Convergence tests

For numerical computations it is imperative to check certain algorithmic parameters for "convergence". In general, this implies that the same system is calculated repeatedly while applying different values for these parameters until a certain output property does not change anymore. In that case, a parameter is called "converged".

In standard DFT electronic structure calculations (i. e. plane wave basis set and Monkhorst-Pack Brillouin zone sampling), the two main parameters which are imperative "to converge" are the plane wave energy cut-off (`ecut`) and the grid of wave vectors in reciprocal space (`kpts`). The property which should not change after a suitable set of parameters has been found is usually chosen to be the total energy (or in GPAW language: `potential_energy`).

1. Check convergence for your system with respect to the aforementioned parameters. First define a set of reasonable values, e. g. for the energy cut-off a range between 200 and 1200 eV and a uniform grid of $n \times n \times n$ points with n in the range from 4 to 16.

2. Use a for-in-loop to iterate through the energy range in a pythonic way. What energy cut-off is at least necessary to "converge" the system to a total energy difference of less than 2 meV per atom?
   Solution:

```python
# import required modules
from ase.build import bulk
from gpaw import GPAW, PW
import time

# create bulk structure
si = bulk("Si", "diamond", 5.43)
# prepare empty list for storing resulting total energies
etots = []
# loop over following ecuts: [200, 300, 400, ..., 1200]
ecuts = range(200, 1201, 100)
for ecut in ecuts:
    # create and set calculator for silicon cell object
    si.calc = GPAW(mode=PW(ecut), xc="PBE", kpts=(8, 8, 8), txt=None)
    # get total energy via DFT and time calculation
    start = time.time()
    etot = si.get_potential_energy()
    end = time.time()
    # append result to list
    etots.append(etot)
    # round results and print some user information
    print("%4d \t %.6f \t %.2f" % (ecut, etot, end - start))

# convert results into energy differences in meV per atom
dEcuts = []
# list of differences has one element less than the 'etots' list
for i in range(len(etots) - 1):
    dE = etots[i+1] - etots[i]
    dEcuts.append(dE)
    print("E(%4d) - E(%4d) = % .2e" % (ecuts[i+1], ecuts[i], dE))
```

3. Use the same script again and adapt it for testing convergence with respect to the k-grid.
Solution:

```python
# import required modules
from ase.build import bulk
from gpaw import GPAW, PW
import time

# create bulk structure
si = bulk("Si", "diamond", 5.43)

etots = []
kpts = range(4, 16)
for k in kpts:
    si.calc = GPAW(mode=PW(100), xc="PBE", kpts=(k, k, k), txt=None)
    start = time.time()
    etot = si.get_potential_energy()
    end = time.time()
    etots.append(etot)
    print("%2d \t %.6f \t %.2f" % (k, etot, end - start))

# convert results into energy differences in meV per atom
dEkpts = []
for i in range(len(etots) - 1):
    dE = etots[i+1] - etots[i]
    dEkpts.append(dE)
    print("E(%2d) - E(%2d) = % .2e" % (kpts[i+1], kpts[i], dE))
```

Formatted printing in Python 3
Have a look at the python printing function options for prettier text output:
https://pyformat.info

In general, you should prefer the `.format()` version over the older one using %. The solution examples make already extensive use of formatted printing but stick with the old style for line-length reasons.

4. After analyzing the output numerically, what can you tell about:

- Values of converged parameters for your system compared to the other systems
Solution: Larger unit cells can be calculated using a coarser k-grid. Larger atoms with more electrons need higher cut-off energy. Metals need more kpts than semiconductors due to the occupation around the Fermi level, which makes calculations more difficult.

- Time consumption when increasing the reciprocal grid size or energy cut-off
Solution: The most common way to parallelize is over the k-grid, i. e. each process calculates "its" k-point independently of all other processes. Conversely, in serial runs the Kohn-Sham equations are calculated for each k-point seperately. Each k-point takes approximately the same computing time. Since we use a regular grid, a doubling of the number of k-points in each direction results in an increase of total number of k-points by a factor of $2^3 = 8$ and equally for the total computation time.

5. Visualize the data using matplotlib
   Solution:

```
# since dEcuts and dEkpts have one element less than ecuts and kpts,
# we have to remove e.g. the first element from latter using slices:
# array[start:stop] gives elements from index start till (stop-1).
# for [:stop] --> start = 0, for [start:] stop --> end of list


# -- easy version: two separate plots --
# import plot module
import matplotlib.pyplot as plt

fig1 = plt.figure()
plt.plot(ecuts[1:], dEcuts)
plt.xlabel("Plane wave energy cut-off in eV")
plt.ylabel("deltaE = Etot[i+1] - Etot[i]")
# use separate figures for both plots
fig2 = plt.figure()
plt.plot(kpts[1:], dEkpts)
plt.xlabel("K-point grid (k x k x k)")
plt.ylabel("deltaE = Etot[i+1] - Etot[i]")
# draw both plots to screen
plt.show()


# -- advanced version: one figure with 2 different axis-systems --
# import plot module
import matplotlib.pyplot as plt

# create new empty figure
fig = plt.figure()
# create subplot for lower x-axis
ax1 = fig.add_subplot(111)
# create second x-axis at the top
ax2 = ax1.twiny()
# plot the two data sets
ax1.plot(ecuts[1:], dEcuts, "r", lw=2, label="PW cut-off")
ax2.plot(kpts[1:], dEkpts, "b", lw=2, label="k-grid")
# set labels
ax1.set_xlabel("Plane wave energy cut-off [eV]")
ax2.set_xlabel("K-point grid size")
# with r'text with formula $a^2 + b^2$' we can use latex in labels
ax1.set_ylabel(r"$E_\mathrm{tot}^{i+1} - E_\mathrm{tot}^i$ [eV]")
# enable legends for both plots
ax1.legend(loc="upper left")
ax2.legend(loc="upper right")
# render the plot to screen
plt.show()
```