AiiDA Defects: a python toolkit for modeling point defects

Chiara Ricca[1,2] and Ulrich Aschauer[1,2*]

[1] *University of Bern, Department of Chemistry and Biochemistry, Freiestrasse,3 CH-3011 Bern Switzerland* and
[2] *National Centre for Computational Design and Discovery of Novel Materials (MARVEL), Switzerlan*

(Dated: December 19, 2018)

* ulrich.aschauer@dcb.unibe.ch

I. INTRODUCTION

aiida_defects is a python package to automate calculations oif defective system with Quantum ESPRESSO [1] through the AiiDA platform [2] (See Fig. 1) . It consists of several modules allowing to create defective structures (*tools*) thorugh AiiDA and containg AiiDA workflows to calculate defect formation energies according to different correction schemes ($formation\_energy$). This last module requires the knowledge of the dielectric constant of the material and to perform calculation of the electrostatic potential. AiiDA workflows for these tasks have also been developed (*epsilon* and *pp*) and can be used independently from the other modules of the package.

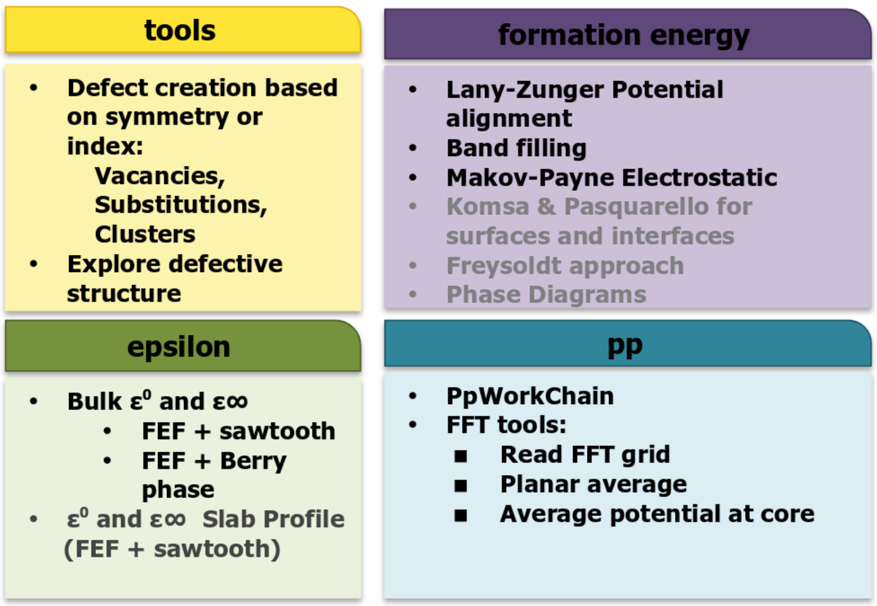
tools
Defect creation based on symmetry or index:
Vacancies,
Substitutions,
Clusters
Explore defective structure
formation energy
Lany-Zunger Potential alignment
Band filling
Makov-Payne Electrostatic
Komsa & Pasquarello for surfaces and interfaces
Freysoldt approach
Phase Diagrams
epsilon
Bulk ε⁰ and ε∞
FEF + sawtooth
FEF + Berry phase
ε⁰ and ε∞ Slab Profile (FEF + sawtooth)
pp
PpWorkChain
FFT tools:
Read FFT grid
Planar average
Average potential at core

FIG. 1: Schematic representation of the main functionalities of the main modules (*tools*,*formation energy*,*epsilon*,*pp*) of the *aiida_defects* package. Entries in grey are currently underdevelopment.

A. Requirements

The package is written in Python 2.7 and was tested with version 0.12.1 of AiiDA. It requires some python libraries that can be easily installed usig *pip*: NumPy, SciPy, matplotlib, pymatgen, and ASE.

B. Installation

To install the package it is sufficient to clone the repository in your `/path/aiida/workflows/user` folder:

```
cd /path/aiida/workflows/user
```

```
git clone https://gitlab.com/ChRicca/aiida_defects.git
```

In order to use the functions and workflows provided by the package, it is the enough to import them in your python script, for example:

```
from aiida.workflows.user.aiida_defects.tools.defects import defect_creator
```

II. THEORETICAL FRAMEWORK

A. Defect Formation Energy

The formation energy is a key property to describe the defect formation in a material, being the cost to form or create an isolated defect (*i.e.* dilute limit) in the host system. Periodic boundary conditions (PBCs) are ideal to describe pristine crystals and are often emplyed in DFT calculations. In this case, defects are simulated using supercells of the host unit cell, which are then periodically repeated in the three dimensions. In these type of calculations, the formation energy of a defect X in a charge state q ($\mathrm{E_{f,X^q}}$) can be computed as [3]

$$\mathrm{E_{f,X^q}} = \mathrm{E}_{\mathrm{tot},X^q} - \mathrm{E_{tot,host}} - \sum_i n_i \mu_i + \mathrm{qE_F} + \mathrm{E_{corr}} \tag{1}$$

where $\mathrm{E}_{\mathrm{tot},X^q}$ is the total energy of the defective supercell, $\mathrm{E_{tot,host}}$ is the total energy of the host supercell, the third therm is the chemical potential of the species involved in the formation of the defect, the fourth term corresponds to the energy of the electronic reservoir and $\mathrm{E_{corr}}$ accounts for finite-size corrections within the supercell-approach, like electrostatic and/or elastic interactions betweeen supercells or finite **k**-point sampling in the case of shallow defects, that will be discussed in detail in section II A 3.

1. *Chemical Potential*

Chemical potentials represent the energy of the reservoir with which atoms involved in the defect formation are exchanged. In equation 1, n_i indicates the number of atoms of a certain specie i that are added ($n_i > 0$) or removed ($n_i < 0$) from the supercell to form the defect, while the chemical potential of the specie i is given by the sum chemical potential of that specie in its stable configuration μ_i^0 and a quantity $\Delta\mu_i$ that define the thermodynamic limits of the chemical potential:

$$\mu_i = \mu_i^0 + \Delta\mu_i \tag{2}$$

Chemical potentials reflect the experimental synthesis conditions. By varying them different growth conditions can be explored, allowing the use of this formalism as guide to defect engineering in material design.

2. *Fermi Energy*

The forth term in equation 1 defines the cost associated to the addition or removal of electrons from the host system when charged defects are created. This term depends from the Fermi energy ($\mathrm{E_F}$), the electron chemical potential, which is conventionally defined with respect to the valence band maximum ($\mathrm{E_V}$) of the defect-free system:

$$\mathrm{E_F} = \mathrm{E_V} + \Delta\mathrm{E_F} \tag{3}$$

with $\Delta\mathrm{E_F}$ indicating the position of the Fermi level within the band gap (E_g) of the host ($0 \leq \Delta\mathrm{E_F} \leq \mathrm{E}_g$)

3. *Corrections for finite-size supercells*

When PBCs and supercell geometries are used to simulate defects, the isolated defect is replaced by a periodic array of defects, often resulting in unrealistically large defects concentrations and unphysical sizable interactions between the defect and its periodic images in the neighboring supercells. Such interactions can be electrostatic, elastic or quantum mechanical in nature. Quantum mechanical effects due to the overlap of the wavefunctions between neighboring supercells influences the predicted dispersion of the defect band. Electrostatic interactions instead affect the simulation of charged defects. Such simulations require including a compensating background charge ensuring the neutrality of the unit cell to avoid divergence of the electrostatic potential. Furthermore, the creation of a charged defect modifies the electrostatic energy of the atoms closest to it. The result is a constant shift in the electrostatic potential compared to the host. Since the formation energy of charged defects depend on the Fermi energy, which is referenced to the host valence band maximum and depends on the average electrostatic potential in the stoichiometric material, the electrostatic potential of the defective and host supercells needs to be realigned [3]. Moreover, the unphysical Coulombic interactions of a defect with its periodic images and the constant background also make spurious contributions to the calculated energy of the defective system that need to be adressed. Finally, elastic interactions can arise when the defect distorts the surrounding lattice [4], but they decay more rapidely in real space with respect to electrostatic interactions and are often minimal and will hence not further discussed [3].

Since the formalism in which equation 1 is derived is consistent with the dilut limit, minimizing these artificial effects is important to obtain accurate formation energies. The most straightforward approach is to perform simulation of supercell of increasing size and to extrapolate the formation energy to the one of the infinitely large supercell. However, the slow convergence of the defect formation energy with the supercell size, especially in the case of the long-range electrostatic interactions, can make this approach very computationally demanding. An alternative strategy is to use smaller supercells and apply *a posteriori* corrections. In the following we are going to discuss the correction schemes implemented in the package.

Makov-Payne Correction

The magnitude of the spurious image-charge interactions can be estimated from the Madelung energy af an array of point charges in the presence of homogeneous compasating background charge, which decays as q^2/L, where L is the supercell-size, and dependes on the Madelung constant α_M of the lattice [5]. Focusing on cubic systems, Makov and Payne[6] introduced one of the first correction schemes for the electrostatic interaction (E_{MP}) made by a Madelung-like and a quadrupolar term arising from the interaction of the localized charge distribution (Q) with the uniform neutralizing background:

$$E_{MP} = \frac{q^2\alpha_M}{2\varepsilon_r L} - \frac{2\pi q Q}{3\varepsilon_r L^3} \tag{4}$$

where ε_r is the macroscopic dielectric constant to account for screening. This correction is normally applied for bulk geometries under the assumption that the dielectric response of the material is isotropic. Lany and Zunger[7] proposed a simplified version, where E_{MP} is computed by rescaling the first order correction ($\frac{q^2\alpha_M}{2\varepsilon_r L}$) computed in the case of $q = 1$ and $\varepsilon_r = 1$, according to the dielectric constant of the host system (ε_r) and its geometrical properties (c_{sh}):

$$E_{MP} = [1 + c_{sh}(1 - \frac{1}{\varepsilon_r})]\frac{q^2\alpha_M}{2\varepsilon_r L} \tag{5}$$

The shape factor c_{sh} is derived from known values on the base of the geometry of the supercell as can be seen in table I. More accurate evaluations whoud require to calculate the c_{sh} for the specifc supercell geometry.

TABLE I: Values for the c_{sh} factor on the base of the geometry of the host cell-

Geometry	c_{sh}
sc	-0.369
fcc	-0.343
bcc	-0.342
hcp	-0.478

Potential Alignment

One of the easiest way of computing the potential alignment (ΔE_{PA}) is to calculate the difference in the electrostatic potential between the charged ($V^r_{X,q}$) and host (V^r_{host}) supercells on one atom far away from the defect:

$$\Delta E_{PA}(X,q) = q(V^r_{X,q} - V^r_{host}) \tag{6}$$

where r is the distance of the selected atom from the defect. In order to increase the accuracy of the correction, Lany and Zunger[8] suggested to determine $V^r_{X,q}$ and V^r_{host} by atomic-sphere-averaged electrostatic potentials at more than one atomic site located at a reasonable distance from the defect.[8, 9]

Bandfilling Correction

Because of the high defect concentrations normally associated with the use of finite-size supercells, the carriers (electrons or holes) occupy defect band states that are artificially dispersive due to the overlap of the defect wave-functions in neighboring supercells. This band filling Moss-Burstein-type effects[10, 11] are particularly important for shallow defect states. To recover the dilute limit for the formation energy, correction of the energy for the defect cell must be applied. The correction for a shallow donor (ΔE_{MB}^{don}) or acceptor (ΔE_{MB}^{acc}) is given by:

$$\Delta E_{MB}^{don} = -\sum_{n,k} w_k \eta_{n,k} (\epsilon_{n,k} - \widetilde{e_C}) \Theta(\epsilon_{n,k} - \widetilde{e_C}) \tag{7}$$

$$\Delta E_{MB}^{acc} = -\sum_{n,k} w_k (2 - \eta_{n,k}) (\widetilde{e_V} - \epsilon_{n,k}) \Theta(\widetilde{e_V} - \epsilon_{n,k}) \tag{8}$$

where $w_{n,k}$ represents the weight of the k-point k, $\eta_{n,k}$ the occupation of band n for k-point k, $\epsilon_{n,k}$ is the energy of band n at k-point k, $\widetilde{e_C}$ and $\widetilde{e_V}$ the conduction and valence band energies in the host system after the potential alignment, and Θ is the Heaviside function.

B. Dielectric Constant

The dielectric constant for bulk can be evaluated either using a finite electric field (FEF) method in the framework of the Berry Phase formalism or by applying a finite sawtooth potential withinin the FEF approach [12**?**] can be applied. More accurate calculation based on density-functional perturbation theory (DFPT) [13, 14] could also be applied, but these methods have the advantage that the computationally demanding calculation of the dynamical matrix is avoided.

1. Sawtooth potential and Finite Electric Fields

In this approach a sawtooth potential V^{saw} is applied to an elongated cell of the bulk material. This will induce a change in the effective potential (ΔV^{SCF}), screened by the dielectric constant, that will also show a sawtooth shape (cf. Fig. 2). The dielctric constant can be computed by comparing the slope of the applied and self-consistent potential between the turning points:

$$\varepsilon = \frac{\partial V^{saw}/\partial z}{\partial \Delta V^{SCF}/\partial z} \tag{9}$$

When applying this approach, it is important that the following conditions are respected: [15]

- the hight of the sawtooth potential should be lower than the bandgap to avoid dielectric breakdown
- tighther convergence criteria are required to ensure accurate results
- the turning points should be excluded from the slope fitting because on correspondence to these points there are deviations from the macroscopic screening behaviour which extend over a range of $a_0/2$

If ionic relaxation are not taken into account into the defect calculation, the dielectric constant that must be used for the calculation of the corrective terms to the formation energies is the high-frequency one, while if relaxations are included, one need to use the low-frequency dielectric constant, meaning that in the sawtooth calculations ionic relaxation must be included [15, 16].

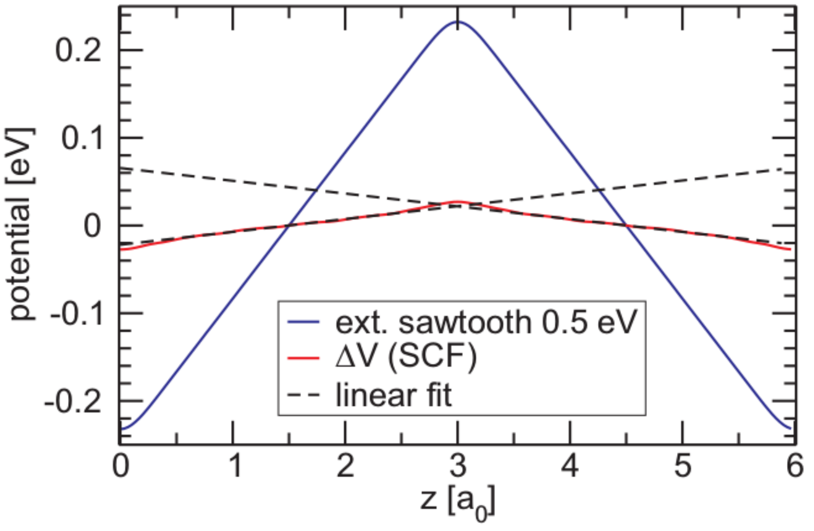

FIG. 2: Determination of the dielectric constant applying to the material a sawtooth potential V^{saw} (in blue) that induces a achange in the self.considtent potential (ΔV^{SCF}, in red). From ref. [15]

2. Berry Phase Formalism

An alternative approach would be to compute the dielectric constant using FEF coupled to the Berry phase formalism [17–19]. In this case the static and high-frequency dielectric constants can be calculated from differences of the polarization induced by different values of the applied electric field. The dielectric constant can be computed according to the following formula:

$$\varepsilon = 1 - \frac{4\pi}{a}\frac{\partial E^2}{\partial \epsilon^2} \tag{10}$$

where E is the energy per primitive cell and ϵ the intensity of the applied electric field and a is the periodicity. [18, 19]

III. MODULES AND WORKFLOWS

IV. TOOLS

A. Module for defect creation: *defects.py*

The *defects.py* module allows to create defects in a host structure. The defects can be vacancies, substitutional defects (and/or antisites) or cluster of defects obtained by the combination of several vacancies and substitutional defects at the same time. Interstial defect are not implemented. It contains the following functions:

- `defect_creator(host_structure,vacancies,substitutions,scale_sc,cluster)` This workfunction allows you to create defects into a host structure on the base of symmetry considerations. The function stronly employs the capabilitieof *pymatgen* [20] for the symmetry analysis. It takes as input:
  - `host_structure`: StructureData object corresponding to the host structure
  - `vacancies`: AiiDa List containg the specie of the host_structure for which we want to create vacancies. If clusters of the same vacancy type are to be formed the element should be repeted.
  - `substitutions`: AiiDA ParameterData object containing entries like $"Mn"$: $["Ti", "Co"]$ meaning that Mn in the host_structure will be substituted with either Ti or Co.
  - `scale_sc`: scale parameter for the supercell creation. AiiDA List of the type [a,b,c], which will result in a (a×b< $times$c) supercell.

  - `cluster`: AiiDA boolean. If true defective structures containing all the requested defects are created. Otherwise defective structures, each containing only one of the requested defect type, will be created.

  Since it is an AiiDA workfunction, the inputs should be AiiDA objects: StructureData, List, ParameterData, Int, Bool. It returns *defective_structure*, dictionary containing all the defective structures created, as StructureData objects. The key value is given by the type of the defect (vacancy, substitutions, cluster) followed by a different number for every different structure (*e.g.* "vacancy_", "substitution_1", "cluster_2"). The first structure is always the host structure (*i.e* "vacancy_0", "substitution_0", "cluster_0"). Being an AiiDa workfunctions provenance of the defective structure is properly preserved in the database.

- `defect_creator_by_index(structure,find_defect_index_output)`

  This workfunction allows to create defective structure starting from the host, but in this case one should indicate in the input which is the index in the host structure corresponding to the site that we want to subtitute or to remove to form a vacancy.

  - `host_structure`: StructureData object corresponding to the host structure
  - `find_defect_index_output`: AiiDA ParameterData object containing entries the information about the index of the atom the StructureData object interested by the defect creation:
    - For a **vacancy**; `find_defect_index_output=ParameterData(dict={'vacancy_1':{'index':XX}})`, where `XX` is an integer corresponding to the index of the atom for which we want to create the vacancy.
    - For a **substitutional**; `find_defect_index_output=ParameterData(dict={'substitution_1':{'index':XX,'atom_type':YYY}})`, where the atom at index `XX` is going to be substitued with another type specified by the string `YY`.
    - For a **cluster**; `find_defect_index_output=ParameterData(dict={'cluster_1':{'defect_name_v_0':{'index':XX},'defect_name_s_0':{'index':XX,'atom_type_s_0':YYY}})}`

- `explore_defect(host_structure,defective_structure,defect_type)`

  This function explores the defective structure by comparing it with the host one. It finds the position and the element type of the defect and assigns a name to it (*e.g.* V_O for oxygen vacancies or Ba_Sr for the substitution of a Sr site with Ba). It takes as inputs:

  - `host_structure`: a StructureData object corresponding to the host structure
  - `defective_structure`: a StructureData object corresponding to the defective structure
  - `defect_type`: type of defect. Possible values: "vacancy", "substitution", "cluster", "unknown".

  It returns a dictionary containing for every defect found the following items:

  - Numpy array of the position of the defect (in cartesian coordinates)
  - atom type
  - name to classify the defect

  This function is based on the assumption that supercell shape, volume and atom coordinates are the same in `host_structure` and `defective_structure`. If the defective structure was generated using the `defect_creator` workfunction, use the host structure returned by the same function for `host_structure`.

- `distance_from_defect(defective_structure,defect_position)`

  This function used *pymatgen* to compute the distance from a defect for every site of the structure. It takes as inputs:

  - `defective_structure`: StructureData object containing the defective structure
  - `defect_position`: numpy array containing the cartesian coordinates of the defect.

  It returns a dictionary containing one entry for each Periodic Site (*pymatgen* periodic site) corresponding to the distance of the site from the defect.

The reader can find examples of the usage of this functions in the Jupyter Notebook *tools.ipynb* in the `aiida_defects/examples_aiida_defects` folder.

V. PP

In order to compute the dielectric constant or to compute the potential alignment, the calculation of the electrostatic potential is necessary and in Quantum ESPRESSO that can be performed using the *pp.x* program (*PpCalculation* in AiiDA).

A. *pp.py* Module: *PpWorkChain*

The *PpWorkChain* allows to automatize a *PpCalculation*. One could either decide to perfom first a *PwCalculation* (setting `pw_calc=Bool(True))` followed by the *pp.x* calculation, or could directly perform the *pp.x* one. In the latter (case `pw_calc=Bool(False))`, a parent folder of a *PwCalculation* (`remote_folder`) or the *PwCalculation* itself (`parent_calculation`) should be specified. One needs to specify as input:

- `structure`: a StructureData object
- `code_pw`: Str with the name of the *pw.x* code. It is not mandatory in case in which the *PwCalculation* has been previously computed.
- `code_pp`: Str with the name of the *pp.x* code.
- `pseudo_family`: Str with the name of the family of pseudopotentials to use. Not necessary if the *PwCalculation* has been previously computed.
- `options`: ParameterData
- `settings`: ParameterData
- `kpoints`: KpointsData to apply in the *pw.x* run. Not necessary if the *PwCalculation* has been previously computed.
- `parameters`: ParameterData for the *pw.x* run. Not necessary if the *PwCalculation* has been previously computed.
- `parameters_pp`: ParameterData for the *pp.x* run. Not necessary if the *PwCalculation* has been previously computed.
- `remote_folder`: FolderData obtanied from the *PwCalculation* if this was previously performed
- `pw_calc`: Bool, True if you need to perform the *pw.x* run
- `parent_calculation`: *PwCalculation* if this was previously performed

The output of the workchain is the *'retrieved'* node with the FolderData produced by the *PpCalculation*. Since the result of the *PpCalculation* is not parsed by AiiDA, in case the calculation did not finish correctly the in the report of the *PpWorkChain* will appear a message indicating that the *'filplot'* file containing the 3D-FFT grid was not found.

The reader can find examples of the usage of this functions in the Jupyter Notebook *pp.ipynb* in the `aiida_defects/examples_aiida_defects` folder.

B. Manipulating the FFT grid: *fft_tools.py*

The electrostatic potential computed through a *pp.x* calculation needs to be adequately manipulated to extract the desired information. *fft_tools.py* provides some tools necessary for this task. It contains the following functions:

- `read_grid(folder_data)`

  This workfunction reads the FFT grid stored into the *'aiida.filplot'* file stored into the FolderData object that it is stored into the database after a *PpCalculation* and returns as output a dictionary containing two entries: *'fft_grid'* corresponding an ArrayData object contaning the FFT grid and *'info'* corresponding to a dictionary containing information about the dimension of the smooth and dense FFT meshes. It takes as input:

  - `folder_data`: FolderData object output of the Ppcalculation

- `planar_average(Grid,structure,axis,npt=400)`

  This function compute the planar average of the quantity contained in the FFT grid. It take as input:

  - `Grid`: dictionary created by the `read_grid` workfunction
  - `structure`: StructureData object for which the Grid was computed
  - `axis`: string corresponding to the axis along which the planar average should be computed. Allowed values: $'x'$, $'y'$, or $'z'$.

  It returns as output a dictionary containing three entries: $'average'$ a numpy array corresponding to planar averaged potential in eV and $'ax'$ a list containing the values along the axis in Angstrom.

VI. MODULE *epsilon*

1. *bulkepsilonberry.py Module:* $BulkEpsilonBerryWorkChain$

The *BulkEpsilonBerryWorkChain* allows to compute the dielectric constant of a bulk material using the FEF approach coupled with the Berry phase method (See Sec. II B 2). The $Workchain$ will first perform a calculation in absence of external field followed by one in presence of a field of a certain amplitude and then simply derive the dielectric constant from the change in the electronic dipole observed in the two cases and from the volume of the supercell. At the moment on ly the high-frequency dielectric constant is implemented. One needs to specify as input:

- `structure`: a StructureData object
- `code_pw`: Str with the name of the *pw.x* code. It is not mandatory in case in which the *PwCalculation* has been previously computed.
- `code_pp`: Str with the name of the *pp.x* code.
- `pseudo_family`: Str with the name of the family of pseudopotentials to use. Not necessary if the *PwCalculation* has been previously computed.
- `options`: ParameterData
- `settings`: ParameterData
- `kpoints`: KpointsData to apply in the *pw.x* run. Not necessary if the *PwCalculation* has been previously computed.
- `parameters`: ParameterData for the *pw.x* run. Not necessary if the *PwCalculation* has been previously computed.
- `eamp`: Float containing the values of the amplitude of the applied field in a.u. Default value is 0.001 a.u.
- `edir`: Int referring to the direction along which the fild is to be applied. Allowed values: 1, 2, and 3 correspondinf to x, y, and z axis

The $WorkChain$ provides in the output the node $'epsilon'$ with the float value of the computed dielectric constant.

WARNING: You would need to check convergence of the results with respect to the k-mesh. One normally need to have a denser sampling of the k-points for the direction along which the field is applied.

The reader can find examples of the usage of this functions in the Jupyter Notebook $epsilon_bulk_berry.ipynb$ in the `aiida_defects/examples_aiida_defects` folder.

2. *bulkepsilonsawtooth.py Module: BulkEpsilonSawtoothWorkChain*

The *BulkEpsilonSawtoothWorkChain* allows to compute the dielectric constant of a bulk material using the FEF approach coupled with the Berry phase method (See Sec. II B 1). The *WorkChain* will first create a supercell of the bulk unit cell provided as input in the direction along which the field will be applied than it will perform a calculation of the electrostatic potential in absence of external field, follwed by a calculation of the electrostatic potential and of the applied sawtooth potential in presence of the field. The dielectric constant is then computed according to equation 9 performing linear fitting of the planar avarage of the above quantities taking care of avoiding the region close to the turning point of the applied sawtooth potential. One needs to specify as input:

- `structure`: a StructureData object
- `code_pw`: Str with the name of the *pw.x* code. It is not mandatory in case in which the *PwCalculation* has been previously computed.
- `code_pp`: Str with the name of the *pp.x* code.
- `pseudo_family`: Str with the name of the family of pseudopotentials to use. Not necessary if the *PwCalculation* has been previously computed.
- `options`: ParameterData
- `settings`: ParameterData
- `kpoints`: KpointsData to apply in the *pw.x* run. Not necessary if the *PwCalculation* has been previously computed.
- `parameters`: ParameterData for the *pw.x* run. Not necessary if the *PwCalculation* has been previously computed.
- `epsilon_type`: Str with the type of the dielectric constant. Allowd values: $'low-frequency'$ and $'high-frequency'$ (default is $'high-frequency'$)
- `eamp`: Float containing the values of the amplitude of the applied field in a.u. Default value is 0.001 a.u.
- `edir`: Int referring to the direction along which the fild is to be applied. Allowed values: 1, 2, and 3 correspondinf to x, y, and z axis
- `sc`: Int indicating the dimension of the supercell that will be created along the direction of the applied field. It is suggested not to modify the default value of 6.
- `eopreg`: Float indicatinf the zone in the unit cell where the saw-like potential decreases. (0 ¡ eopreg ¡ 1). See `https://www.quantum-espresso.org/Doc/INPUT_PW.html#eopreg`). It is suggested not to modify the default value.
- `eomax`:Float indicating the position in fractional coordinates of the maximum of the saw-like potential along crystal axis edir. It is suggested not to modify the default value. axis edir

The *WorkChain* provides in the output the node *'epsilon'* with the float value of the computed dielectric constant. Both low- and high-frequency dielectric constatnt can be computed.

The reader can find examples of the usage of this functions in the Jupyter Notebook $epsilon_bulk_sawtooth.ipynb$ in the `aiida_defetcs/examples_aiida_defects` folder.

VII. MODULE *formation_energy*

1. *MakovPayneCorrection*

MakovPayneCorrection allows to compute the Makov-Payne electrostatic correction for charged defects. In particular, the the first order correction ($\frac{q^2\alpha_M}{2\varepsilon_r L}$) (see equ. 4) is obtained through the following procedure:

1. we remove all the atoms from the bulk/host optimized structure

2. we add one H atom in the middle of the cell
3. we perform an scf calculation and $\Delta E_1(q = 1, \epsilon_r = 1)$ corresponds to the Ewald energy in the Quantum ESPRESSO output.

This value is then rescaled by the defect charge and the dielectric constant of the material. One should provide as input:

- `bulk_structure`: a StructureData object for the host system
- `code_pw`: Str with the name of the *pw.x* code. It is not mandatory in case in which the *PwCalculation* has been previously computed.
- `code_pp`: Str with the name of the *pp.x* code.
- `pseudo_family`: Str with the name of the family of pseudopotentials to use. Not necessary if the *PwCalculation* has been previously computed.
- `options`: ParameterData
- `settings`: ParameterData
- `kpoints`: KpointsData to apply in the *pw.x* run.
- `parameters`: ParameterData for the *pw.x* run.
- `epsilon_r`: Float corresponding to the bulk dielctric constant. It can be taken from experiments of computed using the $BulkEpsilonBerryWorkChain$ or $BulkEpsilonSawtoothWorkChain$.
- `defect_charge`: Float corresponding to the defect charge.

The output of the workchain is the 'Makov_Payne_Correction' node containing the Makov Payne electrostatic correction value in eV. This value can be simply added to the uncorrected fromation energy computed.

The reader can find examples of the usage of this functions in the Jupyter Notebook *makov$_p$yne.ipynb* in the `aiida_defects/examples_aiida_defects` folder.

2. $BandFillingCorrectionWorkChain$

$BandFillingCorrectionWorkChain$ allows to compute the bandfilling corrections (see II A 3) after the calculation of the bandstructure of the host and defective system is performed. One should provide as input:

- `host_structure`: a StructureData object for the host system
- `defect_structure`: a StructureData object for the defective system
- `code`: Str with the name of the *pw.x* code. It is not mandatory in case in which the *PwCalculation* has been previously computed.
- `pseudo_family`: Str with the name of the family of pseudopotentials to use. Not necessary if the *PwCalculation* has been previously computed.
- `options`: ParameterData
- `settings`: ParameterData
- `kpoint_mesh`: KpointsData to apply in the *pw.x* run. Not required.
- `kpoint_distance`: KpointsData to apply in the *pw.x* run. Not required.
- `host_parameters`: ParameterData for the *pw.x* run fo the host system.
- `defect_parameters`: Float, default value 0.2
- `potential_alignment`: Float corresponding to the potential alignment.

- `host_bandstructure`: Node output of a *PwBandsWorkChain* previously performed on the defective system. If this is provided as input the wokchain will skip the *PwBandsWorkChain* for the host.
- `skip_relax`: Bool. if False only one scf prior to the bandstructure calculation is performed.
- `skip_relax`: Dictionary that need to be specfied for the *PwBandsWorkChain* in case the relax option is requested.
- `defect_bandstructure`: Node output of a *PwBandsWorkChain* previously performed on the defective system. If this is provided as input the wokchain will skip the *PwBandsWorkChain* for the defective system.

The output of the workchain are `'E_donor'` and `'E_accpetor'` containing the bandfilling correction for donorsor acceptors, respectively. These values can be simply added to the uncorrected fromation energy computed.

The reader can find examples of the usage of this functions in the Jupyter Notebook *bandfilling.ipynb* in the `aiida_defects/examples_aiida_defects` folder.

3. pot_align.py Module: PotentialAlignmentLanyZunger

PotentialAlignmentLanyZunger allows to compute the potential alignment according to the scheme of Lany and Zunger (see II A 3). One have has two different possibilities: 1) performing a pw calculation followed by a pp calculation using the PpWorkChain for either bulk and defective systems; 2) provide the folder output of the *PwCalculation* or the *PwCalculation* Node for either the bulk or the defective system. It takes as input:

- `host_structure`: a StructureData object for the host system
- `defect_structure`: a StructureData object for the defective system
- `code_pw`: Str with the name of the *pw.x* code. It is not mandatory in case in which the *PwCalculation* has been previously computed.
- `code_pp`: Str with the name of the *pp.x* code.
- `pseudo_family`: Str with the name of the family of pseudopotentials to use. Not necessary if the *PwCalculation* has been previously computed.
- `options`: ParameterData
- `settings`: ParameterData
- `kpoint_mesh`: KpointsData to apply in the *pw.x* run. Not required.
- `kpoint_distance`: KpointsData to apply in the *pw.x* run. Not required.
- `parameters_pw_host`: ParameterData for the *pw.x* run fo the host system.
- `parameters_pw_defect`: ParameterData for the *pw.x* run fo the defective system.
- `parameters_pp`: ParameterData for the *pp.x* run
- `run_pw_host`: Bool. If True a *pw.x* run for the host is performed
- `host_parent_folder`: FolderData output of a *PwCalculation* of the host previosuly performed. `run_pw_host` = False
- `host_parent_calculation`: *PwCalculation* of the host previosuly performed. `run_pw_host` = False
- `run_pw_defect`: Bool. If True a *pw.x* run for the host is performed
- `defect_parent_folder`: FolderData output of a *PwCalculation* of the host previosuly performed. `run_pw_host` = False
- `defect_parent_calculation`: *PwCalculation* of the host previosuly performed. `run_pw_host` = False

The output of the workchain is a Float `'pot_align'` corresponding to the computed potential alignment. These values should be multiplied by the defect charge and then be simply added to the uncorrected fromation energy computed.

The reader can find examples of the usage of this functions in the Jupyter Notebook $lz-potalign.ipynb$ in the `aiida_defects/examples_aiida_defects` folder.

VIII. *DefectWorkChain*

This *WorkChain* allows to to automatize the calculation of defects. It will first perform an scf or a relaxation of the host structure provided as input using the *PwBaseWorkChain* or the *PwRelaxWorkChain*, respectively. It will then create the defect requested as input (in the charge state also specified in the inputs) and perform a an scf or a geometry optimization for all the created defective structures. Finally, it will compute the requested corrections: at the moment, one can request the Makov-Payne correction, the bandfilling correction or the Lany-Zunger potential alignment scheme. The workchain requires as input:

- `structure`: a StructureData object for the host system
- `code_pw`: Str with the name of the *pw.x* code. It is not mandatory in case in which the *PwCalculation* has been previously computed.
- `code_pp`: Str with the name of the *pp.x* code.
- `pseudo_family`: Str with the name of the family of pseudopotentials to use.
- `options`: ParameterData
- `settings`: ParameterData
- `kpoint_mesh`: KpointsData to apply in the *pw.x* run. Not required.
- `kpoint_distance`: KpointsData to apply in the *pw.x* run. Not required.
- `run_primitive_host`: Bool. If True a scf or geometry optimization calculation will be performed.
- `host_folder_data`: RemoteData object output of the calculation of the host structure if this was previosuly performed. It should be specified if `run_primitive_host` is set to False.
- `type_run_primitive_host`: if `run_primitive_host` you need to specify the type of *pw.x* run you want to perform for the host. Allowed values are: $'vc-relax'$, $'scf'$, or $'relax'$. Default is $'vc-relax'$.
- `type_run_defects`: you need to specify the type of *pw.x* run you want to perform for the host. Allowed values are: $'vc-relax'$, $'scf'$, or $'relax'$. Default is $'relax'$.
- `vacancies`: AiiDa List containg the specie of the host_structure for which we want to create vacancies. If clusters of the same vacancy type are to be formed the element should be repeted. By default no vacancies will be created.
- `substitutions`: AiiDA ParameterData object containing entries like $"Mn" : ["Ti","Co"]$ meaning that Mn in the host_structure will be substituted with either Ti or Co. By default no substitutionals defects will be created.
- `scale_sc`: scale parameter for the supercell creation. AiiDA List of the type [a,b,c], which will result in a (a×b< *times*c) supercell.
- `cluster`: AiiDA boolean. If true defective structures containing all the requested defects are created. Otherwise defective structures, each containing only one of the requested defect type, will be created. By default no clusters will be created.
- `defect_charge`: Float corresponding to the charge of the defect
- `epsilon_r`: Float corresponding to the dielectric constant of the host material
- `corrections`: ParameterData containg the type of corrections to the defect formation energy you wan to compute. By defasult no corrections are computed. Example: `corrections_default=ParameterData(dict= {'makov_payne':False,'bandfilling':False,'potential_alignment':True,})`

The output of the workchain is the node of the calculation of performed on the defctive structures and eventually of the host, together with the computed corrections.

The reader can find examples of the usage of this functions in the Jupyter Notebook $defect - workflow.ipynb$ in the `aiida_defects/examples_aiida_defects` folder.

[1] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M. Buongiorno Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A. Dal Corso, S. De Gironcoli, P. Delugas, R. A. Distasio, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J. Jia, M. Kawamura, H. Y. Ko, A. Kokalj, E. Kücükbenli, M. Lazzeri, M. Marsili, N. Marzari, F. Mauri, N. L. Nguyen, H. V. Nguyen, A. Otero-De-La-Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A. P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, and S. Baroni, J. Phys.: Condens. Matter. **29**, 465901 (2017), arXiv:1709.10010.

[2] G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, and B. Kozinsky, Comput. Mater. Sci. **111**, 218 (2016).

[3] C. Freysoldt, B. Grabowski, T. Hickel, J. Neugebauer, G. Kresse, A. Janotti, and C. G. Van de Walle, Rev. Mod. Phys. **86**, 253 (2014).

[4] C. Varvenne, F. Bruneval, M.-C. Marinica, and E. Clouet, Phys. Rev. B **88**, 134102 (2013).

[5] M. Leslie and N. J. Gillan, Journal of Physics C: Solid State Physics **18**, 973 (1985).

[6] G. Makov and M. C. Payne, Phys. Rev. B **51**, 4014 (1995).

[7] S. Lany and A. Zunger, Modelling and Simulation in Materials Science and Engineering **17**, 084002 (2009).

[8] S. Lany and A. Zunger, Phys. Rev. B **78**, 235104 (2008).

[9] Y. Kumagai and F. Oba, Phys. Rev. B **89**, 195205 (2014).

[10] T. S. Moss (1954) p. 775.

[11] E. Burstein, Phys. Rev. **93**, 632 (1954).

[12] R. Resta and K. Kunc, Phys. Rev. B **34**, 7146 (1986).

[13] S. Baroni and R. Resta, Phys. Rev. B **33**, 7017 (1986).

[14] X. Gonze and C. Lee, Phys. Rev. B **55**, 10355 (1997).

[15] C. Freysoldt, J. Neugebauer, and C. G. Van de Walle, Physica Status Solidi (b) **248**, 1067 (2011), https://onlinelibrary.wiley.com/doi/pdf/10.1002/pssb.201046289.

[16] T. A. Pham, T. Li, S. Shankar, F. Gygi, and G. Galli, Applied Physics Letters **96**, 062902 (2010), https://doi.org/10.1063/1.3303987.

[17] R. D. King-Smith and D. Vanderbilt, Phys. Rev. B **47**, 1651 (1993).

[18] P. Umari and A. Pasquarello, Phys. Rev. B **68**, 085114 (2003).

[19] I. Souza, J. Íñiguez, and D. Vanderbilt, Phys. Rev. Lett. **89**, 117602 (2002).

[20] S. P. Ong, W. D. Richards, A. Jain, G. Hautier, M. Kocher, S. Cholia, D. Gunter, V. L. Chevrier, K. A. Persson, and G. Ceder, Computational Materials Science **68**, 314 (2013).