set\_matplotlib\_formats("svg")

**研究背景**

**锂离子电池简介**

**二维材料简介**

**Python编程语言**

对于涉及量化的科研工作者而言, 学习并掌握一门适合自己领域的程序语言至关重要. 因为程序的运用可极大的简化我们对数据的操作, 甚至影响我们看待问题的方式--能否以一种批量化的方式自动处理琐粹而易错的计算任务? 这亦与我们试图通过某些实验现象归纳实验规律的科研过程是不谋而合的. 现阶段, 中国高校通用的程序教学语言(非计算机专业)是C语言,MATLAB等, 但是C语言是一种'繁琐而令人生畏'的语言, 而MATLAB是一门价格昂贵的商业程序. 除了C语言和MATLAB, 仍有许多程序语言可供我们学习, 如C++, Java, Visual Basic, R等, 但无一例外的是这些语言设计的专业性太强或者并不适合我们的领域.  
Python是一种面向对象的程序语言, 具有简单, 开源, 规范, 可扩展性强, 库丰富等特点, 无论其诞生之初的愿景如何, Python显然已成为各行各业从业人员学习一门程序语言的首选. Table 1是Python在化学或计算化学领域常见的一些项目. 当然Python也有缺点, 如在运行速度方面远远的落后与C语言, 但是一般的用户并不会有明显的感觉, 并且Python的很多模块都用C语言进行了优化(如Numpy).

| **Python工具** | **简介** |
| --- | --- |
| amp | 旨在将机器学习轻松地纳入原子计算是一个开源软件包. |
| ase | 与原子模拟环境环境相关的工具集. |
| chemview | 专为Jupyter Notebook设计的交互式分子查看器. |
| ChemSpiPy | ChemSpider的包装器, 进行化学数据检索. |
| deepchem | 药物发现和量子化学的深度学习模型. |
| mendeleev | 以Python化的方式访问元素周期表中元素的各种属性. |
| phonopy | 采用简谐近似或准简谐近似的声子计算包. |
| pyvib2 | 用于分析振动和振动光谱的程序. |
| PLAMS | 用于自动化分子模拟的Python库. |
| pybel | OpenBabel的Python接口, 进行各种化学文件的相互转换. |
| pymatgen | 分析模拟结果的工具集, 集成了Materials Project的接口. |
| PyMOL | 由Schrödinger用户赞助的开源分子可视化系统. |
| pyscf | 用Python编写的量子化学包. |
| PyQuante | 一个开发量子化学方法的开源程序套件. |
| QUIP | 用于进行分子动力学模拟的软件工具集合. |

figurex.列出了Python在化学或计算化学领域中常见的一些项目, 其更多的项目列表可在GitHub库awesome-python-chemistry中查看[x]. 按用途可大致可以分为以下几类: 普通化学或化学信息学mendeleev, pybel), 分子可视化(ase-gui, PyMOL), 化学模拟(ase,pymatgen), 数据库包装器(ChemSpiPy)等.

Python的简单性可在与C语言的对比中说明. 比如现有一个问题, 计算线性代数中常见的矩阵乘法(AB). 如果要使用C语言实现的话, 可能必须使用循环, 同时遍历矩阵A的行和矩阵B的列, 然后将对应元素求和后放入输出矩阵C中由A的行号和B的列号同时决定的指定位置, 如此循环, 计算完C的所有元素. 更要命的是在C语言的循环中很可能会涉及到指针的操作, 而对于指针的操作, 即时是在计算机专业的初学者看来也是一道难以跨越的门槛. 而如果使用Python的话, 问题变得豁然开朗, Python有太多的库涉及矩阵的这一操作, 早已将其包装成了某个函数(例如Numpy或Scipy中的dot函数), 而我们要所做的仅仅是知道该函数的位置并了解其参数. Python的流形使得我们面临的"任何问题"很可能早已被别人解决, 而Python这个平台则为我们提供了现成的实现方案, 即使某个问题未在这个平台上解决, 我们仍可以从该平台搜寻到用之无尽的"方案原型", 在此基础上实现自己的想法将变得极其简单.

Python的开源性体现无需为任何开源软件支付任何费用, 因为这些软件并不是由商业公司开发, 而是由各个开源社区的人一起创建, 希望每个人都贡献自己的一份力量并服务于所有人. 开源运动起源于Linux的成长和发展, 其目的在于开放源代码, 追求信息的自由和共享. Python开源并不意味着与Python相关的软件都是开源的, 如Python的的集成开发环境PyCharm便是一个商业软件, 这是因为它不仅提供了Python的基本功能, 还提供了非Python的商业公司参入开发的功能, 可以方便代码的调试, 重构,测试等. 作为一般的用户, 我们无需使用商业软件, 因为某种商业软件都有其可替代的开源软件, 甚至某些商业软件也提供了其功能较为单一的开源版本. 开软项目的资金一般来源于开源社区的捐赠.

Python中规范包括强制性规范和非强制性规范, 这与Python的优雅性和简单性是密切相关的. 由于在C语言中代码缩进((语句前的空格数一致)是一种非强制性规范, 因此一段逻辑完全相同的代码在不同的程序员手下, 可以有不同的外观. 但是程序语言是给人看的, 而不是给机器(汇编语言)看的, 所以在Python中, 代码缩进变成了一种强制性规范, 要求逻辑一致的代码段必须具有一致的缩进, 因此可以保证逻辑一致的代码具有相同的外观. Python中最常见的非硬性规范是文档字符串, 文档字符串是包, 模块, 类或函数的第一个语句, 由于注释说明该对象的用途. 一个广为使用的Python模块必定有丰富的文档字符串, 这是我们学习某个模块的第一手资料, 这极大的降低了我们初次接触某个新模块时的困惑.

Python的可扩展性体现在两个方面. 一方面, 如果某段Python代码的运行速度成了整个程序的瓶颈, 此时不必重写整个程序, 只需把瓶颈处的代码换成C语言代码, 然后嵌入原先的Python程序当中即可. 这种Python+C的组合方式融合了Python易于实现以及C便于运行的双方优点. 另一方面, 非Python编写的流形软件都提供了Python访问程序接口(API), 而通过该接口我们不必掌握其他程序的的内部结构便可使用Python获取其解决方案. API可以简单的理解为连接Python和非Python之间的一座桥, 从桥的非Python端看, 该端为Python提供了自己所拥有的资源. 但从桥的Python端看, 可以看到无数的桥, 这使得Python API又好像一个"杠杆", 通过Python获取无数个桥背后的无尽的资源.

Python的库丰富性可从站点PyPI(Python包索引网站)查看, 其包含了与各行各业从业人员相关的近20万个python项目, 并且仍在不断发展当中. Python中关于科学计算和可视化的基础库有Numpy, Scipy, Matplotlib, Sympy, Pandas, Scikit Learn, Jupyter Notebook等. 其中, Numpy提供了多维数组操作和随机数生成的功能; Scipy提供了线性代数计算,统计分析和稀疏矩阵的功能; Matplotlib提供了二维或三维数据可视化功能; Pands提供了类似类Excel的数据表操作等功能; Scikit Learn则是一个关于及其学习的开源项目; Jupyter Notebook则是一个基于各种程序语言内核的文档式网页"开发环境".

再写点关于应用Python的文献简介

**理论基础**

**电子结构理论**

**起源**

自发现电子以来, 电子结构理论已成为认识材料性质的基本理论, 无论这种材料是自然的还是人造的, 无论这种性质是处于基态的还是处于激发态的. 而电子结构理论与量子力学的创立和发展息息相关, 并在诺贝尔奖档案史或科技史上留下了一次次辉煌的印记.

1896年, 洛伦兹用其电子理论解释了塞曼所发现的磁致谱线分裂现象, 该理论认为物质中存在带负电的粒子, 能传导电流且振荡产生光. 1897年, 汤姆逊在气体放电实验中验证了电子的存在. 1900年, 普朗克用能量量子化的概念推导出了黑体辐射的能量密度公式, 标志着量子理论的诞生. 1905年, 爱因斯坦用光量子的概念解释了光电效应, 光量子(光具有粒子性)的真实性于1924年由康普顿的散射实验所验证. 1907年, 爱因斯坦用量子理论解释了原子比热与温度的关系. 1911年, 卢瑟福发现了原子核的存在. 1913年, 玻尔用量子概念解释了原子结构--为何电子和原子核的不会因为吸引而塌缩. 1923年, 德布罗意提出了物质波的概念(即电子具有波动性), 并于1927年由戴维逊和盖末的电子衍射试验所证实. 1925年, 泡利用不相容原理解释了元素周期性. 1923年至1926年期间是是量子力学史上最辉煌的时期, 海森堡以矩阵力学的形式, 薛定谔以波动力学的形式创立了量子力学, 之后狄拉克还创立了相对论性的量子力学.

量子力学创立之后, 人们迅速将其应用于解释分子和固体的电子结构和性质, 逐渐形成了化学键的概念. 例如1927年, 海特勒和伦敦使用量子力学处理了氢分子的电子结构, 指明其分子轨道可通过修改原子轨道的方式创建, 标志着量子化学的诞生. 随着量子力学的发展, 其所能处理的对象不再局限于简单原子和分子, 已逐渐扩展为复杂的体系, 如晶体结构, 其中理解晶体电子结构的关键一步是布洛赫定理, 即晶体中的波函数是晶格周期调幅的平面波. 1933年, 魏格纳和赛茨计算了Na晶体中的能量-距离曲线, 1934年斯莱特计算了金属钠的能带, 其具有近自由电子的特点.

**挑战**

式2.1是薛定谔方程H^Ψ=EΨH^Ψ=EΨ 电子和原子核体系的哈密顿算符, 总共包含五项, 分别是电子动能项, 电子与电子的交互项, 电子与核的交互项, 核的动能项, 核与核之间的动能项. 其中小写字母表示电子相关量, 大写字母表示核相关量., 矢量rr表示粒子在空间中的位置, 不引起歧义的情况下可用标量r表示.

\begin{equation} \begin{split} \displaystyle \hat{H} = &- \frac{\hbar^2}{2m\_e}\sum\_i \nabla^2\_i

* \frac{1}{2}\sum\_{i\neq j}\frac{e^2}{\left|\mathbf{r}\_i-\mathbf{r}\_j\right|}
* \sum\_{i,I}\frac{Z\_Ie^2}{\left|\mathbf{r}\_i-\mathbf{R}\_I\right|}\ &- \frac{\hbar^2}{2M\_I}\sum\_I\nabla\_I^2
* \frac{1}{2}\sum\_{I\neq J}\frac{Z\_IZ\_Je^2}{\left|\mathbf{R}\_I-\mathbf{R}\_J\right|},\,其中 \displaystyle \nabla^2*i = \frac{\partial^2}{\partial^2*{x\_i}}
  + \frac{\partial^2}{\partial^2\_{y\_i}}
  + \frac{\partial^2}{\partial^2\_{z\_i}}. \end{split} \tag{2.1} \end{equation}

由于量子体系中电子与电子交互的复杂性, 使得求解薛定谔方程十分困难, 必须引入很多近似处理, 如非相对论近似, 只考虑核与电子,电子与电子之间的静电相互和不考虑磁效应等; 绝热近似, 将核运动和电子运动分开讨论, 从而将多粒子体系转化为多电子体系; 单电子近似, 忽略电子之间的相关性, 每个电子都在其他电子的平均有效势中运动, 从而将多电子体系转化为单电子体系; 中心力场近似, 假设单电子有效平均势是球对称的, 可以简化计算;

**应用**

电子结构理论可用于用以描述物质的性质解释实验观察到的现象. 物质性质可分为基态性质和激发态性质. 其中基态性质有内聚能, 平衡晶体结构, 结构相变, 弹性常数, 电荷密度, 磁序和磁极化率, 静态介电常数, 核振动等; 激发态性质有涉及如金属比热,电子输送的低能激发和涉及绝缘体能隙,光学特性,添加或去除电子光谱的高能激发.

基态电子密度决定键的类型. 对于稀有气体和分子固体, 其相中原子的电子态与单原子存在时的电子态类似(闭壳层体系), 其能带只有很小的展宽, 这种原子之间的键合一般被描述为范德华键. 电负性相差较大的元素组成离子晶体时, 元素之间会发生电荷转移形成闭壳层离子, 阴离子以密堆积的方式排列, 阳离子位置由阴阳离子之间的库仑力决定, 一般是带隙较大的绝缘体. 在金属中, 没有阻碍电子激发的能隙, 金属阳离子位于"电子海"当中, 其电子密度的分布比较均匀. 对于共价键晶体如金刚石等, 体系中原子的电荷不会发生转移, 而是多个原子之间公用电子对, 尽可能的使多个原子形成闭壳层的稳定结构, 原子之间依靠共价键连接, 由于共价键具有明显的方向性, 所以孤立原子的电子态形成共价原子态时会发生明显的变形. 电子密度在实验上可通过X射线散射测量.

确定晶体的平衡结构的晶胞参数, 确定晶体的相变. 晶胞能量(E), 晶胞体积(ΩΩ), 胞内压力(V)具有如下的关系, B是晶体的体弹性模量:

P=−dEdΩ,B=−ΩdPdΩ=Ωd2EΩ2.(2.2)(2.2)P=−dEdΩ,B=−ΩdPdΩ=Ωd2EΩ2.

根据该关系式, 当晶体处于平衡状态时(平衡体积), 此时晶体的能量最小, 压力为0. 调节晶体的体积时, 并计算该体积对应的能量, 拟合体积和能量之间的关系可得固体的状态方程, 状态方程曲线上能量最低点的体积即是晶体的平衡体积, 根据平衡体积即可求得晶体的晶胞参数, 例如体心立方的结构的晶胞参数是平衡体积的立方根. 晶体体积的调节很可能是在一个非常小的范围内进行的, 如果范围扩大, E(ΩΩ)曲线可能并不是只有一个极值点, 每个极值点都对应一种可稳定或亚稳定存在的晶体结构, 因此研究能量-体积曲线可研究晶体的相转变(在压强改变下).

可将电子理论的核心输出量是能量, 能量是所有性质或实验现象的根源. 与体弹性模量类似, 材料的大多物理性质都可以直接或间接的通过对E的求导获得. 除基本的物理性质外, 电子理论还可以判断材料的类型(金属,半导体,绝缘体), 可以研究原子扩散, 催化剂的反应机理, 研究固体表面, 界面, 缺陷等; 当然也包括涉及电子激发的热容, 传导和光学谱等.

**密度泛函理论**

**简介**

将多电子体系转化为单电子体系的第一性原理计算大致有两类: Hartree-Fock方法(HF方法)和密度泛函理论(DFT). HF方法主要应用于分子体系, 首先不考虑电子相关性, 将使用原子轨道波函数叠加处理成单电子的波函数, 然后使用多个Slater行列式的线性组合, 在这些单电子波函数中引入电子相关性. HF方法只能处理原子数较少的体系, 而对于晶体等复杂体系需要使用DFT方法. DFT方法由Hohenberg, Kohn和Sham于1964提出, 经过五十多年的发展, 已成为物质电子结构计算和性质计算最有利的工具. 传统的量子力学计算以电子波函数作为基本变量, 而在DFT方法中以电子密度作为基本变量, 在求解薛定谔方程的过程中, 采用合适的交换关联能近似, 可以将哈密顿量完全转化为电子密度的函数而不必关注电子相关的复杂性, 可以大大简化计算的复杂度.

量子力学的变分原理表明, 在薛定谔方程无法求解或求解十分困难的情况下, 可使用变分原理逼近体系基态的能量. 首先能量可以表示成波函数的泛函形式, E[Ψ]E[Ψ], 然后给出一些列波函数的初始猜测值, 这些初始猜测值所对应的能量必定大于或等于体系基态的能量, 因此当初始波函数的个数很大的情况下, 可用其所得最小能量近似表示真实体系的基态能量, 即求泛函的极小值. 相关理论表明, 该极值问题的复杂度与三倍粒子数的幂次相关O(e3N)O(e3N), 即随着体系的增大, 该泛函的极值将变得无法求解. 1964年Hohenberg和Kohn基于非均匀电子气理论提出两条定理, 其大意是, 基态能量可以表示成电子密度的泛函即E[ρ]E[ρ], 并且在粒子数保持不变的条件下, 该泛函的极小值等于体系处于基态时的能量, 其中体系的外势是也可由电子密度表示. 当能量由波函数的泛函表示成电子密度的泛函后, 其极值的求解复杂度不再与粒子数有关.

**Kohn-Sham方程**

一个体系的能量由外场和内场两部分组成, 如式[]表示:

E[ρ]=Vext[ρ]+F[ρ]=Vext[ρ]+T[ρ]+V[ρ](2.3)(2.3)E[ρ]=Vext[ρ]+F[ρ]=Vext[ρ]+T[ρ]+V[ρ]

内场是指电子体系内部的场, 包括电子的动能以及电子与电子之间的相互作用, 外场是电子体系外部所受的场, 如核对电子的势场.  
引入单电子波函数的基组 ϕi(r),i=1,2,⋯ϕi(r),i=1,2,⋯, 则电子密度可表示为式:

ρ(r)=∑i=1N|ϕi(r)|2(2.4)(2.4)ρ(r)=∑i=1N|ϕi(r)|2

动能项, 电子-电子项, 外场项均可表示为电子密度的泛函:

Vext[ρ]T[ρ]V[ρ]=∫ν(r)ρ(r)dr=∑i=1N∫drϕ∗i(r)[−∇2]ϕi(r)=12∬drdr′ρ(r)ρ(r′)|r−r′|(2.5)(2.5)Vext[ρ]=∫ν(r)ρ(r)drT[ρ]=∑i=1N∫drϕi∗(r)[−∇2]ϕi(r)V[ρ]=12∬drdr′ρ(r)ρ(r′)|r−r′|

在能量泛函中加入一个未知形式的交换关联泛函 EXC[ρ]EXC[ρ], 用于修正泛函2.3与实际系统总能之间的误差(也即: 将所有不可实际求解的项概入其中), 其经验形式可通过实际体系的精确解进行拟合而来. 最后的总能泛函可表示为:

E[ρ]=∫ν(r)ρ(r)dr+∑i=1N∫drϕ∗i(r)[−∇2]ϕi(r)+12∬drdr′ρ(r)ρ(r′)|r−r′|+EXC[ρ](2.6)(2.6)E[ρ]=∫ν(r)ρ(r)dr+∑i=1N∫drϕi∗(r)[−∇2]ϕi(r)+12∬drdr′ρ(r)ρ(r′)|r−r′|+EXC[ρ]

对式2.6进行变分处理 δE(ρ)δρ=0δE(ρ)δρ=0, 得Kohn-Sham方程:

[−∇2+VKS[ρ(r)]]ϕi(r)=Eiϕi(r),其中VKS[ρ(r)]=ν(r)+∫dr′ρ(r′)|r−r′|+δEXC[ρ]δρ(r)(2.7)(2.7)[−∇2+VKS[ρ(r)]]ϕi(r)=Eiϕi(r),其中VKS[ρ(r)]=ν(r)+∫dr′ρ(r′)|r−r′|+δEXC[ρ]δρ(r)

交换关联泛函的没有精确的表示, 最常见的是两种近似形式是局域密度近似(LDA)和广义梯度近似(GGA), 可通过均匀电子气的交换关联能密度的相关积分计算. 其中LDA只与空间元密度的密度值有关, 而GGA还与空间元之间的密度差有关, 如式2.8所示:

ELDAXC=∫ρ(r)ϵXC(ρ(r))drEGGAXC=∫ρ(r)ϵXC(ρ(r),∇ρ(r))dr(2.8)(2.8)EXCLDA=∫ρ(r)ϵXC(ρ(r))drEXCGGA=∫ρ(r)ϵXC(ρ(r),∇ρ(r))dr

**KS方程的求解**

KS方程可通过自洽的方式求解. 式2.4表明, 在求解KS方程之前必须使用一组初始的基函数, 由这些基函数确定初始的电荷密度. 由电荷密度确定KS方程中的各种势(式2.5), 从而确定KS方程的哈密顿量, 解哈密顿量的本征方程式2.7, 即可确定一组非收敛的本征波函数(以及本征能量). 将该组本征函数重新作为输入, 继续求解式2.7直到体系能量收敛. 由收敛后波函数和电荷密度即可确定该体系基态的总能以及其他物理量算符的期待值.  
虽然KS方程的自洽过程虽然简单, 但是KS方程的实际求解过程却十分复杂, 此处只简要概述一些需要特别注意的关键步骤. 交换关联泛函的选取. 对于一般的氢原子体系和很多金属, GGA泛函相比于比LDA泛函可以给出更精确的解, 所以一般使用GGA泛函即可, 如PW91和PBE泛函. 对于一些要求更加精确的计算, 亦可使用杂化泛化, 但是要十分注意精度与时间消耗的权衡.  
基组, 赝势, 计算方案. 基函数的选取不是随意的, 必须保证其在迭代的过程中不会发散, 收敛迅速, 并且在保证精度的前提下其使用数目必须尽可能的小等, 从而可以使用最小的计算量拟合最终的KS轨道. 最简单的基函数是平面波函数(自由电子气的本征函数)和原子轨道. 平面波只适合描述价电子的波函数, 而不适合描述内层电子的波函数(无法描述内层电子波函数的震荡性), 所以晶体中的基函数并不能使用单一类型的平面波函数, 而必须构造一种同时适合内外电子的赝波函数. 赝波函数可通过以确定的赝势与KS方程来计算, 赝势可简单的认为是离子实对内外电子的势. 根据离子势和基函数构造的不同, DFT计算方案可分为LCAO(原子轨道线性组合), PP-PW(赝势平面波), LMTO(线性MT轨道), LAPW(线性缀加平面波)等.

**python的基础知识**

Python中需要注意的基本概念有: 对象, 模块, 包, 类, 方法, 属性, 特性等. Python是一门面向对象的程序设计语言, 可以将对象简单的认为是Python中所有有名字的变量, 是计算机内存中占有的一段数据, 不管是模块,类,函数, 还是普通的变量都可以当做是对象. 对象有时专指类的实例化对象, 而类是对现实生活中一类具有共同特征的事物的抽象, 类为其对象定义一系列与其相关的其描述性信息, 即属性或特性(可通过赋值进行修改), 以及与其相关的操作, 即方法. 包是包含源代码的总目录, 而模块是包中的文件.

模块一般定义了实现某种功能的类和常量, 以及对这些功能的说明文档即文档字符串, 可以说模块是是底层代码和文档字符串的集合. 文档字符串是我们学习Python某个函数的第一手资料, 很多解释器都可以自动搜集文档字符串, 如如IPython可使用对象?或??查看. 还可以使用内嵌函数dir()查看对象的属性和方法列表, dir2(自定义函数)实现带通配符的索引, 配合魔法命令 %col 可以将其输出按列显示, 见附录Code22.

In [2]:

%%**disabled**

# 查看Python的内置函数

import builtins

%col 2 dir2(builtins)

Python的内置函数和数据类型. 附录Code xx 可查看所有的内置函数: 有些内置函数是关于数学运算的如sum,max等; 有些是关于输入输出的如input, print等; 有些是定义数据类型或数据结构的, 如str(字符串), list(列表), tuple(元组), dict(字典); 还有一些可用于Python自身的反射操作(搜集对象的文档字符串, 检查变量的类型等), 如help, dir等; 除了基本的整数,实数,字符串类型外, Python的内置类型还有列表, 字典等. 列表是任意对象的序列, 而字典是键值对序列, 列表和字典都可以进行迭代和索引. Python中的循环语句(可迭代序列中元素)和条件语句与其他语言类似, 只不过要特别注意缩进的问题即可. 除了标准的内置类型, 其他的模块也定了一些自己的数据类型, 如numpy定义了数组, Panda定义了Dataframe等.

更多的说明(如序列解包, 列表生成式, 生成器等)可以在任何一本Python教材中查看. 详细研讨一本Python教材, 对Python语言框架式的认识是有必要的, 但是一次性的掌握Python的所有细节却既不现实又没有必要. 因此更明智的学习方式是遇到问题之后再去详细的学习. 而关于一个函数或者类的学习, 可能根本不需要花费时间去查找其用途, 应为只需在IPython解释器中在其名称之后键入问号即可. 可以很方便的查看某个函数的源码, 参数及其说明, 有的还会给出示例.

下图显示了Python的组织结构, 使用dot程序绘制[]. 把Python放置在根节点, 其下的第一层是其可扩展的包, 可在PyPI等渠道下载. 简要显示ASE包内的层次, 目录calculators是ASE内有关计算器的子包, atoms.py是 ASE内的一个模块, 其中定义了类Atoms以及类实例所拥有的方法set\_\*()和get\_\*().

In [3]:

dot2graph("data/python\_tree.dot")

Out[3]:

模块包PythonNumpyMatplotlibAsePymatgen...应用程序APIcalculatorsatoms.pyabinit.pydmol.pyvaspAtomsset\_\*()get\_\*()

学习Python的第一步是Python解释器或IDE(集成式开发环境)的选择. Python解释器是按行"翻译"Python语句的程序, Python IDE是集成了解释器,编辑器,调试器等与编程相关功能的程序集, 以便为项目开发提供一个便捷和舒适的环境. Python的IDE种类丰富各有特色, 其中最常见的是有: Python官网提供的IDLE, 轻便但功能较为单一; 可通过U盘在Windows系统之间移植的WinPython, 只能在Windows下使用; 功能十分齐全但是专业版价格较贵的PyChem; 开源, 且在科研领域十分流行的Anaconda; 本文将选择Anaconda为主要的Python开发环境, 因为其内置的众多包可以免去在孤立系统中安装时的繁琐(依赖项), 并且其内置了Ipython解释器和Jupyter Notebook, 以及类似于Matlab的程序界面Spyder.

Ipython(Interactive Python)主要功能是具有交互式增强的Shell以及可与Jupyter Notebook进行耦合. 交互式增强Shell的主要特点有: TAB对象补全; 动态对象内省, 即可以在任何对象的后面键入?或?? 查看文档字符串, 函数定义原型, 源代码等; 系统命令调用, 即通过在系统命令前键入!的方式可以在Python解释器中直接调用系统命令; 可扩展的Magic方法, 内嵌的行命令以%开头, 内嵌的单元命令以%%开头;  
Jupyter Notebook是一种基于网页的文档编辑工具, 类似于Word但比其更加简单和便捷. 如果将Jupyter和ipython耦合, 那么在编写文档中程序代码的同时可以使其运行并将其结果嵌入到文档当中. 启动Jupyter Notebook的服务器之后, 会自动在浏览器内打开一个页面(其默认地址为<http://127.0.0.1:8888>), 可以在该页面内创建目录和文件, 实则对应的是本机中的目录. 每个文档都有一些竖向排列的单元格构成, 每个单元格可以有两种格式, 普通Markdown文本以及可运行的Python代码, 如[图] Markdown是一种简化的格式化文本工具, 基于Markdown的Jupyter可以便捷的在文档内输入公式, 图表, 甚至是视频和声音等. Ipython相当于Jupyter的运算核, 进行代码的运行, Jupyter可以配置任意语言的运算核, 例如MATLAB和C等. 不是自己写的函数放在ibr包中, 直接调用

**材料模拟软件简介**

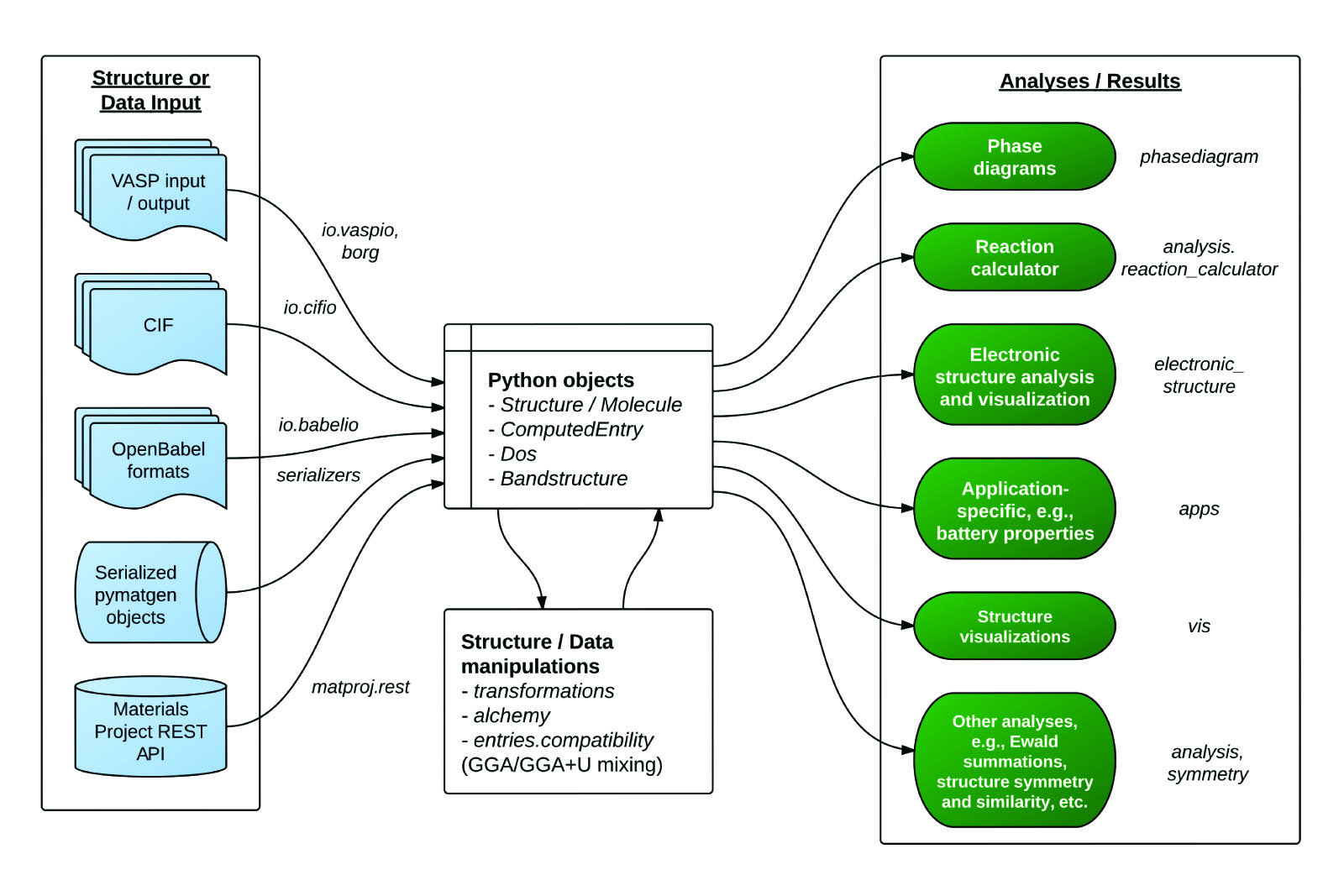
**Material Studio 和 Vasp**

Material Studio是一款使用门槛较低的材料计算软件, 功能十分丰富, 可以进行各种尺度的材料模拟问题. Material studio采用客户端/服务器结构, 客户端是一个图形用户界, 主要由核心模块Visualizer组成, 用于搭建晶体结构, 设置各模块运算参数, 以及分析运算结果, 服务器运行在集群之上, 主要执行计算密集型的任务.  
Vasp是一个单纯的使用赝势平面波方法的从头量子力学分子动力学的软件包. 每个Vasp任务都必须包含几个输入文件, INCAR设置计算参数, KPOINTS设置k空间的网格划分, POSCAR是初始的晶体结构, POTCAR文件包含了必要的赝势信息. Vasp是一款完全建立在集群之上的计算程序, 因此可以借助Linux系统强大的系统管理命令简化其一些常规操作, 比如附录Code--kk是一个可一次性检查Vasp输入文件, 判断计算是否收敛并获取能量的一个小脚本.

**Python包**

字数不够的话再加上Numpy, sympy等的介绍, 使用到的函数等

表显示了各种化学用途Python包, 此处只介绍本文中使用到的几个软件包--普通化学包mendeleev和 ChemSpiPy, 化学模拟包ase和pymatgen.  
mendeleev(门捷列夫) [ck] 和 ChemSpiPy 是两个普通化学Python, 可以分别提供元素性质访问和分子数据库查询的功能. 使用mendeleev可用于访问元素周期表中元素(或离子,同位素)的各种属性, 亦可可可视化元素周期表的功能. chemspider是一个化学信息学在线数据库, 可进行化学数据检索和化学文件下载, ChemSpiPy则提供了一种在Python中与ChemSpider交互的方法.  
. ASE(Atomic Simulation Environment)是一组Python模块, 可用Python化的方式于设置, 操作, 运行常用的模拟软件(能量计算器), 并且可以分析和可视化模拟软件的输入和输出结果. 例如模块build提供了使用Python语句搭建晶体结构的可能, caculators模块提供了使用Python语句操纵Vasp任务的可能. Pymatgen (Python Materials Genomics) 是一个用于材料分析的强大的开源库, 可用于生成相图, 进行扩散分析, 电子结构分析等等, 甚至提供了 Materials Project数据库的API, 下图是pymatgen的典型工作流. 提供API功能的Python包可能会受制于某些站点或数据库的要求, 需要用户自行申请一个API-key, 用于控制数据库的访问权限.



**python在科研工作中的应用**

**引言**

Python的提供了一个良好的平台, 可以用于处理日常工作和学习当中遇到的问题.  
用Python处理理论学习当中遇到的数学基础. 在量子化学的基础理论当中, 常常涉及大量复杂的数学问题, 虽然我们应该把重点放在理论本身, 但是如果完全忽略理论的数学基础, 单从感性的角度又往往很难对理论有一个深刻的认识, 所以数学基础在理论的学习当中是不应该完全忽视的. 在不可忽略的数学基础当中, 我们通常把重点放在数学原理的理解上, 而不是将时间浪费在某些实质性的数学运算. 对于那些浪费时间的数学计算, 我们更希望通过程序而不是手工计算的方式应对, 好在Python中有很多优秀的数学模块, 如numpy,sympy,scipy等, 可以帮助我们在认识数学原理的过程中自动执行其中繁琐的计算, 并且有许多优秀的可视化模块, 如matplotlib等, 可以形象的展示其中的数学图像. 在3.2节将以类氢离子波函数的图像为例展示Python的这一用途.  
将Jupyter Notebook-Python作为一个文献翻译平台. 对于英语水平不高的同学来说, 英文文献的阅读和翻译可能是读研期间面临的一个不小的问题, 尤其是在大学期间完全没有上过英文专业课的同学来说, 因此文献翻译工具(如有道)是手头必备工具之一. 对于某些很难理解的文献我们需要不断的理解和修改, 因此我们需要使用文档工具(如Word)保存前几次翻译的译文, 否则遗忘之后可能需要重头翻译. 同时使用翻译软件和文档工具需要在两者之间频繁的转换, 显得十分啰嗦, Jupyter Notebook作为一个文档式的Python环境, 既可以当做普通文档工具, 又可以在其中执行Python代码), 我们我们能够在Jupyter Notebook调用中使用Python调用某款翻译软件的API, 如[有道翻译API](https://ai.youdao.com/docs/doc-trans-api.s#p10) 则可实现一个简易的一体式翻译-保存工具.  
使用Python API访问某些数据库. API(Application Programming Interface)是应用访问接口的简称, 它屏蔽目标应用程序内部的底层细节，从而当其他程序调用此程序以获取其数据时将变得非常便捷. 如前面所述的有道翻译是一个目标应用程序, 我们希望使用它完成翻译工作, 有道翻译的Python API相当于一个连接有道翻译的通道, Python API既向道翻译软件提供输入, 又获取其输出, 因此可简单的认为Python API就是Python版本的有道翻译. 在本文中我们还会用到mendeleeve以获取元素周期表中的元素性质, 使用Pymatgen中的MPRester访问Material Project中的计算数据. 在3.2中我们将使用Python来展示元素的周期性质.  
将Python作为数据分析与处理的工具. 对于一般的数据分析通常涉及两个问题, 一个是查看离散点的分布, 另一个是通过拟合确定出模型中的参数. 离散点的分布可通过模块Matplotlib查看, 数据拟合可使用Scipy中的函数处理. 最小二乘法是最一般的拟合方式, 相当于求解线性代数中的超定方程组, 超定方程组中方程的个数小于未知数的个数. 在本文中, 我们将使用最小二乘法拟合多项式类型的晶体状态方程, 以确定MoS的晶胞参数.

**结果和讨论**

**类氢离子波函数的图形演示**

类氢离子波函数的详细的求解过程可在量子化学教材中查看, 可表示为式3.1的形式.

Ψnlm(r,θ,ϕ)=RnlΘlm(θ)Φm(ϕ)(3.1)(3.1)Ψnlm(r,θ,ϕ)=RnlΘlm(θ)Φm(ϕ)

其各部分的详细表示如式3.2所示, 其中 PP 表示连带勒让德多项式, LL 表示连带拉盖尔多项式.

ΦmΘlmRnl=12π−−√eimϕ=(−1)m2–√2(l−|m|)!(l+|m|)!(2l+1)−−−−−−−−−−−−−−√P(|m|)l(cos(θ))=2(2Zra0n)lZ3(−l+n−1)!a30n4(l+n)!−−−−−−−−−−−−−−√e−Zra0nL(2l+1)−l+n−1(2Zra0n)(3.2)(3.2)Φm=12πeimϕΘlm=(−1)m22(l−|m|)!(l+|m|)!(2l+1)Pl(|m|)(cos⁡(θ))Rnl=2(2Zra0n)lZ3(−l+n−1)!a03n4(l+n)!e−Zra0nL−l+n−1(2l+1)(2Zra0n)

从 ΦmΦm 可以看出波函数是一个复函数, 根据波函数的叠加原理可以将其表示成实函数, 如式 3.3所示

Φm⇒⎧⎩⎨⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪sin(|m|ϕ)π−−√12π−−√cos(|m|ϕ)π−−√m<0m=0m>0(3.3)(3.3)Φm⇒{sin⁡(|m|ϕ)πm<012πm=0cos⁡(|m|ϕ)πm>0

由于波函数是一个三元函数, 无法在三维空间中作图, 所以一般情况下的波函数都做了降维处理, 如只观察波函数的径向部分(径向分布图), 或只观察波函数的角度部分(角度分布图), 或者将波函数中的z坐标设置为常数(波函数-变量对图), 或者使用界面图和电子云图等. 本文我们只讨论两种简单的形式, 即波函数的径向分布图和电子云的角度分布图. 类氢离子波函数的径向分布图如 fig3.1 曲线 R2r2−rR2r2−r 所示, 表示在某个与角度无关的向径处电子存在的几率, Code3.1是其绘制程序. 类氢离子电子云的角度分布图是 Θ2Φ2Θ2Φ2 与 θ,ϕθ,ϕ之间的关系曲面, 将其转化为直角坐标系如图 fig3.2所示, 对应Code3.2.  
Code3.1主要有两部分组成, 第一部分是一个函数, 定义了Rnl与量子数之间的函数关系, 此处我们使用了Sympy模块直接将公式3.2中的Rnl写成符号表达式即可, 此处我们使用原子单位, 即ZZ和a0a0 都为1, 最后函数返回为可使用Numpy模块进行离散处理的普通表达式函数. 第二部分为绘图部分, 我们将所有量子数n小于等于3条件下的曲线绘制在了同一图中, 便于对比分析. Code3.2与Code3.1类似, 需要注意的是在其绘图部分使用了一个函数sph2cart(),便于将球坐标中的离散点转化为直角坐标系下的离散点, 函数sph2cart()的具体形式可根据求坐标和直角坐标的关系自行写出.

In [4]:

%%**disabled**

# 径向分布图

plt.rc('lines', lw=2.5, mew=5)

plt.rc('font', size=16)

plt.rc('axes', linewidth=1.2)

def Rnl(R\_n=2, R\_l=1):

from sympy import var,exp,assoc\_laguerre,factorial,sqrt,lambdify

var("a0,n,l,r,rho,Z")

N = -sqrt( (2\*Z/n/a0)\*\*3 \* factorial(n-l-1) / (2\*n\*factorial(n+l)))

L = -assoc\_laguerre(n-l-1,2\*l+1,rho)

Rnl\_ = N \* L \* exp(-rho/2) \* rho\*\*l

Rnl = Rnl\_.subs({rho:2\*Z\*r/n/a0}).subs({Z:1,a0:1,n:R\_n,l:R\_l})

return lambdify([r],Rnl,modules="numpy")

plt.figure(figsize=(8,4))

aur = np.linspace(0,25,1000)

for i in range(1,4):

for j in range(0,i):

Dr = (aur\*Rnl(i,j)(aur))\*\*2

plt.plot(aur,Dr,label="{}{}".format(i,'spd'[j]))

plt.xlim(0,25)

plt.ylim(0,0.6)

plt.xlabel('r/a0')

plt.ylabel('D(r)')

plt.legend(ncol=2)

plt.savefig(fname='data/drr.png',format='png',bbox\_inches='tight')

In [5]:

*# %%disabled*

plt.rcdefaults()

**def** Ylm(lx=0,mx=0):

**from** **sympy** **import** var,pi,sqrt,exp,sin,cos,I,factorial,assoc\_legendre,lambdify

var("l,m,theta,phi")

*# Theta的归一化系数*

N = sqrt( (2\*l+1)\*factorial(l-abs(m)) / (2\*factorial(l+abs(m))) )

P = (-1)\*\*m \* assoc\_legendre(l,abs(m),cos(theta))

*# 将复波函数转化为实波函数*

**if** mx==0: Phi = 1/sqrt(2\*pi)

**elif** mx<0: Phi = sin(abs(m)\*phi)/sqrt(pi)

**else**: Phi = cos(abs(m)\*phi)/sqrt(pi)

Ylm\_ = N \* P \* Phi

Ylm = Ylm\_.subs({l:lx,m:mx})

**return** lambdify([theta,phi],Ylm,'numpy')

**from** **mpl\_toolkits.mplot3d** **import** Axes3D

num=3

fig = plt.figure(figsize=(4\*num,4\*num))

t,p = np.mgrid[-np.pi:np.pi:200j,-np.pi:np.pi:200j]

**for** i **in** range(num):

**for** j **in** range(i+1):

ax = fig.add\_subplot(num,num,num\*i+j+1, projection='3d')

Y = Ylm(i,j)(t,p)

ax.plot\_surface( \*sph2cart(Y\*Y,t,p), rstride=2, cstride=2, cmap=plt.cm.jet, lw=2)

plt.xlabel('x')

plt.ylabel('y')

ax.set\_title("l=**{}**,m=**{}**'".format(i,j), fontdict={'fontsize':20})

print("**{}{}**'".format(i,j),end=';')

fig.savefig(fname='data/ylm.png',dpi=160, format='png')

00';10';11';20';21';22';

In [67]:

%%**disabled** 在另一个页面就可以用

# 使用OpenCV批量剪裁照片--原始照片尺寸最好相同

import cv2

names = ['2s0.BMP', '2p0.BMP',

'3s0.BMP', '3p0.BMP', '3d0.BMP', '3d1.BMP']

for name in names:

img = cv2.imread(name)

cropped = img[0:174, 210:390]

#裁剪坐标为[y0:y1, x0:x1], # img.shape配合画图工具查看像素值位置

cv2.imwrite("clip/"+name, cropped)

%cd Y:\Git\ldr\Doing\data\OV

import cv2

def show\_images(imgs,titles):

fig=plt.figure(figsize=(12,2))

for idx, img in enumerate(imgs):

ax = fig.add\_subplot(1,len(imgs),idx+1)

plt.imshow(img)

plt.title(titles[idx],fontdict={'fontsize':16,'weight':'bold'})

ax.set\_axis\_off()

plt.subplots\_adjust(0.02,0,0.98,1,0.02,0)

plt.savefig('Psi.png', bbox\_inches='tight')

names = ['clip/2s0.BMP', 'clip/2p0.BMP',

'clip/3s0.BMP', 'clip/3p0.BMP', 'clip/3d0.BMP', 'clip/3d1.BMP']

titles=['$\Psi\_{2s0}$', '$\Psi\_{2p0}$',

'$\Psi\_{3s0}$', '$\Psi\_{3p0}$', '$\Psi\_{2d0}$', '$\Psi\_{3d1}$']

# imgs = [cv2.imread(name) for name in names]

img0 = cv2.imread('clip/2s0.BMP')

img1 = cv2.imread('clip/2p0.BMP')

img2 = cv2.imread('clip/3s0.BMP')

img3 = cv2.imread('clip/3p0.BMP')

img4 = cv2.imread('clip/3d0.BMP')

img5 = cv2.imread('clip/3d1.BMP')

imgs = [img0, img1, img2, img3, img4, img5]

show\_images(imgs=imgs,titles=titles)

%cd Y:\Git\ldr\Doing\

根据图3.1和3.2, 我们讨论下波函数的节点个数问题, 节面是波函数的值为0的面. 由图3.1可以看出, 径向中波函数中极大值峰的个数为(n-l), 径向节面(球面)的个数为(n-l-1), 由图3.2, 可以看出角节面的个数等于量子数l, 所以总波函数中节点的个数应该是(n-1)个, 这可以在波函数界面图或者轮廓图中查看. 由于使用Python绘制轮廓图稍微复杂些, 此处我们使用一款轨道查看器软件(Orbital Viewer)绘制几个常用的轮廓图(原子轨道), 如图3.3所示(忽略坐标轴), 波函数Ψ2s0Ψ2s0, Ψ3s0Ψ3s0中l=0, 所以在图中没有角节面, 波函数中的节面全都是径向节面, 节面的个数为(n-l); 波函数Ψ2p0Ψ2p0, Ψ3d0Ψ3d0(锥面), Ψ3d1Ψ3d1,径向节面数(n-l-1)为0, 波函数的节面全都是角节面. 对于波函数Ψ3p0Ψ3p0, 径向节面(n-l-1)的个数为1, 角节面的个数为1.

**调用化学数据库**

材料学数据库  
API  
程序流程  
程序结果

In [69]:

%%**disabled** 当做一个Code,但不在正讲了

from chemspipy import ChemSpider

cs = ChemSpider('FF37OwRwIAJYux1vZicusKGG4iGnV9xw')

# 根据名称查找序号

# list(cs.search('adderall')) => 13852819

a = cs.get\_compound(13852819)

# 待查化合物的信息

%C 5 a.common\_name; a.molecular\_formula; a.molecular\_weight; a.smiles

#化合物的结构

a

图[]和图四是从mendeleev获取数据之后使用Matplotlib绘制的图, 图[]原子第一电离能与原子序数之间的关系, 可以看出电离能具有明显的周期性. 图[]是 是过渡金属的熔点对比图, 每个周期的金属熔点的分布形状类似..

In [16]:

%%**disabled** 元素周期表

plt.rc('lines', lw=2.5, mew=5)

plt.rc('font', size=14)

plt.rc('axes', linewidth=1.2)

from mendeleev import element

fig = plt.figure(figsize=(14,4))

# 原子第一电离能与原子序数的关系

ax1 = fig.add\_subplot(1,2,1)

energies = [np.log10(element(int(i)).ionenergies[1]) for i in np.arange(1,100)]

atomic\_number = [element(int(i)).atomic\_number for i in np.arange(1,100)]

plt.plot(atomic\_number,energies,':o',lw=2)

cols = [1,2,3,10,11,18,19,36,37,54,55,86,87]

for i in cols:

plt.text(i+0.2,energies[i-1],element(int(i)).symbol)

plt.xlabel("Atomic Number")

plt.ylabel("$log(I\_1$) / eV")

# 第四, 第五, 第六周期金属的熔点分布

ax2 = fig.add\_subplot(1,2,2)

raw4 = np.arange(19,31)

raw5 = np.arange(37,49)

raw6 = [55,56,57,72,73,74,75,76,77,78,79,80]

for raw in [raw4, raw5, raw6]:

melting\_point = [element(int(i)).melting\_point for i in raw]

groups = [element(int(i)).group\_id for i in raw]

plt.plot(groups,melting\_point,':o',lw=2,label='Raw{}'.format(raw4))

symbols = [element(int(i)).symbol for i in raw]

for idx in np.arange(4,8):

plt.text(groups[idx]+0.1,melting\_point[idx]+10,symbols[idx])

plt.xlabel('Group ID')

plt.ylabel('Melting Point / K')

plt.legend(["4","5","6"])

plt.xlim(0,13)

fig.savefig('data/ptable.png',bbox\_inches='tight')

In [ ]:

plt.legend(["1","2","3"])

In [46]:

raw4 = np.arange(19,31)

str(raw4)

Out[46]:

'[19 20 21 22 23 24 25 26 27 28 29 30]'

**计算结构的晶格常数**

In [2]:

%%**disabled**

# 使用ase调用vasp, 计算9个原胞的能量

a0, c0 = 3.183, 3.127

traj = Trajectory('MoS2\_2H.traj', 'w')

eps = 0.01

for a in a0 \* np.linspace(1 - eps, 1 + eps, 3):

for c in c0 \* np.linspace(1 - eps, 1 + eps, 3):

MoS2\_2H = mx2('MoS2', kind='2H', a=a, thickness=c, vacuum=7.5)

calc = Vasp2(directory='mos2\_ltc', xc='PBE', gamma=True, kpts=(12, 12, 1))

calc.set(prec='Accurate', encut=400, algo='fast', ibrion=2, ediffg=-0.02,

ismear=0, sigma=0.05, ispin=2)

MoS2\_2H.set\_calculator(calc)

MoS2\_2H.get\_potential\_energy()

traj.write(MoS2\_2H)

# 求解E(a,c)的系数

from ase.io import read

configs = read('MoS2\_2H.traj@:')

E = np.array([config.get\_potential\_energy() for config in configs])

a = np.array([config.cell[0, 0] for config in configs])

c = np.array([config.cell[2, 2]-15 for config in configs]) # 实际高度为晶格高度-真空层的高度

M = np.c\_[a\*\*0, a, c, a\*\*2, a \* c, c\*\*2] # Mp=E

# M = np.array([a\*\*0, a, c, a\*\*2, a\*c, c\*\*2]).T

p = sp.linalg.lstsq(M, E)[0]

# 求二元函数E(a,c)的极值, A是其一阶偏导

A = np.array([[2\*p[3], p[4]],

[p[4], 2\*p[5]]])

beta = -np.array([p[1],p[2]])

# a0是原胞的底边, c0是原胞的高

a0, c0 = np.linalg.solve(A,beta)

# k=3, prec=默认, a0: 3.191, c0: 3.106; k=12, prec='accu': a0: 3.184, c0: 3.128.

print("a0: {:.3f}, c0: {:.3f}.".format(a0,c0))

**结论**

结论中提一下, 批量操作--结构, 图形拼接, 波函数的对图等等

**机器学习**

**或者Li-S电池**

**Li离子在 TMDs上的吸附**

**总结和展望**

In [183]:

%**kk** 2

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$!kk2:end>$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$