Vasp定压计算脚本vaspeqstress.sh使用教程

跟QE不一样，VASP目前比较尴尬的地方在于只能对体系施加静水压，而没办法施加一些我们希望的外压。前不久偶然看到[小侯飞氘][https://zhuanlan.zhihu.com/p/32862436]在2018年用python2写了一个VASP定压脚本，但是有同学好像反映试了以后出现了bug。闲来无事，我对大佬的代码进行了修改优化，用python3重写成vaspeqstress.py。本着bash能解决的事情就不用python，模块能少用就少用的原则，强迫症发狂的我于是用bash重写成vaspeqstress.sh。好吧，前面的是胡扯，主要是考虑到工作站或者服务器并不一般会给学生部署python3及其模块，加上bash本身更适合直接操作文件，更适合与作业管理系统配合，我就喜欢这种直截了当的style。

说了一堆废话，下面我主要先讲一下原理，因为一些参数的设置需要你先了解原理。原理很简单，其实就是广义胡克定律。首先，我们从VASP输出的OUTCAR文件获得当前体系的所受外压矩阵，通过跟我们的目标外压矩阵相减，就获得了下一步需要施加的外压矩阵M_{ad}，通过广义胡克定律，我们可以将M_{ad}中的每个应力组元转换为应变组元。然后通过如下公式即可向当前体系施加目标应变。

> $M_{new} = (E + M_{ad} * P) * M_{old}$

其中，M_{old}是当前POSCAR中的应变矩阵，E为单位矩阵，P为阻尼系数，M_{new}为施加应变后新的POSCAR的应变矩阵。重复以上步骤，直到当前外压与目标外压的差值达到我们的收敛条件。

容我喝口*咖啡*。现在，我们具体讲一下用法。首先复制vaspeqstress.sh到你的工作目录。我以VASP官网的bccNi为例，设备是我自己的虚拟机，因为我自己用的是广州超算，所以并不支持脚本批量调用程序，也就是说，*这个脚本对我自己并没有用*。好了，言归正传，现在你需要修改一下脚本中的相关参数。

```
22 ##PBS -V
23 ##PBS -N test
24 #Set the target pressure tensor(xx, yy, zz, xy, yz, zx) in Kbar. Note pressu
25 #= -stress in VASP
26 Setpress=( 100 0.0 0.0 0.0 0.0 0.0 )
27 #convergency criteria for pressure,
28 presscirt=0.1
29 #Young's modulus in Kbar
30 E=2010
31 #Possion ratio
32 v=0.29
33 #Shear modulus in Kbar
34 G=$(echo "scale=8;$E/(2+2*$v)"|bc -l)
35 #maximum iteration cycles
36 imax=100
37 #damping factor for deformation
38 P=0.9
39 #Set your path effectively to run vasp
40 mpiexe="vasp_std"
41 #mpiexe='mpirun -np 12 vasp_std"
42
43 ###do not change following codes unless you know what you are doing###
44 #######initial run to get the pressure tensor present####
```

你需要将Setpress设置为你的目标外压(**FBI WARNING: VASP的外压正号表示压应力，负号表示拉应力**)，从左往右分别是XX, YY, ZZ, XY, YZ, ZX分量。presscirt是应力差的收敛标准，0.1够了，没事别改。E是材料的杨氏模量，v是泊松比，这俩你需要在[这][www.baidu.com]找。imax是最大迭代步数，100够了，要是超过100都没收敛，你。。就增大接着来吧。P是阻尼系数，防止外加应变矩阵加过头了。有进有退，方为中庸，P的作用就是中庸，嗯。mpiexe是你执行VASP的语句，这个就看你的作业系统了。其他就不要动了。脚本修改好后，INCAR之类就跟普通的优化是一样的，但要确保ISIF=2，现在就像这样。

```
INCAR   KPOINTS   POSCAR   POTCAR   vaspeqstress.sh
```

然后执行脚本。每一步的POSCAR会复制成poscar.*这样，你可以在pressure.all中查看每一步的外压，确定目前的收敛情况。

```
 13 95.38386 4.18844 4.18844 -0.00000 0.00000 -0.00000
 14 93.99385 2.90428 2.90428 -0.00000 0.00000 0.00000
 15 93.78063 4.21146 4.21146 0.00000 0.00000 0.00000
 16 93.06030 3.84064 3.84064 -0.00000 0.00000 -0.00000
 17 92.61911 5.71107 5.71107 0.00000 0.00000 -0.00000
 18 90.36736 6.20196 6.20196 -0.00000 0.00000 0.00000
 19 86.78365 6.03265 6.03265 -0.00000 0.00000 -0.00000
 20 86.24331 7.94320 7.94320 -0.00000 0.00000 0.00000
 21 83.90931 9.13736 9.13736 -0.00000 0.00000 0.00000
 22 78.41727 12.14339 12.14339 -0.00000 0.00000 -0.00000
 23 72.37715 15.83533 15.83533 -0.00000 0.00000 0.00000
 24 62.72041 19.74078 19.74078 0.00000 0.00000 0.00000
 25 38.49285 17.33533 17.33533 0.00000 0.00000 0.00000
 26 69.87023 82.16589 82.16589 0.00000 0.00000 0.00000
 27 -26.84989 5.65877 5.65877 0.00000 0.00000 -0.00000
 28 67.26247 45.44502 45.44502 -0.00000 0.00000 0.00000
 29 104.46233 2.50317 2.50317 -0.00000 0.00000 -0.00000
 30 97.66826 -0.95693 -0.95693 -0.00000 0.00000 -0.00000
 31 101.15994 0.26784 0.26784 0.00000 0.00000 0.00000
 32 99.30674 -0.09344 -0.09344 0.00000 0.00000 -0.00000
 33 100.37121 0.03941 0.03941 0.00000 0.00000 0.00000
 34 99.81953 -0.05727 -0.05727 0.00000 0.00000 -0.00000
 35 100.01022 0.01084 0.01084 0.00000 0.00000 0.00000
```

在我给的例子中可以看到，一共迭代了34次。最后的外压与我们的目标完全一致。

然后我们把施压的目标外压改成(0.0 100 0.0 0.0 0.0 0.0)，并且允许原子弛豫。从前面算完的外压状态go on。

```
 25 3.61601 95.47450 3.90724 0.00000 -0.00000 0.00000
 26 3.42987 94.41729 3.10453 0.00000 0.00000 0.00000
 27 3.48695 93.02731 3.96076 0.00000 0.00000 0.00000
 28 4.69870 93.87042 4.20261 -0.00000 0.00000 0.00000
 29 4.96765 92.31944 5.41058 0.00000 0.00000 0.00000
 30 6.52839 90.96527 6.29097 0.00000 0.00000 0.00000
 31 5.77709 86.66445 5.73447 0.00000 0.00000 0.00000
 32 8.58086 87.30423 8.71257 0.00000 0.00000 0.00000
 33 7.42551 82.49030 7.19082 0.00000 0.00000 0.00000
 34 12.47742 80.77355 12.73581 0.00000 0.00000 0.00000
 35 15.29584 73.45228 15.01354 0.00000 0.00000 0.00000
 36 18.89567 64.06291 19.23358 0.00000 0.00000 0.00000
 37 24.54394 49.53718 24.39475 0.00000 0.00000 0.00000
 38 15.21145 9.30816 15.21145 -0.00000 -0.00000 0.00000
 39 65.36422 31.65811 65.36422 0.00000 0.00000 0.00000
 40 44.02787 51.46745 44.02787 0.00000 0.00000 0.00000
 41 7.01026 105.41807 7.01026 -0.00000 -0.00000 0.00000
 42 -1.28343 97.18891 -1.28343 0.00000 0.00000 0.00000
 43 0.41316 101.52461 0.41316 0.00000 0.00000 0.00000
 44 -0.16376 99.02102 -0.16376 -0.00000 0.00000 0.00000
 45 0.02040 100.52456 0.02040 0.00000 -0.00000 0.00000
 46 0.00249 99.76045 0.00249 -0.00000 0.00000 0.00000
 47 -0.00381 100.06619 -0.00381 -0.00000 0.00000 0.00000
```

这次明显难收敛了，花了46次迭代。总之，目前我测试看来，代码还是works well。

最后说几个tips。

- E 和 v不用太精确，毕竟实际的值都是多晶的，跟DFT不是严格符合的，意思一下差不多就好。
- 建议分次优化。第一次原子不要弛豫，第一次OK后接着第二次再放开原子。
- 再次警告，VASP的pressure项正值是压应力。

- 算的时候自己要清楚，VASP对压应力的计算是出了名的差。
- 脚本中有相应的PBS参数，如果你需要，请把#删了。
- 尽量使用正交胞，因为VASP的应力也好，压力也好，都是基于笛卡尔坐标系的。确保晶格的三个基矢要沿着X, Y, Z，当然，你如果头脑清醒的话，怎么弄都一样。

OK，基本就这样。有任何问题，请联系18709821294@outlook.com，或者随便在个DFT群里说一下，我就在你的身后。

任何人都可以自由地分享，传播，修改这个脚本，也希望更多的人加入到开源的队列中。

最后，感谢刚哥的邀请。

ponychen

2019/05/18