Siesta caminando

Andrei Postnikov
Université Paul Verlaine – Metz
June 2009

For whom this reading

Vuestra merced came to an idea to try the SIESTA code – but is this a really good idea, and what to begin with? The present document intends to save you some pains of decision making, of trial, and of error. It does *not* replace basic reading on condensed matter physics, density functional theory, or SIESTA documentation.

What SIESTA is and isn't

SIESTA is a calculation method and computer code which solves problems at the level of density functional theory (DFT). These problems are, generally, related to ground-state properties. Energy/volume curves, phase diagrams (e.g., magnetic ones), phonons, molecular dynamics are all related to ground-state properties. Electron excitations are not. So if you need to calculate a good excitation spectrum you should better go for a good TDDFT code, or a method which implements the GW approximation.

As other DFT codes, SIESTA provides no access to many-electron wavefunction and deals with electron density which is in many aspects good and reasonable, *but* is a single-determinant density. Hence the effects of *Configuration Interaction*, accounted for in quantum chemistry methods, are not included, even insofar as ground-state properties are concerned! Extensions beyond "standard" DFT like hybride functionals, LDA+U, self-interaction correction are not included either. In the description of correct equilibrium geometry, even as the latter is a ground-state property, SIESTA has problems (albeit of fundamental character and shared by many other DFT methods) in correctly describing van-der-Waals interactions and hydrogen bonds.

In this sense, SIESTA falls, in what regards its general range of applicability, into what is also covered (apart from small variations in the list of individually calculable properties) by a number of existing DFT codes, including TBLMTO, WIEN2k, CRYSTAL, FPLO, VASP, ABINIT, PWSCF. Why should you use SIESTA for your purposes, and should you really?

Among the above listed realizations of DFT calculations, there are differences in practical realization, and, as a result of it, some codes are better suited for special types of problems and materials than the others. The first important distinction comes along the line all-electron vs. pseudopotential methods. TBLMTO, WIEN2k, CRYSTAL, FPLO are all-electron methods, which take into account, and re-adjust in every iteration, the wave functions of all electrons in each atom, from $1s$ upwards. On the contrary, SIESTA together with ABINIT, PWSCF, VASP and many others is a pseudopotential method. In such methods, only valence electrons (down to some depth, probably including so-called semicore states) are explicitly included in the equations, whereas deep core states is excluded from the treatment, thanks to the use of pseudopotential (or, projected augmented waves approach). This is usually accurate enough for describing chemical bonding, equilibrium geometry, phonons etc. However, the properties immediately related to core electrons, like e.g. hyperfine field at atom cores, isomer shift, electric field gradient are either

hopelessly bad, or simply non existent in SIESTA. If you are interested in calculating any of these properties, SIESTA is not your choice.

The second important distinction concerns the choice of basis functions, used to expand the solutions of the Kohn-Sham equation. Two big groups of methods are i) those which use atom-centered bases, usually relatively compact ones (of the order of $\sim 10^1$ functions per atom), and ii) atom-independent ones, the most typical example – plane waves. The number of plane waves in a calculation may easily go into thousands, but they allow simple evaluation of matrix elements, easy parallelization and many other advantages. Finally, there are hybride methods, which combine numerical accuracy in intraatomic regions with reasonable flexibility in the interatomic region. Speaking of the above methods – TBLMTO, CRYSTAL, FPLO use atom-centered bases, let's refer to them as "tight-binding" group of methods; ABINIT, PWSCF, VASP use plane waves, and the basis in WIEN2k is a hybride one. Which basis to use is a trade between accuracy, flexibility and efficiency. Planewave methods allow systematic improvement of basis by varying a single parameter (planewave cutoff), but the necessary number of basis functions and hence calculation time may, as the desired accuracy increases, grow very fast. In a planewave method, one pays for the volume of the periodic cell (as the number of planewaves, for a given $\mathbf{k}$-cutoff, scales with volume) and not directly for the number of atoms. Therefore a calculation for a low-density porous structure, or a molecule put in a sufficiently large simulation box, in order to prevent its interaction with its repeated/translated replicas, may become very expensive. Differently with atom-centered basis functions: the volume as such costs nothing (or nearly); the price to pay scales with the number of atoms (and number of basis functions on each atom). So if we managed to find a good basis which offers good variational freedom in the regions where it is most needed, i.e. where the chemical bonding takes place, a calculation may become very efficient. Unfortunately, there is no general rule how to construct such efficient basis functions, therefore a tuning of good basis sets becomes an issue, and often a source of artistic pride, in a corresponding tight-binding community. Enjoying a good atom-centered basis set, we can never be sure that we haven't missed the variational freedom is some other region of space (due to anomalies at a surface or defect, accumulation of charge at the chemical bond, or whatever else) where it was in fact badly needed, but haven't been thought about.

SIESTA falls into category of methods with atom-centered basis sets. It means that it is not that much afraid of big unit cell volumes, that for big systems it would make sense to optimize basis set in order to perform calculations within reasonable time, and that in sensible cases it makes sense not to trust the results immediately but make a test (whenever possible, on a reduced model system) against a planewave or hybrid-method calculation. Moreover we add that SIESTA does not use any symmetry in the calculated system whatsoever (apart from the time reversal symmetry for $\mathbf{k}$-points, allowing to use a half of the full Brillouin zone). So we cannot gain anything when the system is supposed to be in fact symmetric (and it comes out *never* exactly symmetric from the relaxation in SIESTA, some numerical noise is always present), but we either don't lose anything when the system prefers to develop large distortions, when we study vibrations, etc.

Now with this attribution done, what is special about SIESTA? What is its standing relative to other methods which use atomic-centered bases, e.g., Gaussian-type orbitals?

The specifics of different tight-binding schemes is how matrix elements are evaluated. The matrix elements needed are between two basis functions (for overlap) or between two basis functions with a potential, or nabla operator, or Laplace operator in between (for the Hamiltonian). Different realizations exhibit a broad spectrum of technical solutions. Either it is done analytically (Gaussian-type orbitals), that however demands a special shape of basis functions, or the shape of functions is not restricted, but special tricks are used for spatial integration (e.g., diophantine

random integration in the Discrete Variational Method). The convention in SIESTA is that basis function might be of *any* form, even drawn by hand and digitalized, but they *must* have a finite radial extension. (This applies, of course, to the radial component of basis functions, as the angular part is described by spherical harmonics, as in almost all other methods). A not necessary but a very reasonable additional condition is that the basis functions go continuously into zero starting from a certain radius; even better if they do *smoothly* go into zero. The evaluation of matrix elements is done in a tricky way, via transformation, tabulation and interpolation, efficient because the range of overlap of two given functions is limited. The range of overlap is ultimately defined by a human calculator, but typically it includes second / third coordination spheres around each atom, depending on the system.

What follows from this strategy: SIESTA keeps trace on overlapping of basis functions between atoms; if there are not so many neighbouring atoms (examples: organic molecules, 4-coordinated semiconductors), the calculation can be very fast. This is further helped by the fact that the number of $\mathbf{k}$-point needed for crystals with band gaps and not much pronounced energy dispersion is low. SIESTA is very forgiving in introducing, if necessary, a lot of vacuum around a molecule, a slab or an atomic chain, much more forgiving than plane-wave methods are. The vacuum comes not quite for free in SIESTA, but at a generously discounted price. On the contrary, close-packed metal systems are very hard cases for SIESTA, for two reasons: large number of overlaps to count in say, fcc or bcc structure, and a large number of $\mathbf{k}$-points needed to well describe a metal, probably with some flat bands crossing the Fermi level. You remember that SIESTA does not make use of symmetry, therefore a modest $10\times10\times10$ division of the Brillouin zone results in more or less 501 inequivalent $\mathbf{k}$-points! For such a system it can easily happen that an accurate all-electron method like WIEN2k would beat SIESTA by far in efficiency.

First steps: crystal structure, system type

In order to start a new calculation, you need to provide a single `.fdf` file, PLUS an extra pseudopotential file for each (chemically distinct) species involved. Pseudopotentials will be discussed later on. The main `.fdf` file is well documented in the SIESTA user's guide, has flexible format, and allows many options to be omitted and to be replaced by default settings. This is in fact quite convenient. In my opinion, there are only two worries about this whole convention. The first is, many entries are not required and constructed by default without duly informing the user what is going on. (So, you should carefully read the output file). A problem may happen when you misprint a parameter name, it was not duly recognized and was replaced by a default setting, whereas you are in good faith that you have properly declared what had to be declared. The second worry is that if the `.fdf` contains more than one declaration of the same parameter, only the first appearance is registered, and the following ignored. This might be easily overlooked by a user who desperately corrects an entry at the end of the `.fdf` file whereas there is the similar entry near the beginning which in fact takes effect.

The first good thing to know is that SIESTA *always* use a simulation box, like all bandstructure methods do. If one wants to calculate a molecule or a cluster, it is usually possible with any bandstructure method – one simply chooses the periodic unit cell large enough so that the spurious interaction between a molecule and its trnnnlated replicas can be neglected. (You remember that in planewave methods, an increase of the cell size, even empty, costs you computer resources – memory and calculation time). The interaction between translated fragments, however small, give rise to energy band dispersions. The larger the cell – the weaker the interaction – the smaller the

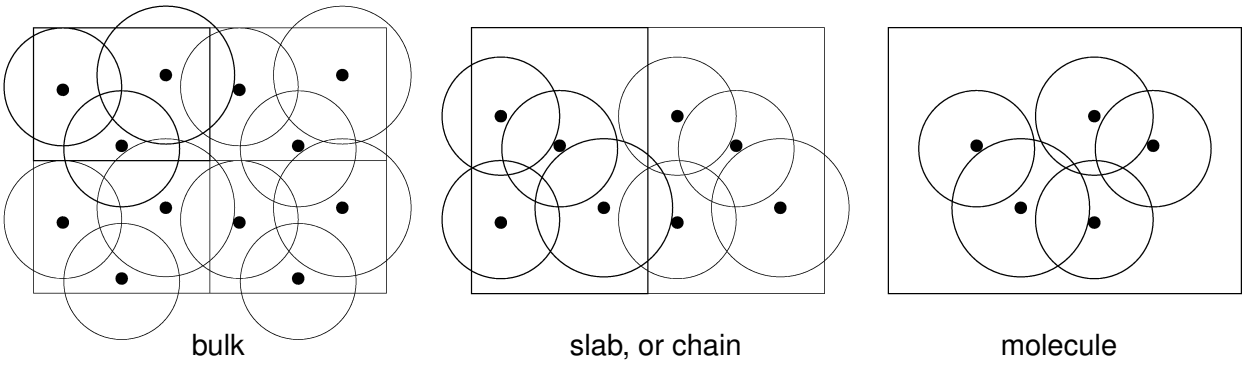

Figure 1: Types of structure automatically recognized by SIESTA. Periodic unit cells and the extension of basis functions centered at atoms are schematically shown.

band dispersion, which ultimately can be, for practical purposes, neglected, and calculation done with a single $\mathbf{k}$-point. So far the general observation. The peculiarity of SIESTA is that, since the basis functions are *strictly* confined, the interaction between them becomes, from a certain distance on, *exactly* zero, so the further increase of the cell size does not change anything[1]. Hence the band dispersion is *strictly* zero; you can calculate with a single $\mathbf{k}$-point or with many – the result will be the same. Therefore SIESTA allows to treat *on the same footing* 3-dimensional periodic systems (crystals), 2-dimensional periodic ones, cut in the third dimension (slabs), 1-dimensional periodic and finite in other two dimensions (chains) or 0-dimensional (finite in all three dimension; clusters or molecules) systems. The distinction between these cases is done automatically, judging by whether the basis functions "overlap across the unit cell boundary" in 1,2,3 dimensions. The result of this analysis is reported someplace in the main output file at the beginning of calculation, as a message

```
siesta: System type = bulk
```

or

```
siesta: System type = molecule
```

etc. So it is good to check whether this identification corresponds to your expectations. How can SIESTA misinterpret your definitions? There are many ways. Note that the positions of atoms

```
%block Atomic_Coordinates_and_Atomic_Species
 ...
%endblock Atomic_Coordinates_and_Atomic_Species
```

must be defined in the `.fdf` file, SIESTA is not *that* intelligent to guess them. However, the lattice parameters are not a *must* entry; hence if they are missing, SIESTA will apply its own logic. It will admit that your *System type* is "molecule", it will measure the size of your "molecule" (i.e. min-max spread of whatever is in your "atomic coordinates" block along each of X,Y,Z directions), it will add the spread of the basis functions on corresponding atoms, it will add some 10% extra for safety, and with thus maximal obtained value along each of three Cartesian directions it will construct a cubic box around your atoms. So better check if *that* is really what you need. Personally I believe it is something you'd never really need, because if you are calculating a molecule, you may wish to reserve some freedom for structure relaxation, or for other distortions of your molecule, and

[1] or nearly...

it seems more reasonable to fix the size of the simulation box by hand, simultaneously fixing by this the real-space mesh (see below) etc., than let SIESTA to vary it *ad hoc* with the shape of the molecule. How to estimate the size of molecule by the eye, finding min and max over thousands of coordinates of atoms? A trick owed to Gustav Bihlmeyer: read the list of coordinates as (X,Y,Z) table into gnuplot or Grace and plot it "with lines" as Y(X) and then as Z(X)...

Apart from this molecule/simulation box problematics, even if you think that you declared the `Lattice_vectors` or `Lattice_parameters` block explicitly and you are in the clean, there is always a possibility to mess up with the units. The chances to make something wrong are: to define `LatticeConstant` in Bohr instead of Å or vice versa or forget it whatsoever, in which case SIESTA will figure out the value herself (as the cubic box size above); declare atomic coordinates as `ScaledCartesian` whereas in fact your numbers belong to `Fractional`, feed them in as if they were in Bohr, or many other nice possibilities. The only rocksolid way to check that SIESTA understood your input correctly is to look at the `.XV` file (produced by SIESTA after the first CG step; also if zero steps were demanded). This file contains all structure information (simulation box + atoms) in a fixed format, as it was internally recognized by SIESTA, it can be easily transformed into XCrySDen format for visualization, and could (should) be looked at.

Another friendly default definition is for $\mathbf{k}$-points: if they are forgotten to define, the default `kgrid_cutoff` is zero and hence Γ point only is calculated. A surprise for whoever might have in mind calculating a metal...

Basis

If you don't know better, the standard SIESTA setting – DZP – is quite reasonable, so you can go ahead with it. The only problems appears when your pseudopotential includes semicore state, and SIESTA demands you to explicitly describe them in the basis. This is indeed very simple, here is an example for Cr with double-ζ $3p$ semicore, double-ζ $4s$ with polarization orbitals, and triple-ζ $3d$:

```
%block PAO.Basis
 Cr  3   0.0
   n=4  0  2  P
     0.0  0.0
   n=3  2  3
     0.0  0.0  0.0
   n=3  1  2
     0.0  0.0
   ...
```

The resulting basis set is documented in the output file; notably the figure of its maximal extension (confinement radius) for each atom is hidden behind a quite misleading title like

```
Vna:  Cut-off radius for the neutral-atom potential:   6.250681
```

in Bohr units. Note that the extension of basis could be needed for estimating the box size in the treatment of a molecule. The `PAO.EnergyShift` parameter, as described in the SIESTA literature, influences the confinement of basis: the larger the EnergyShift, the stronger the confinement. Note that the files with basis functions are explicitly generated and can be directly plotted, e.g. the beginning of the file called `ORB.S3.2.Ga` looks like:

```
# Ga  2  3  2  0    0.0000
#(species label, l, n, z, is_polarized, popul)
      0.00000000000          16.7721119042
     0.307593796936E-02      16.7717020784
     0.615187593872E-02      16.7704470847
     0.922781390808E-02      16.7683555217
     ....
```

which is hardly publishable in a paper but which (the plot) could add a touch of credibility to your report or a PhD work :-) An extensive literature in the basis set generation in SIESTA is aimed at obtaining bases with less functions yet not inferior quality. This can be achieved by using auxiliary Simplex code.

An arguable weak point of SIESTA bases is that they are fixed – same as most of Gaussian-type orbital packages – and not adjustable from iteration to iteration, as in hybride methods like LAPW or LMTO. As a consequence, one cannot *a priori* expect equally good performance of basis over a large range of bond lengths and, say, pressures: as the pressure is varying, the "completeness" or "variational freedom" of basis in the region of chemically relevant, under a given pressure, bond lengths do vary; consequently the calculated pressure/energy curve may slightly differ from that obtained in a "benchmark" FLAPW calculation. In any case, special care must be taken with respect to this issue. I repeat: be careful when exploring large volume variations with SIESTA.

Pseudopotentials

If you don't know better, you'd probably look for a suitable pseudopotential in the SIESTA database, Abinit→SIESTA conversion database, or issue a desperate cry into the mailing list. It won't always help, so from time to time you'd need to generate pseudopotentials on your own. It is not so difficult as a procedure, but the criterion of "good" pseudopotential is difficult to formalize; that's why the construction of pseudopotentials has an aura of art and mystery. The `atom` package within SIESTA is well documented; I'd outline here only essential steps. Note that a reasonably detailed documentation *is* available in ⟨SIESTA-VERSION-DIRECTORY⟩`/Pseudo/atom/doc/atom.tex`

The construction of pseudopotential comprises several desicions, apart from the more general choice of the exchange-correlation potential (LDA or GGA relativistic or not) and the flavour for the pseudopotential construction (Kerker, Troullier–Martins etc.), namely 1) electron configuration; 2) pseudoization radii; 3) use or not of the core correction.

1) The choice of configuration means which number of electrons to take in valence s, p, d, f states. As a common sense, this configuration should be representative for a system in question. The nominal free-atom configuration is usually a reasonable first choice; for transition metals, a common variation is the $s \rightarrow d$ promotion, e.g., $3d^7 4s^1$ for Fe, instead of the nominal $3d^6 s^2$. Another variation can be trying an ionic configuration – an anionic one for what is expected to be more electronegative and cationic one for whatever is likely to lose its electrons in a compound. Note that the choice of atomic configuration for the pseudopotential *has nothing to do* with doing calculation for a charged system, or with enforcing certain charges on the atoms in the system under study. The distribution of electron density between the atoms will come out of the band structure calculation, by itself, whereas the different charge the pseudopotential generation will merely yield a little bit different pseudopotential, that's all.

2) The pseudoization radius r_c, defined in each l-channel separately, is the radius beyond which the pseudopotential equals the true potential and the pseudo-wave function matches the

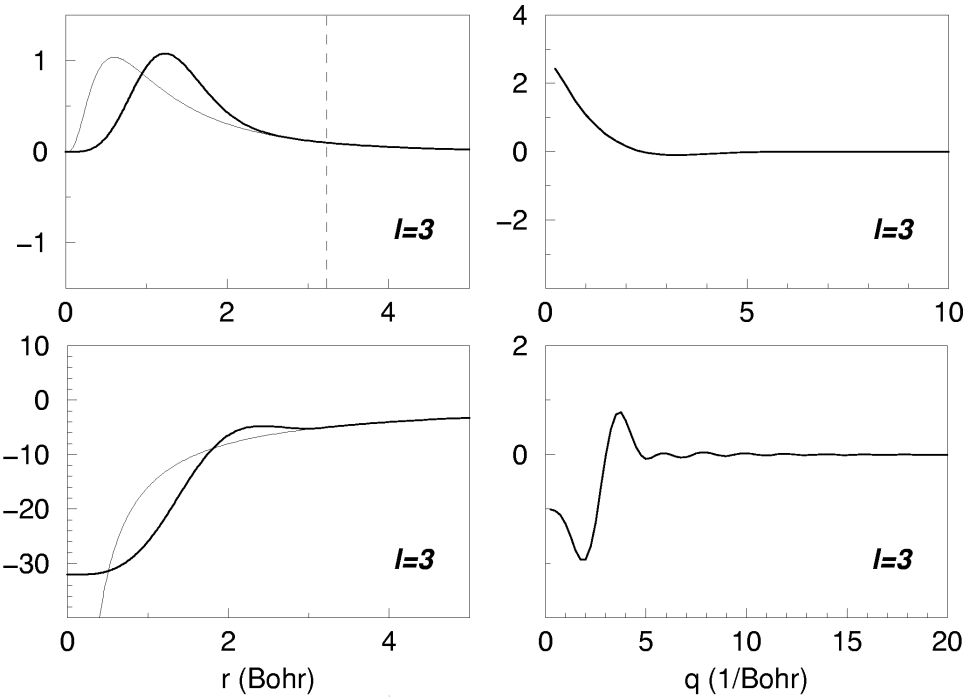

Figure 2: Pseudopotential generation with r_c=3.2 Bohr. Top left: All-electron (thin line) and pseudo (thick line) functions; top right: Fourier transform of the pseudo function. Bottom left: true (thin line) and pseudo (thick line) potential; bottom right: Fourier transform of the pseudopotential.

true ("all-electron"[2]) function. Inside the r_c, the pseudofunction will be constructed such as to have *no nodes* (in contrary to all-electron function which would have $n - l - 1$ nodes), but the same norm as the all-electron one. Our aim is to achieve a faithful description of the maximum of the pseudofunction, responsible for the chemical bonding and at the same time we'd like to have a "well-behaving" pseudopotential whose fluctuations are well localized in both real an reciprocal space.

Figures 2 and 3 show the pseudopotential and corresponding pseudofunction in the f channel, along with their all-electron counterparts, for a fictive (not to follow in real calculation) case of Sm in the $6s^2 6p^0 5d^0 4f^6$ configuration. They illustrate a general trend which will apply for every element and any l value. Setting r_c too large as in Fig. 2 would result in a too weak constraint imposed on the shape of pseudofunction, so that its radial behaviour inside r_c may become too different from that of the all-electron function. On the contrary, setting r_c too short as in Fig. 3 may result in too strong constraint on the pseudopotential, leading to its too large fluctuations.

As a rule of thumb, choose the r_c just beyond the last maximum of the all-electron function. But if you set it more outwards and see that the behavior of the pseudofunction is still OK, you may wish to accept this higher radius. On the contrary, for diffuse outer functions, whose maximum is well beyond the atomic radius, the r_c does not have to be related to this maximum and can be set at smaller value. You still follow? Look at the shape of pseudopotential along the real-space and reciprocal radial axes. Both functions must be smooth, well-balanced, pleasant to the eye,

[2]in a sense that it comes from a calculation of an atom with its all electrons, from $1s$ upwards. But this is of course still one-particle wavefunction, that of Kohn–Sham equation, and by no means a many-particle one. – Just to avoid a possible confusion.

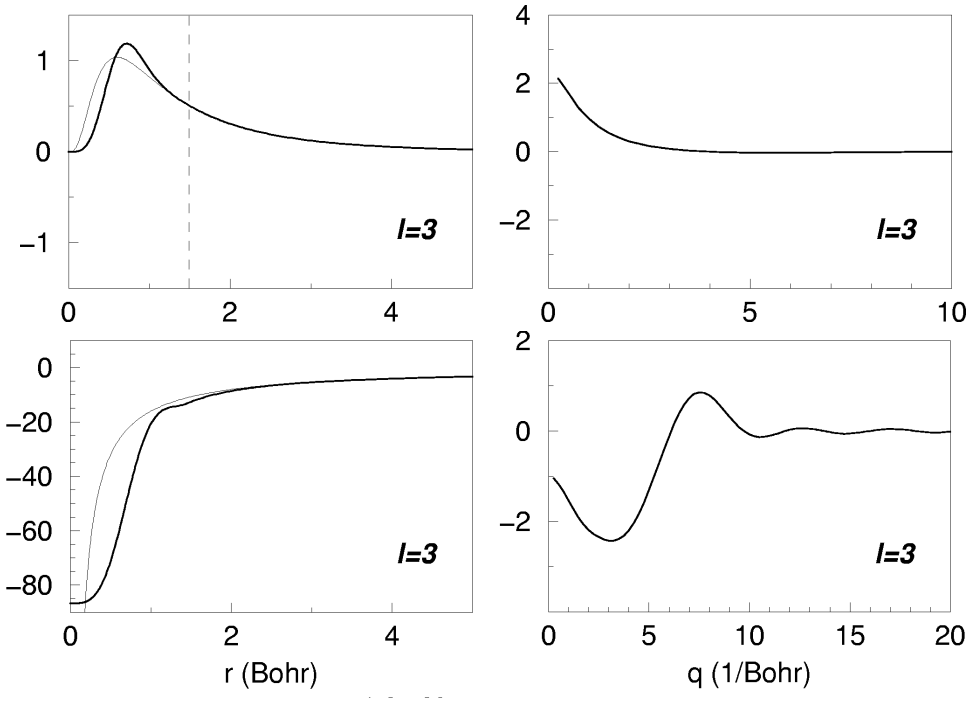

Figure 3: The same as Fig. 2, for r_c=1.5 Bohr.

Zen-like. A good pseudopotential is a nice pseudopotential. You see now that it is difficult to algorithmize?

Finally comes the testing. Mind that any pseudopotential is generated with just one specific atomic configuration, but it (pseudopotential) is supposed to be *transferable*, i.e. provide an adequate description of an atom if the valence electronic configuration of the latter is varying in reasonable limits (shuffling electrons here and back between different l-channels). You should check a number of different trial configuration, calculate atom with a chosen pseudopotential for these trial configurations, and compare the results with the all-electron ones, obtained for the same trial configurations. Which trial configurations to take? Those which are probable to come about in a compound. Taking an all-electron calculation of these different configurations as a reference, one should check whether the pseudopotential calculation yields (i) good energies of valence states, and (ii) relations between total energies of different trial configurations. The absolute total energies in all-electron and pseudopotential calculation, also between different pseudopotentials, will be very different, but the relative energy differences (called cross-excitations) between different configurations within each scheme, either all-electron or pseudopotential, must be close. These energy differences are produced in a form of triangular table at the end of both all-electron and pseudopotential-test calculations, and these tables have to be compared. Have a look at the documentation accompanying several personally uploaded pseudopotentials on the SIESTA web page, this will give you an idea how the tests should be run. At last,

3) the importance of core correction. There are general criteria which tell us that this correction is important when a non-negligible overlap occurs between core and valence functions. In principle the effect of core correction can be explored is a systematic way – adding it and looking at how the energy characteristics (valence-state energies and "triangular table" of total-energy differences) will be affected – in a noticeable way, or not.

k-mesh

This seems to be an easy topic, and the one not much different in SIESTA, as compared with other bandstructure codes. Moreover it is well documented in the main SIESTA paper and (basics of) in the Moreno+Soler PRB paper of 1992. Still, some remarks for impatient readers may be in place:

1) The **k**-mesh with more than one point is needed only for periodic system where there is dispersion of energy bands; for molecules one **k**-point is just fine. Note that exactly this case (**k**=0 only) is the SIESTA default. So be careful and define **k**-mesh explicitly if you do a periodic system.

2) There are different methods of integration over **k**-space available on the market; what is implemented in SIESTA is the Monkhorst-Pack sampling only. It is not particularly bad, but personally I'd prefer to have a tetrahedron method implemented as an addition option, which won't be that difficult to do.

3) The **k**-mesh may be either shifted, or not. The default case is single Γ point (0. 0. 0.) and hence not shifted. The **k**-mesh defined by a single `k-grid cutoff` parameter is generated shifted, for example the input entry

```
 kgrid_cutoff    16.0 Ang
```

in combination with a certain lattice vectors may lead to the following statements in the output:

```
siesta: k-grid: Number of k-points =   500
siesta: k-grid: Cutoff             =    16.041 Ang
siesta: k-grid: Supercell and displacements
siesta: k-grid:   10   0   0      0.500
siesta: k-grid:    0  10   0      0.500
siesta: k-grid:    0   0  10      0.500
```

Here the number of divisions (10×10×10) was estimated from the `kgrid_cutoff` of the input, and the resulting cutoff (16.041 Å) is then adjusted (upwards) from the actual accepted integer numbers of divisions. In the ⟨SystemLabel⟩.KP file we find the explicit list of **k**-points (in an obscure order, with plain cartesian coordinates in the units of 1/Bohr, the last column being the weight attributed to each point):

```
   500
     1    0.036647    0.036647   -0.549630       0.002000
     2   -0.036638    0.109931   -0.476346       0.002000
     3   -0.109924    0.183215   -0.403062       0.002000
     ...
   499   -0.329784    0.989339   -0.329784       0.002000
   500    0.329784    0.329784   -0.989339       0.002000
```

The above list is obviously shifted, because none of the points has zero coordinates. An unshifted list always includes the Γ point. The "unshifting" can be enforced by directly giving the number of divisions, e.g.

```
%block kgrid_Monkhorst_Pack
 12  0  0   0.
  0 12  0   0.
  0  0 12   0.
%endblock kgrid_Monkhorst_Pack
```

For a simple cubic cell, this definition would mean that a cubic Brillouin zone is divided with a step $(\frac{2\pi}{a})/12$.

What is the main difference between shifted and unshifted mesh, and which one to choose? As unshifted mesh explicitly contains the Γ and other symmetric points, it is likely to well represent the band edges – maxima and minima – which might be useful for directly reading the band gaps (obviously wrong in the straight DFT, but anyway) from the calculation. Whereas the shifted mesh misses the band extremities, but is somehow more economic (less inequivalent $\mathbf{k}$-points than in unshifted mesh, at the same mesh density). Basically you should check the convergence of your relevant results with the mesh density, whether shifted or unshifted.

4) Mind that the definitions for energy band plotting, e.g.

```
%block BandLines
  1  0.00000   0.000000  0.000000  \Gamma
 40  2.00000   0.000000  0.000000  H
 28  1.00000   1.000000  0.000000  N
 28  0.00000   0.000000  0.000000  \Gamma
 34  1.00000   1.000000  1.000000  P
%endblock BandLines
```

have nothing to do with the $\mathbf{k}$-mesh. This block describes at which $\mathbf{k}$ points the band energy must be calculated for making a nice (smooth) band structure plot, hence for purely illustrative purposes. This won't affect the quality of your results, previously converged (or not) with a certain $\mathbf{k}$-mesh.

Basis, pseudopotentials, MeshCutoff, $\mathbf{k}$-mesh: what does really matter and how to manage them all in the same time?

SIESTA does not allow, by turning something in basis or pseudopotentials, to tune the results to "whatever you want". This can be considered, generally, as a good news. Usually a reasonable result is there in place already with relatively low level of SIESTA's *ingeniosidad*. The bad news are, it is sometimes very difficult to improve initially good but not yet perfect result. Well, at least we should be consistent in what we are doing and check the convergence of whatever is convergent. In total there are four important pieces of the SIESTA machinery: pseudopotentials, basis, $\mathbf{k}$-mesh and MeshCutoff. (One can be tempted to play around with exchange-correlation flavour, but this is not what we are discussing here). Let's establish some systematics between the four factors.

1. Pseudopotentials should be prepared in the best way we can judging by pseudopotentials' intrinsic criteria (see above) for the system in question (e.g. probably different for oxide than for metal system), and otherwise left in peace. However, sometimes it becomes evident only from the SIESTA calculation that certain states, initially attributed to core, must be treated as valence. It usually becomes clear from markedly wrong results for elastic properties – bulk modulus or Γ-phonon in benchmark calculations, whereas the electronic structure looked OK. In this case, the potential must be generated anew for a different atomic configuration, and the other tests (see below) repeated.

2. Basis set should be prepared in a reasonable and responsible way, but otherwise it remains the author's choice. Within the given class and size of basis chosen (DZP vs. soft-confined vs. whatever else), some optimization of parameters is possible (using e.g. the Simplex program), but *the choice of class and size is not subject to optimization*. Meaning, this is the author/user who ultimately decides whether he/she would trade some accuracy for efficiency, or other way around.

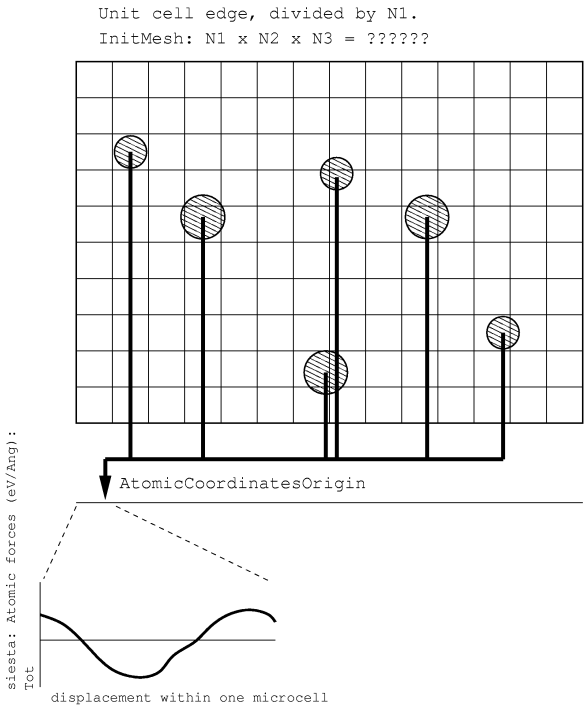

Figure 4: An analysis of the *eggbox effect*. Each edge of the unit cell is cut by the number of divisions convenient for fast Fourier transform. The number of divisions depends on the MeshCutoff and can be found in the output file in the ``InitMesh:'' line. The test consists of applying the uniform displacement of all atoms, using the AtomicCoordinatesOrigin parameter, and observing the numerical noise in the summary force over all atoms in the unit cell. The three Cartesian components of this summary force, which ideally must be zero but in fact fluctuates around zero as a function of uniform displacement, can be found in the output file, in the line following siesta: Atomic forces (eV/Ang):. The fluctuations of summary force are periodic with the edge of a microcell (unit cell divided by the number of mesh divisions), and their amplitude does systematically decrease as ever larger MeshCutoff is taken and, consequently, the number of mesh divisions increases. In order to have reliable forces in a calculation, one should check that the magnitude of fluctuations of the summary force throughout the microcell remains inferior to 0.1 eV/Å.

3. $\mathbf{k}$-mesh is as in any other band-structure code: we should check the convergence, in terms of gradually increasing the $\mathbf{k}$-mesh density (i.e., in fact increasing the kgrid_cutoff parameter) NOT OF THE TOTAL ENERGY but *of those properties which are relevant for our study in question*: the energy difference between two phases, the phonon frequency, etc. We should not bring the $\mathbf{k}$-mesh density to overkill (because dense $\mathbf{k}$-meshes are expensive in SIESTA), but we should be serious about this convergence, especially when dealing with a metallic system. SIESTA does not provide tetrahedron integration, and fine fluctuations of the density of states in the vicinity of the Fermi level might be difficult to converge.

4. MeshCutoff is a peculiarity of SIESTA; it scales the calculation resources (memory, time) with the cell size, independently on the number of atoms – in about the same way as it happens in planewave codes. However, its role is different. The planewave cutoff in planewave codes affects the completeness of basis. In SIESTA the basis is given independently of MeshCutoff, but the latter affects the numerical noise in the Fourier transformation of the charge density. This noise is translated into the noise in the total energy and, especially, forces. The noise mean fluctuations of the respective property (i.e., total energy or forces) depending *on the uniform displacement of all atoms with respect to the unit cell*, which is referred to as *eggbox effect*. This is indeed something very peculiar of SIESTA, let's think about it: no symmetry is taken into account, no special atomic positions exist, atoms are absolutely decoupled from the reference system of lattice vectors. In real world, the drift of crystal as a whole with respect to a fixed coordinate system would cost no energy and induce no force. However, numerically there may be variations, periodic with the step of the mesh grid. Suppose that you have a lattice vector of the length, say, 8 Å, and your MeshCutoff is such that produces 64 divisions along the lattice vector in question (the actual number of divisions stands in the output file, in the InitMesh: ... line). Then, as you smoothly shift *all* atoms of your system along the direction of the said lattice vector (technically this can be achieved by variying the entries in

the Atomic_Coordinates_Origin block), you'll see fluctuations of calculated total energy and forces with the periodicity of 8 Å/64 = 0.125 Å. In order to explore these fluctuations, you'd scan this interval (or slightly beyond it, for curiosity) with a sufficient number of steps to describe a periodic behaviour. Let's say take 16 or so displacements, that will mean a step of roughly 0.008 Å in varying the Atomic_Coordinates_Origin. Seems a small and negligible number, but it offers a view into the microscopics of the SIESTA numerics, so every SIESTA user should try it once to understand how the "eggbox effect" works. I explain it in detail because it is a sensitive issue: you prepare a sequence of jobs, differing only in the values of Atomic_Coordinates_Origin, e.g. (for the case that you have Atomic_Coordinates_Format Ang) like in the following:

```
0.000  0.000  0.000
0.000  0.000  0.008
0.000  0.000  0.016
...
0.000  0.000  0.128
```

and otherwise completely identical. (Make sure that MD.UseSaveXV is set to F, otherwise you'll always read the structure information from the same .XV file, overriding the above definitions). As the jobs are through, you mark down the total energy values and forces, as e.g. in my old test for GaAs (where the displacement is taken along the x direction and not along z as in the above example):

X-disp	E_KS	Fx_(Ga)	Fx_(As)	Sum(Fx)
0.00	-4407.6258	-0.003054	0.000283	-0.002771
0.03	-4407.6263	-0.005995	0.043142	0.037147
0.06	-4407.6275	-0.005154	0.084228	0.079074
0.09	-4407.6284	-0.001510	0.064327	0.062818
0.12	-4407.6291	0.006985	0.028604	0.035589
0.15	-4407.6293	-0.008506	-0.008077	-0.016583
0.18	-4407.6288	-0.008196	-0.039131	-0.047327
0.21	-4407.6278	0.002202	-0.065475	-0.063274
0.24	-4407.6269	0.002043	0.070230	-0.068187
0.27	-4407.6260	-0.000245	-0.023124	-0.023369
0.30	-4407.6259	-0.005478	0.025734	0.020256

What you should be careful about is that fluctuations of the summary force (in the last column above) stay within 0.1 eV/Å or so, otherwise the calculated phonons and elastic properties might be useless, and the relaxation won't be good either. The amplitude of fluctuations can be *systematically* pushed down by increasing the MeshCutoff. Note that the period of fluctuation is diminishing as MeshCutoff and, consequently, the number of divisions grow. The forces as such (the third and the fourth columns of the above example) are of course not *a priori* obliged to be small, unless your system is at equilibrium. But still, they should remain stable within the accuracy of $\sim$0.1 eV/Å.

How to continue smoothly and which files are really needed

In the course of calculation, SIESTA creates a lot of files, some of which are quite large (e.g., ⟨SystemLabel⟩.P? files containing mixing information). The larger part of them is created anew,

so there is no need to keep them once the calculation is done (or, stopped). It makes sense to save, or backup from time to time, only the following files:

⟨SystemLabel⟩.DM – the density matrix. This is the main information which is updated from one iteration to another, and, as a difficult calculation is converged, it makes your most important capital (for further refining the calculation).

⟨SystemLabel⟩.XV – the crystal/atomic structure information, which is updated in the course of CG relaxation, or molecular dynamics.

⟨SystemLabel⟩.ANI (or other dynamic files) – if you are doing molecular dynamics and store its history.

The density matrix, which is sparse, is stored in a specific (yet undrstandable) format, giving a list of who (which basis orbital) overlaps with whom, and what are the occupation numbers associated with each overlap. The .DM is quite robust in a sense that you can smoothly continue calculation after changing MeshCutoff, $\mathbf{k}$-mesh, ElectronicTemperature – none of these parameters is explicitly stored in .DM. When doing such changes, however, be sure to remove the .P1 and .P2 files, which are the mixer files affected by the history of previous iterations. As you change the above calculation parameters, the "path of history" is about to change, and relying on previous mixer information may extrapolate/shoot your calculation in a completely inpredictable way. Better to start from "semi-good" DM smoothly and forget the history of how we arrived at it.

The same robustness of the .DM is valid, within some reasonable limits, as we slightly change the crystal structure: converge one volume and take the resulting DM as a hopefully good start for a slightly different volume, or if we otherwise introduce a small distortion.

What you cannot change "on the fly", however, is the basis set. If you touch it, the whole system of numbering the non-zero elements in the DM will be damaged; SIESTA will complain about it (expected to find so many elements in the DM file... but found only so many) and stop. The same problem will occur if the atomic structure was so changed from the previous step that the previous system of overlaps between basis functions is no more valid. The only quant of solace then can be that the structural information we thus arrived at is more valuable than the density matrix, so that the latter can be sacrificed (renamed/erased), and the calculation continued from this last structure and anew overlapped atoms.

A simple set of tools for manipulating the .DM file available from
`http://www.home.uni-osnabrueck.de/apostnik/Software/DMtune.tar.gz`
allows to make a reasonably good continuation from already available density matrix, avoiding start from scratch, in cases like when a non-magnetic case has to be expanded into magnetic one, spin on certain atom(s) inverted, etc.

A word of caution

You may be tempted to look just at final results you are interested in (total energy for the equation of state curve; magnetic moments for a magnetic system; phonons; ...) and ignore the rest of the output data, unless the code explicitly complains about something. It is more important than you'd *a priori* think to get a clear idea about the electronic structure, whether it comes about as expected. Please make an effort to construct, and to look at, good old band dispersions and densities of states (at least once for every new structure under study), think about whether the placement of different bands is compatible with chemical intuition. If your system is supposed to have a band gap (even if obviously wrong as in DFT), does it really have it? If your system is oxide, do you have O$2s$ band in its usual DFT place, at 17-19 eV below O$2p$? Are the semicore bands

reasonably placed? These evident checkups would make sense when using any band structure method, but SIESTA may have some specific reasons for misconduct. To name just one: basis functions, when less extended than they should have been, may result in missing overlaps with neighbours and spurious lack of dispersion.

My favourite SIESTA traps

1. The 100-atom trap:
by default, the `SolutionMethod` is `diagon` for $\leq$100 atoms and `OrderN` otherwise. If you don't know which `SolutionMethod` is good for you, the answer is `diagon`: it is never wrong, even if sometimes les efficient than `OrderN`. In fact `OrderN` is quite fragile: it doesn't tolerate more than one $\mathbf{k}$-point, needs large band gap, etc. So better leave `OrderN` to the case when "you know what you are doing". Yet this option is activated by default as soon as you constructed a cell with more than 100 atoms! In the best case, the calculations stops immediately, complaining about more than one $\mathbf{k}$-point; in a worse case (you've have one one $\mathbf{k}$-point already) it continues a bit, crashing later with an obscure message. A solution: set `SolutionMethod diagon` explicitly.

2. `UseSaveData` and frozen phonon calculation:
As the frozen phonon calculations (`MD.TypeOfRun FC`) do usually proceed after an achieved CG relaxation, it is reasonable to start from a converged density matrix which is available. However, the option `UseSaveData T` implies not only the use of the available `.DM`, but – if available – of the `.XV` as well. The `.XV` file contains the geometry after the relaxation, so it seems reasonable. However, the `.XV` is updated in *each* geometry step, including each atom displacement in the frozen phonon run. What may happen: your phonon calculation stopped or crashed at some point, and you want to continue. Yet the available `.XV` file is not that for the equilibrium geometry anymore, but that after the last successful FC displacement. And as you continue, you start a series of displacement from this new reference point, which is not quite compatible with the previously collected data. A solution: after completion of the CG run, freeze the relaxed coordinates into the `.fdf` file of the phonon calculation, set there

```
UseSaveData        T
MD.UseSaveXV       F
```

and stop/continue frozen phonon calculations as you wish.

3. Initializing magnetic calculation:
In order to start spin-polarized calculation, you have to set `SpinPolarized T`. If nothing else is defined, all spins will be set parallel and to maximal possible value on *all* atoms (ruthlessly including all oxygens, carbons etc. of your system). This might be not exactly what you have in mind. For keeping control over the initial magnetic configuration, you should use the block `DM.InitSpin` setting *anything* (usually just the "+" or "$-$" entry suffices for a good start) on those atoms only which have to be magnetic.

The author bears no responsibility for nothing,

but please send your bug reports and suggestions of improvement to `postnikov@univ-metz.fr`