plot(signal_alpha_source,energy_alpha_source,'kD',label='alpha-source')
27 pl.plot(signal,energy,'r-',label='interpolation')
28 pl.xlabel('Signal(mV)')
29 pl.ylabel('Energy(keV)')
30 pl.xlim(1300,1800)
31 pl.ylim(5000,6000)
32 pl.legend(loc='upper left')
33 pl.show()
34
```

code-2-2

2.5 三次样条插值

根据以上拉格朗日多项式插值法，n 个数据点可以构造出 n-1 次插值多项式。当 n 较大时，插值函数为高阶多项式，通常会出现震荡行为，从而引入非物理特征。解决这一问题的方法有两种，1）采用分段拉格朗日插值，即把 n 个数据点划分为不同的区间，每个区间分别作拉格朗日插值，将不同区间的插值函数整合起来极为原数据的插值函数。这种分段插值可以避免由于高阶多项式而引入的非物理震荡行为，缺点是不同区间构建的插值函数不连续；2）采用样条插值法，使得不同区间的插值函数光滑连接，得到整体光滑的插值函数。下面以三次样条插值为例，介绍样条插值法。

图 2.5 样条插值示例。

三次样条函数的特征是一阶导数处处光滑，二阶导数处处连续。为了构造符合以上性质的插值函数，我们在区间$(x_j，\ x_{j+1})$内构造以下三次函数：

$$y = Ay_j + By_{j+1} + Cy_j^{''} + Dy_{j+1}^{''} \qquad (2.6)$$

其中，$A = \frac{x-x_{j+1}}{x_j-x_{j+1}}$，$B = \frac{x-x_j}{x_{j+2}-x_j}$为区间$(x_j，\ x_{j+1})$左右端点的线性插值基函数。选取 $C = \frac{1}{6}(A^3 - A)(x_{j+1} - x_j)^2$，$D = \frac{1}{6}(B^3 - B)(x_{j+1} - x_j)^2$，使得（2.6）式后两项的值在区间$(x_j，\ x_{j+1})$左右端点为零，而且插值函数 y 的二阶导数从左端点到右端点线性地由$y_j^{''}$变为$y_{j+1}^{''}$。显然，以上构建的三次插值函数在各个区间边界处满足一阶导数处处光滑，二阶导数处处连续的要求。但是，（2.6）式的$y_j^{''}$和$y_{j+1}^{''}$未知。为了求得各个区间边界点的二阶导数$y_j^{''}$和$y_{j+1}^{''}$，我们对插值函数（2.6）式求一阶导数：

$$\frac{dy}{dx}=\frac{y_{j+1}-y_j}{x_{j+1}-x_j}-\frac{3A^2-1}{6}(x_{j+1}-x_j)y_j^{''}+\frac{3B^2-1}{6}(x_{j+1}-x_j)y_{j+1}^{''} \quad (2.7)$$

根据三次样条函数的要求，一阶导数处处连续，可得如下 n-2 个方程：

$$\frac{x_j-x_{j-1}}{6}y_{j-1}^{''}+\frac{x_{j+1}-x_{j-1}}{3}y_j^{''}+\frac{x_{j+1}-x_j}{6}y_{j+1}^{''}=\frac{y_{j+1}-y_j}{x_{j+1}-x_j}-\frac{y_{j+1}-y_j}{x_{j+1}-x_j} \quad (2.8)$$

另外，采用自然边界条件，即$y_1^{''}=0$；$y_N^{''}=0$,则可完全确定（2.6）式的$y_j^{''}$和$y_{j+1}^{''}$，得到三次样条插值函数。

实际使用中，我们可以调用 scipy 函数库中的插值模块来实现。以下的程序 code-2-3.py 给出了使用三次样条插值实现探测器插值函数的构建。

```
1  # -*- coding: utf-8 -*-
2  import numpy as np
3  import matplotlib.pylab as pl
4  from scipy import interpolate
5
6  energy_alpha_source=np.zeros(5,dtype=float)
7  signal_alpha_source=np.zeros(5,dtype=float)
8  with open('calibrate.dat','r') as fin:
9      i = 0
10     for lines in fin:
11         words=lines.split()
12         energy_alpha_source[i]=float(words[0])
13         signal_alpha_source[i]=float(words[1])
14         i = i + 1
15
16 # cubic spline interpolation
17 tck = interpolate.splrep(signal_alpha_source, energy_alpha_source,k=3,s=1.2)
18 print(tck)
19
20 signal=np.arange(1300,1800,1)
21 energy = interpolate.splev(signal, tck, der=0)
22
23 print('enegy of particle:%8.3fkeV'%interpolate.splev(1588, tck, der=0))
24
25 pl.plot(signal_alpha_source,energy_alpha_source,'kD',label='alpha-source')
26 pl.plot(signal,energy,'r-',label='interpolation')
27 pl.xlabel('Signal(mV)')
28 pl.ylabel('Energy(keV)')
29 pl.xlim(1300,1800)
30 pl.ylim(5000,6000)
31 pl.legend(loc='upper left')
32 pl.show()
33
```

code-2-3

程序 code-2-3 中的 17 行调用函数 splrep()，返回值是三元组 tck，包括节点向量、系数以及插值函数阶数等信息。这些信息完全定义了（2.6）式给出的插值函数。第 21 行计算出(1300,1800)范围内的信号幅度对应的粒子能量值，计算结果如图 2.6 所示。所测得的信号幅度为 1588mV 的粒子对应的能量为 5561keV.

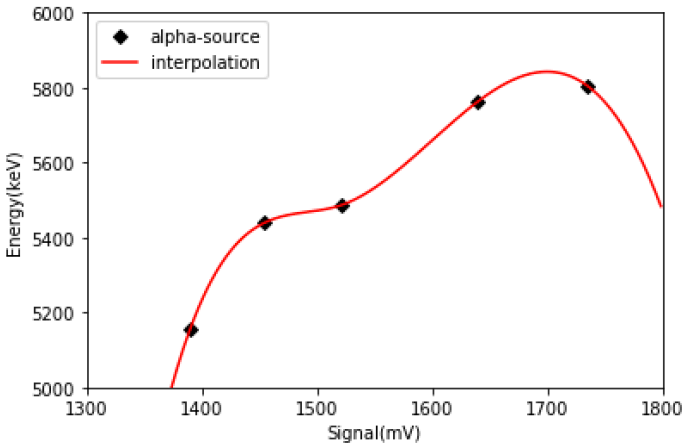

图 2.6. 带电粒子探测器刻度过程中α粒子源的能量和测得的信号幅度，以及三次样条插值函数曲线。

作业 2-1：数据文件“nCov-2019.dat”给出的是 2020-01-26 至 2020-02-24 期间新冠肺炎确诊病例数据，包括全国确诊病例数，湖北省确诊病例数以及湖北省以外确诊病例数。1）请编写 python 程序，分别画出以上确诊病例数随时间变化的散点图及其三次样条插值函数曲线，并预测未来 2 天确诊病例人数；2）改用拉格朗日插值法预测未来 2 天确诊病例人数；3）改用分段拉格朗日插值法预测未来 2 天确诊病例人数；4）对使用以上三种方法进行插值和预测的结果进行讨论；5）通过以上计算、讨论和适当文献调研，对得到更合理的预测提出建议。

2.6 数值微分与比热计算

基于以上数值插值法，我们可以找到离散数据的近似解析表达式。有了解析表达式，便可以很方面地求得离散数据的微分和积分。这便是数值微分与数值积分的基本思路。根据（2.1）式，已知(x_i, y_i)和(x_{i+1}, y_{i+1})两个相邻数据点，我们可以构建线性插值函数

$$y(x) = \frac{x-x_{i+1}}{x_i-x_{i+1}} \cdot y_i + \frac{x-x_i}{x_{i+1}-x_i} \cdot y_{i+1} \qquad (2.9)$$

然后，由插值函数$y(x)$的微分来代替离散数据在（x_i, x_{i+1}）区间的微分，即

$$y'(x) = \frac{y_{i+1}-y_i}{x_{i+1}-x_i} = \frac{y_{i+1}-y_i}{h} \qquad (2.10)$$

其中，$h = x_{i+1} - x_i$为离散数据的自变量变化步长。式（2.10）即为一阶微分的有限差分公式。更一般地，离散数据在x_i点的一阶微分，根据所使用的插值数据点，可通过如下三种差分格式计算：

向前差分格式（图 2.3a），

$$y'(x_i)=\frac{y_{i+1}-y_i}{x_{i+1}-x_i}=\frac{y_{i+1}-y_i}{h} \qquad (2.11)$$

向后差分格式（图 2.3b），

$$y'(x_i)=\frac{y_i-y_{i-1}}{x_i-x_{i-1}}=\frac{y_i-y_{i-1}}{h} \qquad (2.12)$$

中心差分格式（图 2.3c），

$$y'(x_i)=\frac{y_{i+1}-y_{i-1}}{x_{i+1}-x_{i-1}}=\frac{y_{i+1}-y_{i-1}}{2h} \qquad (2.13)$$

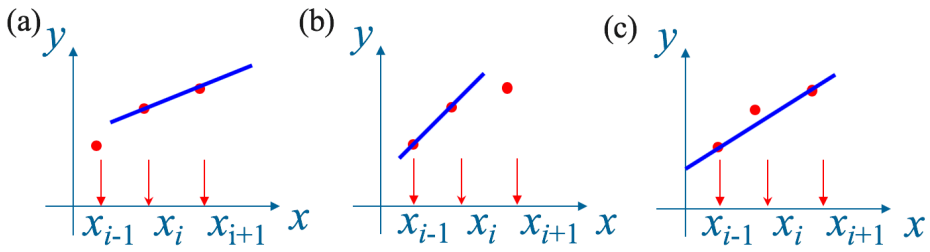

图 2.7 一阶微分的有限差分法示意图。

同样地，利用离散数据的相邻三个点$(x_{i-1},y_{i-1}),(x_i,y_i),(x_{i+1},y_{i+1})$，根据式（2.3）可以构建二次插值函数

$$y(x)=\frac{(x-x_i)(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})}y_{i-1}+\frac{(x-x_{i+1})(x-x_{i-1})}{(x_i-x_{i+1})(x_i-x_{i-1})}y_i+\frac{(x-x_{i-1})(x-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)}y_{i+1}$$

进而可以得到x_i点处二阶微分的差分格式如下：

$$y''(x_i)=\frac{y_{i-1}-2y_i+y_{i+1}}{h^2} \qquad (2.14)$$

以上差分格式既可以用于计算离散数据的一阶和二阶微分，也是将来学习常微分和偏微分方程数值解法的基础。物理学中，多个场合可能会用到差分法计算微分。例如，根据所记录的物体运动轨迹上的离散位置点，来估算物体在各个时刻的运动速度；通过物体在升温过程中的吸热 / 放热数据来计算物体的比热等。

问题 2-2：蛋白质分子在高温时会发生变性，对应蛋白质分子的解折叠相变过程。实验上，人们可以通过量热技术（DSC）测量蛋白质分子体系在升温过程中的内

能变化（吸热 / 放热）；理论上，不同温度下的分子动力学模拟也可以计算出蛋白质分子的内能随温度的变化。表 2.2（数据文件 Etot-Temp.dat）给出的是分子动力学模拟计算出来的不同温度下的蛋白质 CI2 的内能，请编写 python 程序，计算蛋白质 CI2 在不同温度下的比热，并估计该蛋白质分子的相变温度T_m。

表 2.2 分子动力学模拟得到的不同温度（T）下蛋白质分子 CI2 的内能（E）。

T(K)	E(kcal/mol)	T(K)	E(kcal/mol)	T(K)	E(kcal/mol)
300.0	-100.5	335.0	-81.4	370.0	-15.0
305.0	-100.3	340.0	-69.3	375.0	-12.2
310.0	-100.0	345.0	-54.5	380.0	-11.7
315.0	-99.1	350.0	-34.2	385.0	-10.5
320.0	-97.4	355.0	-25.8	390.0	-10.3
325.0	-94.2	360.0	-20.4	395.0	-10.2
330.0	-89.7	365.0	-18.9	400.0	-10.1

由于不同温度下的分子动力学模拟是在等容等温条件下进行的，因此等容比热可由内能对温度的导数给出，即$C_v = dE / dT$。以下 python 程序 code-2-4 采用一阶中心差分法由表 2.2 离散数据计算比热C_v随温度的变化。

```python
# -*- coding: utf-8 -*-
import matplotlib.pylab as pl
# ---------------------- read in temperature and energy----------------------
temperature = []
energy=[]
with open('Etot-Temp.dat', 'r') as fin:
    for lines in fin:
        words = lines.split()
        T = float(words[0])
        E = float(words[1])
        temperature.append(T)
        energy.append(E)
n = len(temperature)
Cv=[0]*n
for i in range(1,n-1):
    Cv[i] = (energy[i+1] - energy[i-1])/(temperature[i+1] - temperature[i-1])

fig, ax = pl.subplots(nrows = 2, ncols = 1)

ax[0].plot(temperature, energy, marker='o',color='k', linewidth=2.0, label='Energy')
ax[1].plot(temperature, Cv, color='b', linewidth=2.0,label='Cv')

ax[0].set_xlim((300, 400))
ax[0].set_ylim((-110., -5.))
ax[1].set_xlim((300, 400))
ax[1].set_ylim((0., 4.))

ax[0].legend(loc='upper left')
ax[1].legend(loc='upper left')

ax[1].set_xlabel(r'Temperature(K)', fontsize=20)
ax[0].set_ylabel(r'E(kcal/mol)', fontsize=20)
ax[1].set_ylabel(r'Cv', fontsize=20)
pl.show()

```

code-2-4

程序 code-2-4 中，第 6-12 行读入不同温度下的蛋白质分子的温度和能量数据，分别记录在列表 temperature 和 energy 中。第 14 行定义了一个包含 n 个元素的列表 Cv，为后续记录比热数据做准备，且列表元素都设为 0。第 15-16 行利用中心差分格式，计算出能量对温度的一阶微分，即比热，并更新列表 Cv 中的元素值。第 18 行定义了一个包含 2 个面板的图。20-21 行分别在图中的两个面板画出能量随温度的变化以及比热随温度的变化。

运行程序 code-2-4，得到的结果如图 2.8 所示。从图 2.8 我们可以看出，在温度较低时（<330K），能量随温度变化较缓慢，蛋白质维持在折叠状态。当温度由 340K 升高至 360K 的过程中，蛋白质分子的能量显著升高，蛋白质解折叠。当温度高于 360K 后，蛋白质的能量随温度变化变缓慢，蛋白质分子维持在解折叠状态。相应地，蛋白质的比热随温度先增加，在 345K 附近达到峰值，然后随温度升高而降低。因此，蛋白质分子 CI2 的解折叠相变温度T_m约为 345K。

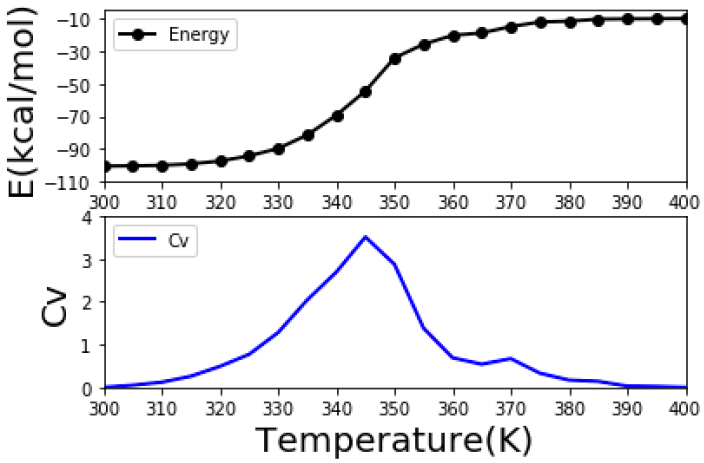

图 2.8.（上图）分子动力学模拟得到的蛋白质分子内能随温度的变化；（下图）差分法得到的等容比热随温度的变化。

2.7 数值积分与带电体静电势计算

积分是物理学中常见的计算问题。相对于微分计算，解析求解定积分的过程更为复杂困难，甚至绝大多数的解析函数，很难通过解析的方式求得积分。因此，数值积分法不仅是计算离散数据的定积分的主要方法，也是计算复杂解析函数的定积分的常用方法。

问题 2-3：如图 2.9 所示，一长为 2m 的带电细杆，其电荷线密度为$\lambda(x) = e^{-x^2}$. 请计算带电细杆在二维平面(x, y)内产生的静电势分布。

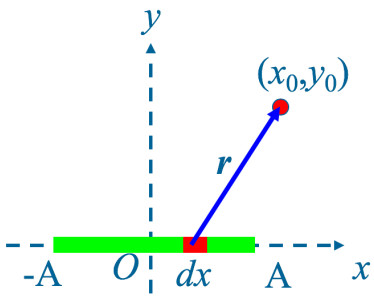

图 2.9. 带点细杆在 2 维平面(x, y)内的真空环境中产生的静电势和静电场。

根据电磁学的知识，带电细杆在二维平面任何一点(x_0, y_0)处产生的电势为：

$$\boldsymbol{V}(x_0, y_0) = \frac{1}{4\pi\varepsilon_0}\int_{-1}^{1}\frac{\lambda(x)}{r}dx = \frac{1}{4\pi\varepsilon_0}\int_{-1}^{1}\frac{e^{-x^2}}{[(x-x_0)^2+y_0{}^2]^{1/2}}dx \quad (2.15)$$

这样，求电势分布的物理问题转化为计算定积分的数学问题。更一般地，如果定义

$$\boldsymbol{f(x)} = \frac{1}{4\pi\varepsilon_0}\frac{e^{-x^2}}{[(x-x_0)^2+y_0{}^2]^{1/2}} \quad (2.16)$$

则, 以上定积分可以写为如下一般表达式：

$$\boldsymbol{V} = \int_a^b f(x)\,dx \quad (2.17)$$

定积分的几何意义是曲线$f(x)$在(a, b)区间内与坐标轴 x 围成的面积，如图 2.9 阴影部分所示。为了求得阴影部分面积，我们可以将区间（a, b）等分为 n 个小区间，宽度为$\Delta x = (b-a)/n$，则阴影部分面积可以近似地表示为宽度为Δx，高度为$f(x_i)$的 n 个矩形面积ΔS_i（图 2.10（a）红色矩形区域）之和，即：

$$\boldsymbol{V} = \sum_{i=1}^{n}\Delta S_i = \sum_{i=1}^{n} f(x_i)\Delta x = f(x_1)\Delta x + f(x_2)\Delta x + \cdots + f(x_n)\Delta x \quad (2.18)$$

式（2.18）即为定积分的**矩形公式**，Δx称为积分步长。显然，当积分步长Δx无限趋近 0 时，式（2.18）能够给出定积分 V 的精确计算。

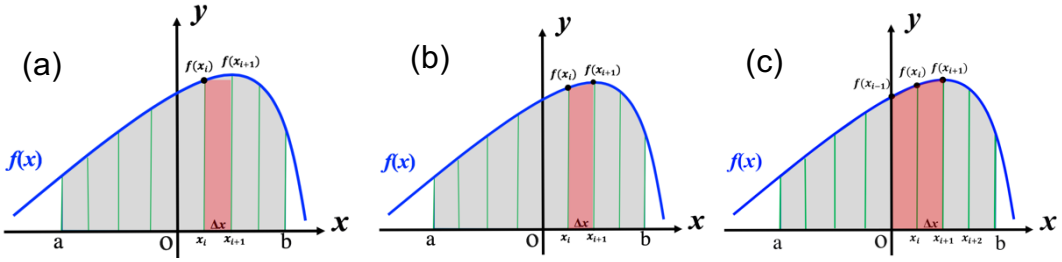

图 2.10. 矩形积分、梯形积分、抛物线积分示意图。

程序 code-2-5 给出了用矩形法计算带电细杆静电势的代码。第 6-7 行定义了被积函数。第 9 行给出积分区间的划分，即划分为 100 个区间，共 101 个点。第 10-11 行确定了带点细杆的长度以及计算得到的电势值的单位。为简化起见，本例选取 $1/4\pi\varepsilon_0$ 为电势值的单位。第 14 行计算出积分步长 h。第 16-17 行计算出积分区间内的节点坐标值。22-28 行是对所有待计算电势的坐标点 (x_0, y_0) 循环，计算出每个坐标点的电势值。其中 26-28 是利用矩形积分法对给定 (x_0, y_0) 点进行电势计算，并将得到的计算结果存储于列表 field 中。第 30-31 行将列表 field 转化为数组。由于 afield 的默认横坐标为 y，纵坐标为 x，为了作图方便，第 32 行将 afield 进行转置，得到横坐标为 x，纵坐标为 y 的数组。34-50 行将计算得到的电势分布用两种不同的等高图表现出来。其中，第 34 行确定绘制电势值等高图的 (x_0, y_0) 范围。第 35 行创建一个新图；第 38 行在新图上进一步创建了一个子图；第 39 行确定了待展示的电势值的范围。第 40-41 行画出电势值的等高线，并进行数字标记。第 44-45 行创建了另一个子图，并画出电势值的填充模式等高图。

```
1  # -*- coding: utf-8 -*-
2  import numpy as np
3  import matplotlib.pyplot as pl
4
5  #---------defining a function------------
6  def f(x,x0,y0):
7      return np.exp(-x**2)/np.sqrt((x-x0)**2+y0**2)
8
9  N=101  # number of points
10 A=1.0  # half-length of the stem
11 c=1.0  # determine the unit of filed. c= 1/4piepsilon
12
13 xx=np.zeros(N)
14 h = 2.0*A/float(N-1)  # integration step
15
16 for i in np.arange(0,N,1):
17     xx[i] = i*h - A    # coordinate aling the x-axis
18
19 xaxis = np.arange(-5.0,5.0,0.2) # the the (x0.y0) points
20 yaxis = np.arange(-5.0,5.0,0.2)
21 field=[] # list to store the calculated field.
22 for x0 in xaxis:   #calculate the fields for a number of (x0,y0) pionts
23     for y0 in yaxis:
24         v = 0.0
25 #rectangular integration
26         for i in np.arange(0,N-1,1):
27             v = v + f(xx[i],x0,y0)*h
28         field.append(v)
29
30 afield = np.array(field) # convert list to array
31 afield.shape = len(xaxis), len(yaxis) #setup the dimension of array
32 afield_xy = afield.T #so that X --> x axis
33
34 extent = [-5.0, 5.0, -5.0, 5.0] # the range of points to be plotted.
35 fig = pl.figure(figsize=(9,4)) # create a figure
36
37 #contour line
38 ax1 =fig.add_subplot(1,2,1) # create a subplot
39 levels = np.arange(0.0,5.0,0.05) # setup the level of field to show.
40 cs = ax1.contour(afield_xy,levels,origin='lower',linewidths=2,extent=extent) # contour line
41 ax1.clabel(cs)
42
43 # color bar
44 ax2=fig.add_subplot(1,2,2)
45 cs = ax2.contourf(afield_xy,levels,origin='lower',extent=extent,cmap=pl.cm.rainbow)
46
47 ax1.set_xlabel(r'X', fontsize=20)
48 ax1.set_ylabel(r'Y', fontsize=20)
49 ax2.set_xlabel(r'X', fontsize=20)
50 pl.show()
```

Code-2-5

运行 code-2-5 得到如图 2.11 所示的结果。其中左图用等高线来表示电势值分布，有图用填充模式的等高图来展示电势分布，其中从红色到蓝色，电势值由高变低。从图 2.11，我们可以很直观地看出，在靠近细杆的位置电势最高，随着远离细杆位置，电势逐渐变弱。

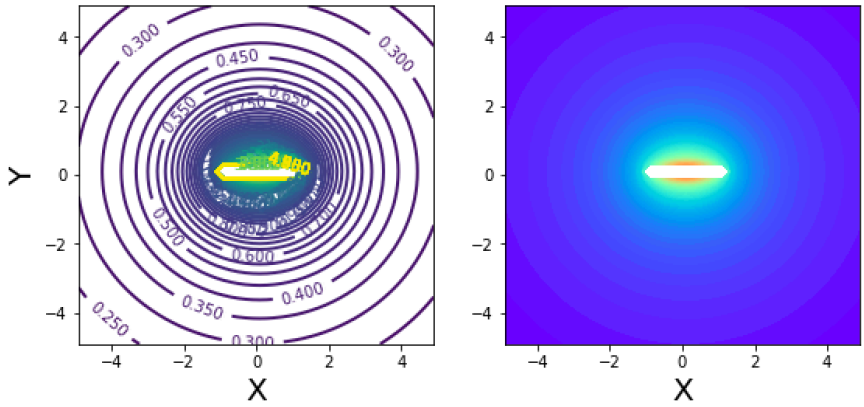

图 2.11. 数值积分得到的电势分布等高线(左)和色条图（由）。

以上矩形公式，当积分步长较大时，会导致较大的误差。为了提高积分计算精度，我们可以用插值函数代替被积函数，进而基于插值函数来计算定积分。

例如，已知$(x_i, f(x_i))$和$(x_{i+1}, f(x_{i+1}))$，我们可以构建线性插值函数：

$$y(x) = \frac{x-x_{i+1}}{x_i-x_{i+1}} \cdot y_i + \frac{x-x_i}{x_{i+1}-x_i} \cdot y_{i+1} \quad (2.19)$$

被积函数$f(x)$在(x_i, x_{i+1})区间与 x 轴围成的面积ΔS_i可以写为：

$$\Delta S_i = \int_{x_i}^{x_{i+1}} y(x)\, dx = \frac{1}{2}(f(x_i) + f(x_{i+1}))\Delta x \quad (2.20)$$

相应地，

$$V = \sum_{i=1}^{n} \Delta S_i = \sum_{i=1}^{n} \frac{1}{2}\big(f(x_i) + f(x_{i+1})\big)\Delta x = \frac{1}{2}f(x_1)\Delta x + f(x_2)\Delta x + \cdots + f(x_n)\Delta x + \frac{1}{2}f(x_{n+1})\Delta x \quad (2.21)$$

式（2.18）即为定积分的**梯形公式**，其截断误差 R 为

$$R = \int_{x_i}^{x_{i+1}} f(x)dx - \frac{\Delta x}{2}(x_i + x_{i+1}) = -\frac{(\Delta x)^3}{12} f''(\xi) \quad (2.21)$$

其中ξ为区间(x_i, x_{i+1})中的一点。

Code-2-6 给出的是用梯形法进行数值积分的代码。与 code-2-5 的只要区别在第 27 行，用梯形积分公式代替矩形积分公式。运行 code-2-6 得到与图 2.11 相同的结果。

```
1 # -*- coding: utf-8 -*-
2 import numpy as np
3 import matplotlib.pyplot as pl
4
5 #--------defining a function-----------
6 def f(x,x0,y0):
7     return np.exp(-x**2)/np.sqrt((x-x0)**2+y0**2)
8
9 N=101  # number of points
10 A=1.0  # half-length of the stem
11 c=1.0  # determine the unit of filed. c= 1/4piepsilon
12
13 xx=np.zeros(N)
14 h = 2.0*A/float(N-1)  # integration step
15
16 for i in np.arange(0,N,1):
17     xx[i] = i*h - A    # coordinate aling the x-axis
18
19 xaxis = np.arange(-5.0,5.0,0.2) # the the (x0.y0) points
20 yaxis = np.arange(-5.0,5.0,0.2)
21 field=[] # list to store the calculated field.
22 for x0 in xaxis:   #calculate the fields for a number of (x0,y0) pionts
23     for y0 in yaxis:
24         v = 0.0
25 #trapezoid integration
26         for i in np.arange(0,N-1,1):
27             v = v + 0.5*(f(xx[i],x0,y0)+f(xx[i+1],x0,y0))*h
28         field.append(v)
29
30 afield = np.array(field) # convert list to array
31 afield.shape = len(xaxis), len(yaxis) #setup the dimension of array
32 afield_xy = afield.T #so that X --> x axis
33
34 extent = [-5.0, 5.0, -5.0, 5.0] # the range of points to be plotted.
35 fig = pl.figure(figsize=(9,4)) # create a figure
36
37 #contour line
38 ax1 =fig.add_subplot(1,2,1) # create a subplot
39 levels = np.arange(0.0,5.0,0.05) # setup the level of field to show.
40 cs = ax1.contour(afield_xy,levels,origin='lower',linewidths=2,extent=extent) # contour line
41 ax1.clabel(cs)
42
43 # color bar
44 ax2=fig.add_subplot(1,2,2)
45 cs = ax2.contourf(afield_xy,levels,origin='lower',extent=extent,cmap=pl.cm.rainbow)
46
47 ax1.set_xlabel(r'X', fontsize=20)
48 ax1.set_ylabel(r'Y', fontsize=20)
49 ax2.set_xlabel(r'X', fontsize=20)
50 pl.show()
```

Code-2-6

类似地，我们可以利用相邻的三个已知点$(x_{i-1},f(x_{i-1})),(x_i,f(x_i))$和$(x_{i+1},f(x_{i+1}))$来构建二次插值函数：

$$y(x)=\frac{(x-x_i)(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})}f(x_{i-1})+\frac{(x-x_{i+1})(x-x_{i-1})}{(x_i-x_{i+1})(x_i-x_{i-1})}f(x_i)+\frac{(x-x_{i-1})(x-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)}f(x_{i+1}) \qquad (2.22)$$

被积函数$f(x)$在(x_{i-1},x_{i+1})区间与 x 轴围成的面积ΔS_i可以近似表达为：

$$\Delta S_i=\int_{x_{i-1}}^{x_{i+1}}y(x)\,dx=\frac{1}{3}(f(x_{i-1})+4f(x_i)+f(x_{i+1}))\Delta x \qquad (2.23)$$

相应地，

$$\boldsymbol{V}=\sum_{i=2,4}^{n}\Delta S_i=\sum_{i=2,4}^{n}\frac{1}{3}\left(f(x_{i-1})+4f(x_i)+f(x_{i+1})\right)\Delta x=\frac{1}{3}f(x_1)\Delta x+\frac{4}{3}f(x_2)\Delta x+\frac{2}{3}f(x_2)\Delta x+\cdots+\frac{2}{3}f(x_{n-1})\Delta x+\frac{4}{3}f(x_n)\Delta x+\frac{1}{3}f(x_{n+1})\Delta x \quad (2.24)$$

式（2.24）即为定积分的**抛物线公式，也称为辛普森公式**，其截断误差 R 为

$$R=\int_{x_{i-1}}^{x_{i+1}}f(x)dx-\frac{\Delta x}{3}(x_{i-1}+4x_i+x_{i+1})=-\frac{(\Delta x)^5}{90}f^{(4)}(\xi) \quad (2.25)$$

Code-2-7 给出的是用抛物线法进行数值积分的代码。其中，与 code-2-5 的主要区别在于第 27 行，用抛物线积分公式代替矩形积分公式将。code-2-7 的运行结果与图 2.11 相同。

```
1  # -*- coding: utf-8 -*-
2  import numpy as np
3  import matplotlib.pyplot as pl
4
5  #--------defining a function-----------
6  def f(x,x0,y0):
7      return np.exp(-x**2)/np.sqrt((x-x0)**2+y0**2)
8
9  N=101  # number of points
10 A=1.0  # half-length of the stem
11 c=1.0  # determine the unit of filed. c= 1/4piepsilon
12
13 xx=np.zeros(N)
14 h = 2.0*A/float(N-1)  # integration step
15
16 for i in np.arange(0,N,1):
17     xx[i] = i*h - A    # coordinate aling the x-axis
18
19 xaxis = np.arange(-5.0,5.0,0.2) # the the (x0.y0) points
20 yaxis = np.arange(-5.0,5.0,0.2)
21 field=[] # list to store the calculated field.
22 for x0 in xaxis:   #calculate the fields for a number of (x0,y0) pionts
23     for y0 in yaxis:
24         v = 0.0
25 #parabola integration
26         for i in np.arange(0,N-1,2):
27             v = v + 1.0/3.0*(f(xx[i],x0,y0)+4.0*f(xx[i],x0,y0)+f(xx[i],x0,y0))*h
28         field.append(v)
29
30 afield = np.array(field) # convert list to array
31 afield.shape = len(xaxis), len(yaxis) #setup the dimension of array
32 afield_xy = afield.T #so that X --> x axis
33
34 extent = [-5.0, 5.0, -5.0, 5.0] # the range of points to be plotted.
35 fig = pl.figure(figsize=(9,4)) # create a figure
36
37 #contour line
38 ax1 =fig.add_subplot(1,2,1) # create a subplot
39 levels = np.arange(0.0,5.0,0.05) # setup the level of field to show.
40 cs = ax1.contour(afield_xy,levels,origin='lower',linewidths=2,extent=extent) # contour line
41 ax1.clabel(cs)
42
43 # color bar
44 ax2=fig.add_subplot(1,2,2)
45 cs = ax2.contourf(afield_xy,levels,origin='lower',extent=extent,cmap=pl.cm.rainbow)
46
47 ax1.set_xlabel(r'X', fontsize=20)
48 ax1.set_ylabel(r'Y', fontsize=20)
49 ax2.set_xlabel(r'X', fontsize=20)
50 pl.show()
```

Code-2-7

以上梯形积分和抛物线积分是常用的数值积分方法。尽管以上例子只涉及一维积分计算，以上方法可以很容易地推广到多重积分问题以及广义积分的计算。值得注意的是，当涉及高维空间的积分问题时，计算效率是需要重点考虑的因素。为了提高计算效率，人们也会采用高斯积分法和蒙特卡洛积分法，在此不多做赘述，有兴趣的同学可以参考专门的“计算方法”教材[1]。

参考文献：

1. Numerical recipes, W. M. Press et al, Cambridge, 3ed. 2007