1 Observing CIF crystal structures using ASE

A helpful tool to create and modify molecule or bulk structures is the Atomic Simulation Environment (ASE). We will use this software to create our own crystal structures and save them as CIF files.

1. Visit the ASE homepage: https://wiki.fysik.dtu.dk/ase
2. Look at available calculators listed on the start page
3. Scan the gallery for common use cases
4. Load your CIF file
   Solution:

```
# load the 'read' function from the ASE library
from ase.io import read

# load your CIF file using ASE and store the resulting object
cell = read("Si-51688.cif")
```

5. Check the data structure via getter-functions as described in the ASE atoms object reference: https://wiki.fysik.dtu.dk/ase/ase/atoms.html
   Solution:

```
# use getter functions to display some system properties
cell.get_cell().round(3)
cell.get_atomic_numbers()
cell.get_chemical_formula()
cell.get_chemical_symbols()
cell.get_positions().round(3)
cell.get_scaled_positions().round(3)
cell.get_number_of_atoms()
cell.get_volume()
```

2 Build your own structure using ASE

1. Build a bulk structure with the same parameters (angles, atom positions, lattice constants) as given in your CIF file using ASE. You can follow this tutorial from the documentation: https://wiki.fysik.dtu.dk/ase/ase/spacegroup/spacegroup.html

   Solution:

```
# create a cell object using ASE's crystal module
from ase.spacegroup import crystal
a = 5.43
cell_crystal = crystal('Si', basis=[(0, 0, 0)], spacegroup=227,
                       cellpar=[a, a, a, 90, 90, 90])
```

2. Compare the resulting cell with the one that has been obtained by simply loading the CIF file into ASE.
   Solution: Basically the same, except possible differences in the order of atomic positions.

3. Repeat the last task but this time use the `build` module provided by ASE to create the cell. https://wiki.fysik.dtu.dk/ase/ase/build/build.html

   What changes? Which one might take less computational time for a ground state calculation?
   Solution:

```
# create another cell object using ASE's bulk module
from ase.build import bulk
cell_bulk = bulk('Si', 'diamond', 5.43)
```

   CIF and crystal module create a conventional cell, whereas the bulk module returns the primitive cell. The larger cell contains more atoms and consequently will take more computational time, even when considering symmetries of the spacegroup.

   For instance, on my laptop a ground state DFT calculation for the primitive setting took 40s whereas the same calculation for the conventional cell run 640s, that's a huge difference!

   Generally, DFT scales for large number of electrons N in the worst case as $\mathcal{O}(N^3)$ because of the diagonalization of the Hamiltonian.

4. Write your cell as CIF file and have a look at the content. Do you recognize any peculiarities?
   Solution:

```
from ase.io import write
# write cell to CIF file
write("ASE.cif", cell_crystal)
```

   Even after creating the conventional cell using the `spacegroup` submodule and setting the SG to e. g. 227, the written CIF will always report a primitive triclinic cell with SG 1. This is because after creating the cell using ASE, no symmetry information is stored in the cell object.

3 Fun fact - Have a guess!

How many HDDs (each able to hold 1 TB of data) are required in order to store the ground state of a H_2O molecule with 10 values per electronic coordinate for a single time step?

Hint 1: The H_2O molecule contains 2x1(H) + 1x8(O) = 10 electrons
Hint 2: The unapproximated many-body state is a function of all 10x3 = 30 coordinates

Solution:

1. $\Psi = \Psi(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{10}, t)$, where $\boldsymbol{x}_i = x_i^1, x_i^2, x_i^3$ $\longrightarrow$ 3x10 = 30 variables (not counting time)
2. 10 values per coordinate $\longrightarrow$ 10^{30} double precision floating point values
3. 8 Byte per value $\longrightarrow$ $\approx 10^{31}$ Byte
4. 1 TB = 1000^4 Byte = 10^{12} Byte per HDD $\longrightarrow$ 10^{19} HDDs
5. 1 kg per HDD $\longrightarrow$ 1×10^{19} kg = 1×10^{16} t

Reasoning behind Density Functional Theory:
Use information contained in ground state density n to reconstruct everything else. Since $n = n(\boldsymbol{x})$ with $\boldsymbol{x} = (x, y, z)^\intercal$, this will require merely 10^3 * 8 byte = 8 kB of storage for a single time step! This would even fit onto old school floppy disks!