TRIQS TUTORIAL
VICOM School Vienna

Markus Aichhorn

February 2017

1 A few words on TRIQS

TRIQS is a software package, released under GPL, that allows to do efficient numerical simulations using dynamical mean-field theory and related approaches. The source code can most easily be downloaded form github

`https://github.com/TRIQS/`

where you will need most probably the *triqs*, *cthyb*, and *dfttools* packages. For help with the installation, follow the guide on the main triqs web site, https://triqs.ipht.cnrs.fr/.

1.1 Basic Philosophy

The main idea of TRIQS is to provide

1. a base layer, written mainly in C++, with all the necessary routines and libraries.

2. a python interface that allows for lightweight programming and combination of different modules and libraries.

We will use only the Python interface for our purposes. Python is a very easy-to-use scripting language, it can be started by typing

```
[training@n41-001 ~]$ python
Python 2.7.11 (default, Nov  8 2016, 09:26:25)
[GCC 4.9.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

It is built on the concept of modules that can be loaded and used. For instance, if you want to do numerical calculations you most probably will use the NumPy package. You can load it by

```
>>> import numpy
```

and try to initialize a 3-by-3 matrix with zeros:

```
>>> A = numpy.zeros([3,3])
>>> print A
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
>>>
```

You see, there is no explicit declaration of variables in python, so be careful when typing, in particular with capital letters (A and a are different)! For further information on python and its usage, I have top refer to the online resources. A very good starting point is

```
https://docs.python.org/2.7/tutorial/
```

Note that we are using python 2.7, and not python3, which has some changes in the syntax and usage.

In simple terms, TRIQS is also coming as a python module. In order to have the location of this module automatically loaded into the PYTHONPATH environmental variable, we have an executable *pytriqs* instead of *python*. TRIQS is a rather big library, that is linked to a quite large number of other libraries. In order to set all the relevant paths and use TRIQS, we provide a shell script that does it for you:

```
[training@l33 ~]$ source init_triqs.sh
  Unloading intel/16.0.0
```

```
  Unloading intel-mpi/5
  Unloading intel-mkl/11.3
  Unloading elpa/2015.11.001
  Unloading xcrysden/1.5.60
  Unloading vesta/3.4.0
  Unloading suitesparse/4.4.1
  Unloading octave/3.8.2
  Unloading gcc/5.3
  Unloading gnuplot/5.0.5
  Unloading vasp/5.4.1.05Feb16
  Unloading python/2.7
  Unloading gobject-introspection/0.10.8
  Unloading numpy/1.9.1
  Unloading pycairo/1.8.8
  Unloading pygobject/2.28.4
  Unloading pygtk/2.24.0
  Unloading p4vasp/0.3.29
  Unloading grace/5.1.25
  Loading gcc/5.3 from: /opt/sw/x86_64/glibc-2.17/ivybridge-ep/gcc/5.3.0/
  Loading python/2.7 from: /opt/sw/x86_64/glibc-2.17/ivybridge-ep/python/2.7.11
  Loading intel-mkl/11
  Loading numpy/1.9.1 from: /opt/sw/x86_64/glibc-2.17/ivybridge-ep/numpy/1.9.1/
  Loading scipy/0.18.0 from: /opt/sw/x86_64/glibc-2.17/ivybridge-ep/scipy/0.18.
  Loading openmpi/1.10 from: /opt/sw/x86_64/glibc-2.12/ivybridge-ep/openmpi/1.1
  Loading intel/14.0.2
  Loading grace/5.1.23
  Unloading intel/14.0.2
  Loading gnuplot/5.0.5 from: /opt/sw/x86_64/glibc-2.17/ivybridge-ep/gnuplot/5.
```

We can no try to start TRIQS:

```
[training@n41-001 ~]$ pytriqs
Python 2.7.11 (default, Nov  8 2016, 09:26:25)
[GCC 4.9.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

which ends at the python prompt. For instance, the functionalities of the Greens function class of TRIQS can now be loaded as

```
>>> from pytriqs.gf.local import *
```

We can now check that there is a Matsubara Green function class available:

```
>>> GfImFreq
<type 'pytriqs.gf.local.gf.GfImFreq'>
>>>
```

In this tutorial, you will not have to write your own python scripts from scratch for the calculations, but can start with predefined scripts and try to play around with them. If you want to learn the details of how to use all the TRIQS modules, please visit the tutorials on the TRIQS website https://triqs.ipht.cnrs.fr/.

By the way, you can exit the interactive python/pytriqs interpreter by CTRL-D.

1.1.1 Running scripts

If you want to run some python code that you previously wrote to a file, it is executed by

```
training@n41-001 ~]$ pytriqs look_inside.py
```

This executes the script and exits the interpreter, you get back to the shell prompt.

1.1.2 Peculiarities on VSC3

Note that the script that loads the TRIQS modules *unloads* Wien2k and VASP modules, for some compiler library reasons. So if you want to go back to either Wien2k or VASP, you have to logout/login again.

Please do all connections with *ssh* using the X11 forwarding (flag -X)!

2 Step 1: The DFT calculation

The first part of every DFT+DMFT calculation is of course the DFT calculation. We will study NiO here, and will restrict ourselves for the time being to the paramagnetic case. DMFT is complicated enough...

Ideally, you have already a converged Wien2k calculation for NiO fcc. In order not to screw up your previous tutorials, it is highly recommended to do everything in the triqs subdirectory for this part on TRIQS! So change to this directory and create a sub dir for the Wien2k related steps,

```
[training@n41-001 ~]$ cd triqs
[training@n41-001 triqs]$ mkdir NiO
[training@n41-001 triqs]$ cd NiO
[training@n41-001 NiO]$
```

Now, before we do the Wien2k calculations, you have to logout/login again, in case you executed *init_triqs.sh* as described above!

I assumed that your Wien2k calculation was called NiO. Now restore your saved Wien2k calculation into this directory, and let it run for one more DFT loop:

```
[training@n41-001 NiO]$ restore -d path_to_wien2k_save_dir -f
[training@n41-001 NiO]$ run -i 1
```

If you don't have a saved Wien2k calculation for NiO fcc, just do the normal steps for a Wien2k calculation to get a non-magnetic converged result. As parameters, please use 20000 k-points in the IBZ, and the PBE functional (flag 13 in Wien2k/lapw0). You can find the struct file (and all other Wien2k files that we will use in the following) in the Wien2kfiles subdirectory. This should be your total energy:

```
TOTAL ENERGY IN Ry =        -3192.09454270
```

3 Step 2: Construction of Wannier Functions

3.1 DOS and bands

First, we have a look on the density of states, projected to the Ni d orbitals. You can use the NiO.int file provided in the Wienk2files directory. Copy it to the directory where you did the DFT calculation and do

```
[training@n41-001 NiO]$ x lapw2 -qtl
[training@n41-001 NiO]$ x tetra
[training@n41-001 NiO]$ xmgrace -block NiO.dos1ev -bxy 1:3
```

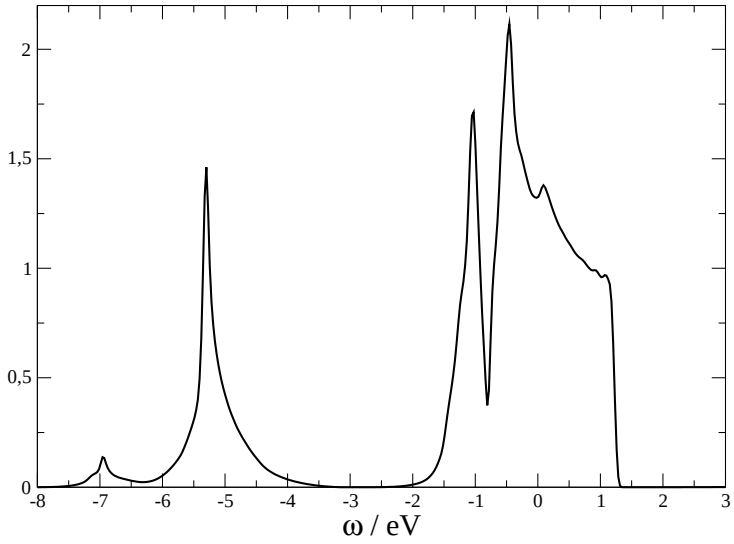

Figure 1: Density of states of the Ni-e$_g$ states.

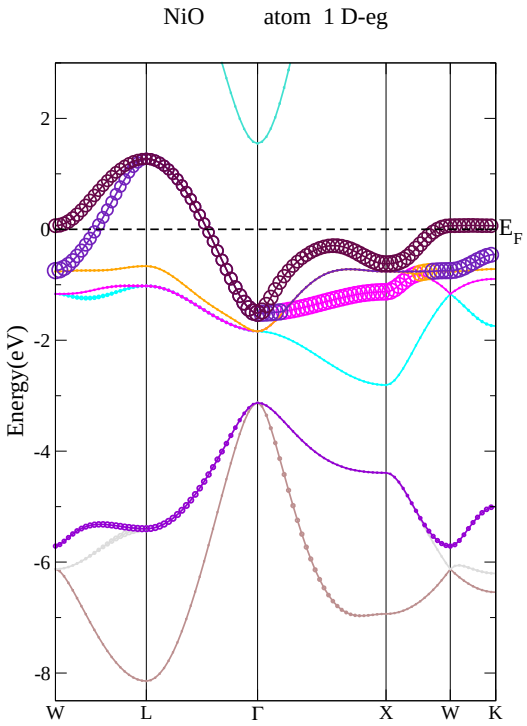

Figure 2: Band structure of NiO fcc, with Ni-e$_g$ fat band character.

This plots the e_g density of states, which you can see in Fig. 1.
This gives already a hint on the energy window that we want to use for the projective Wannier function procedure. However, let's have a look also on the band structure. Copy the *NiO.klist_band* file into your directory, and do

```
[training@n41-001 NiO]$ x lapw1 -band
[training@n41-001 NiO]$ x lapw2 -band -qtl
```

You now copy *NiO.insp*, check that the Fermi energy is set correctly, and produce the band structure with e_g fat bands. The result should be similar to Fig. 2.

3.2 Projective Wannier Functions

In order to calculate the projective Wannier function, we use the little program *dmftproj* that comes with the TRIQS/DFTTools package. To prepare the input for dmftproj, we have to run lapw2 with a special flag, and to one SC loop before (why?):

```
[training@n41-001 NiO]$ run -i 1 -NI
[training@n41-001 NiO]$ x lapw2 -almd
```

The program dmftproj now needs an input file, NiO.indmftpr (you can find it in the Wien2kfiles directory). It looks like

```
2                ! Nsort
1 1              ! Mult(Nsort)
3                ! lmax
cubic            ! choice of angular harmonics
0 0 2 0          ! l included for each sort
0 0 2 0          ! If split into ireps, gives number of ireps. for a given orbi
10               ! eg yes, t2g no
0                ! SO flag
cubic            ! choice of angular harmonics
0 1 0 0          ! l included for each sort
0 0 0 0          ! If split into ireps, gives number of ireps. for a given orbi
wmin wman       ! Ni eg bandwidth
```

The only thing you need to do is to define the energy window for the projection to the e_g states. You have to give the two numbers *wmin* and *wmax* in Rydberg, where the conversion 1 Ryd= 13.605 eV.
(You should get something close to -0.12 Ryd and 0.11 Ryd).
Now we have everything prepared for the calculation of the Wannier functions. Since this is part of the TRIQS package, we switch to TRIQS by

```
[training@n41-001 NiO]$ source ~/triqs/init_triqs.sh
```

And now, we can use our little program to calculate the e_g-like Wanniers:

```
[training@n41-001 NiO]$ dmftproj
```

3.3 Conversion to TRIQS format and Check

We will leave now the working directory of Wien2k, and continue in a different place. However, we have to copy the files that we need:

```
[training@n41-001 NiO]$ cd ..
[training@n41-001 triqs]$ cp NiO/NiO.ctqmcout .
[training@n41-001 triqs]$ cp NiO/NiO.symqmc .
```

Now we can convert the text data to the file format that is used by TRIQS (hdf5):

```
[training@n41-001 triqs]$ mpirun -np 1 pytriqs tut_dmft_ex1.py
```

Note that we need to run it under MPI, since we are doing this tutorial on a computing cluster, and forget about the 'fork' warning, in case you get it.
This produces a file, called *NiO.h5*, with all the DFT data inside that we need for the DMFT calculation. We can have a look inside:

```
[training@n41-001 triqs]$ pytriqs -i lookinside.py
>>> A['dft_input']['corr_shells']
[{'sort': 1, 'dim': 2, 'l': 2, 'irep': 1, 'SO': 0, 'atom': 1}]
>>>
```

Among many others, there is one entry in the archive called 'corr_shells' with the information that we provided by the input file for dmftproj. As you can see, the dimension of this e_g problem is 2, as it has to be. (Exit by CTRL-D).

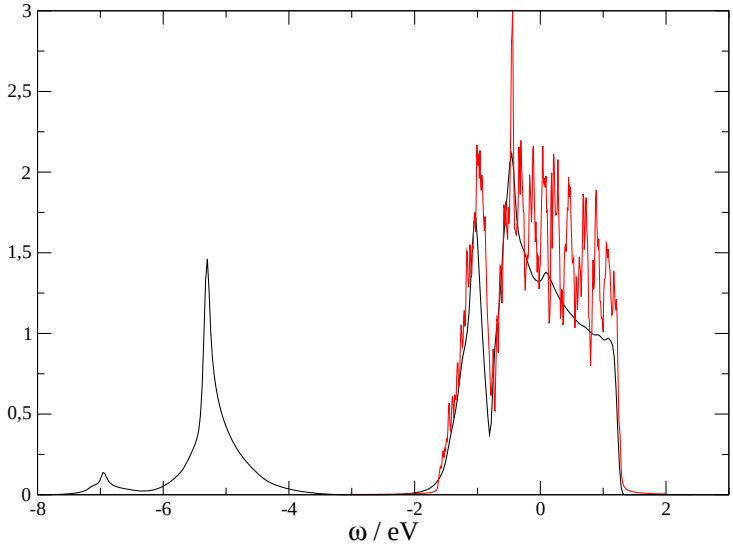

Figure 3: Comparison of Wien2k projected DOS (black) and the Wannier DOS (red) with Lorentian broadening 0.01.

From now on, we always need this file *NiO.h5* with the input data. For a start, we can calculate the density of states of our Wannier functions,

```
[training@n41-001 triqs]$ mpirun -np 1 pytriqs tut_dmft_ex2.py
```

You can plot and compare it to the Wien2k DOS using xmgrace:

```
xmgrace -block NiO/NiO.dos1ev -bxy 1:3 DOS_wann_up_proj0.dat
```

The result is shown in Fig. 3, where the Wannier DOS is multiplied by a factor of 2 in order to account for the spin. First of all, there is no Wannier weight below -2 eV, since this is below the projection window. Moreover, the result is quite spiky due to the simple point integration that TRIQS uses for the k summation. You can increase the broadening factor in tut_dmft_ex2.py to a larger value (c.f. 0.3), and compare again.

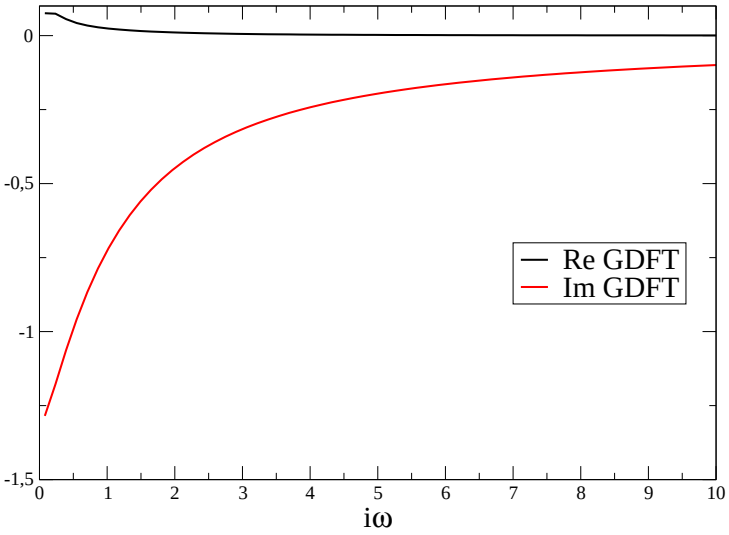

Figure 4: Real and imaginary part of the Matsubara Greens function G_{DFT}, the input from DFT.

4 The Impurity Solver

4.1 Local DFT Green function

We now start setting up the DMFT calculation. For that, we have to calculate to local non-interacting Green function from the DFT input, which serves as the first input to the impurity problem. So lets calculate it by

```
mpirun -np 1 pytriqs tut_dmft_ex3.py
```

This produces text files with the data of this Greens function, which is also plotted in Fig. 4. Note that a finite value of $\mathrm{Im}G_{DFT}(i\omega)$ for $\omega \to 0$ stands for metallic behavior. For an insulator, this value has to extrapolate to zero. As a next step, we can check whether the chemical potential is set correctly,

```
mpirun -np 1 pytriqs tut_dmft_ex4.py
```

We get density close to 2.12 for the e_g manifold, which is close to the DFT value.

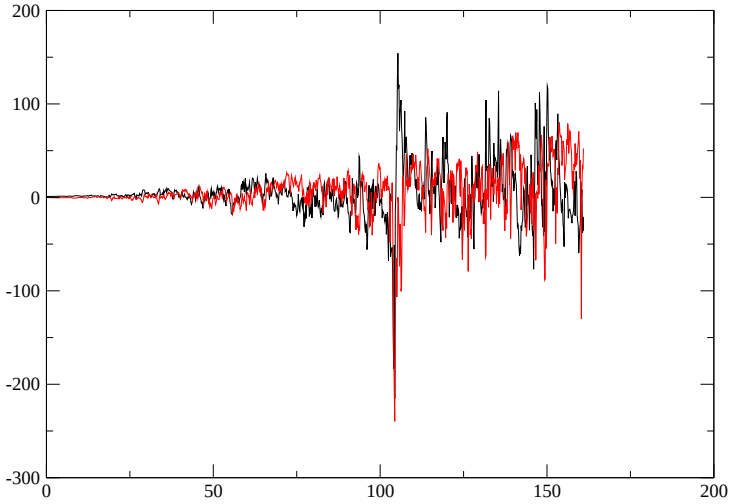

Figure 5: Real and imaginary part of the Matsubara self energy, without tail fit and Hartree constant.

4.2 Setting up the Impurity Problem and Solver

After having checked the main input, we continue with the interaction Hamiltonian. The script tut_dmft_ex5.py is already a bit longer, and will do this job for you. The main parameters to look at are:

```
# Setup the Hamiltonian
U = 4.0
J = 0.9
U4ind = U_matrix(l=2,U_int=U, J_hund=J, basis = 'cubic')
Umat_eg = eg_submatrix(U = U4ind)
H = h_int_slater(spin_names, orb_names, Umat_eg, off_diag = False)

# Parameters for the CTQMC Solver
p = {}
p["max_time"] = -1
p["random_name"] = ""
p["random_seed"] = 123 * mpi.rank + 567
p["length_cycle"] = 200
p["n_warmup_cycles"] = 1000
p["n_cycles"] = 20000
```

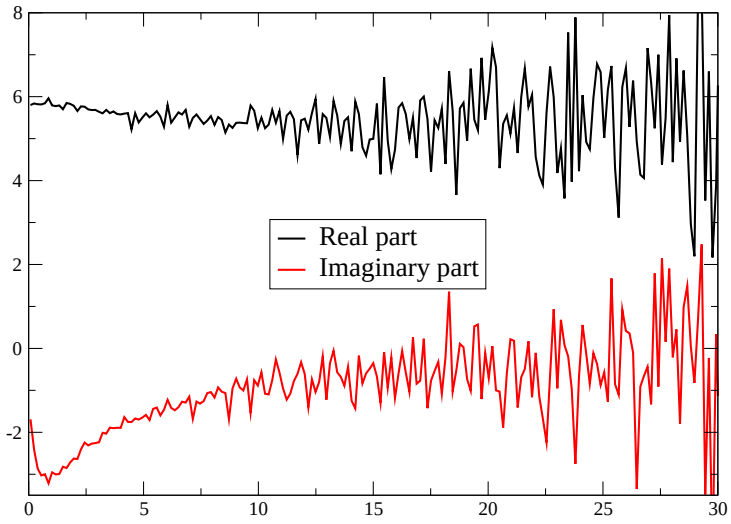

Figure 6: Real and imaginary part of the Matsubara self energy, without tail fit but with Hartree constant. Low energy looks not so bad.

Some of them are self-explanatory. The most important one for the Monte Carlo solver is the number of MC cycles in the last line. We start with a low number, and just let it run:

```
mpirun -np 1 pytriqs tut_dmft_ex5.py
```

First of all, the charge of the impurity problem is very low, around 0.6 as compared to 2.12. In addition, when looking at one of the self energies, e.g.

```
xmgrace -nxy Sigma_ex5_up_0.dat
```

this does not look nice. Let's first take a look at the charge. The problem is that we do not start with a good guess for the self energy, which was just setting it to zero. Instead we initialize it now with the constant Hartree energy in the next example script,

```
mpirun -np 1 pytriqs tut_dmft_ex6.py
```

As guess for the Hartree energy, we use the value for double counting correction. This concept is the same as in DFT+U, where one has the correct for the interaction energy that is already included in DFT. We use here the full-localized-limit (FLL) variant (flag use_dc_value=0).

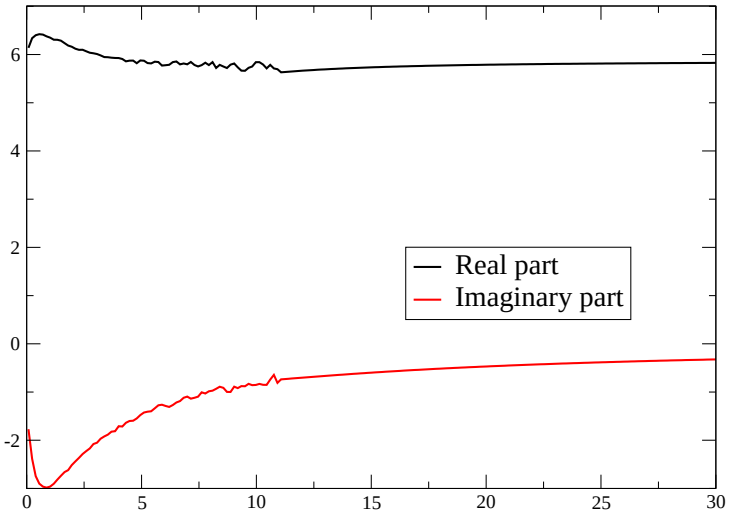

Figure 7: Real and imaginary part of the Matsubara self energy, with tail fit but with Hartree constant. 16 MPI processes have been used for this plot.

Now, the density is much better (around 2.02). We also increased the number of MC cycles a bit to 100000 to get better data. Nevertheless, the problem at high energies persists, and we have to take care of what is called the tail of the self energy. As can be seen in Fig. 6, the low energy part is okay, and we need to replace the noise at high energy with some analytic form. This we do by tail fitting, which is implemented in tut_dmft_ex7.py. The parameters are

```
#This is for the tail:
p["fit_max_moment"] = 3
p["fit_min_n"] = 70
p["fit_max_n"] = 170
p["perform_tail_fit"] = True
```

We fit three moments of the high-frequency expansion, and use data between Matsubara frequency index 70 and 170 for the fit. Note that the Matsubara frequencies are defined as $\omega_n = \pi/\beta(2n+1)$, where n is the Matsubara index. Now the self energy looks much better, with good analytic behavior towards high energies. However, we still have problems with good statistics, since we do not use enough sampling in our problem. This can be cured either by increasing the number of MC cycles further, or by using more MPI threads

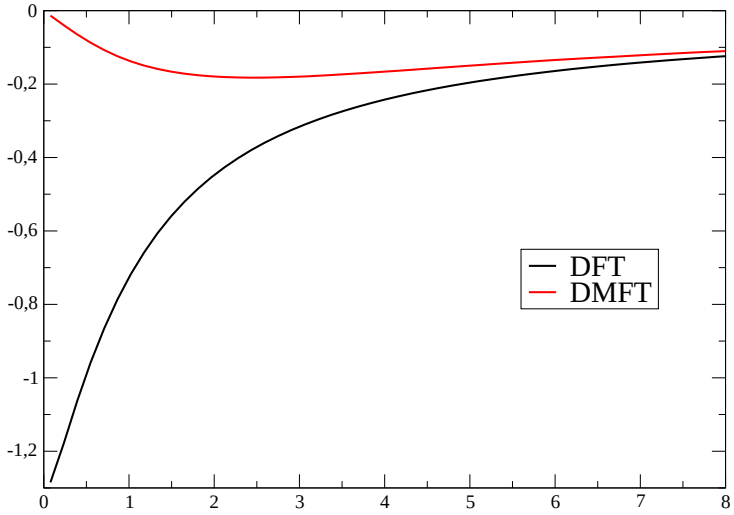

Figure 8: Imaginary parts of the local DFT Green function (black), and of the converged DMFT impurity Green function (black). Whereas the first one is clearly metallic, the latter one is a clear insulator, even without magnetic symmetry breaking.

on the computing cluster. You can increase the latter number simply by

```
mpirun -np 16 pytriqs tut_dmft_ex7.py
```

which let's the code run with 16 instances. This just increases the number of MC data by this factor.

5 DMFT

Now we have set up our Solver correctly for the DMFT calculation. We have a ready made script that let's the self consistent cycle with this setup run for 6 iterations,

```
mpirun -np 6 pytriqs tut_dmft_ex8.py > outDMFT 2>&1 &
```

We let it run in the background, since it takes some time, but pipe all output into a file *outDMFT*. You can check the progress of the calculation by several means, for instance

```
grep 'Total charge' outDMFT
```

tells you the current values of the impurity and local lattice density. These numbers have to converge, of course. The script also prints all the self energies into files, which you can check for convergence.

As a final result, you should get something similar to Fig. 8. The input from DFT was clearly metallic, but for our values of Coulomb interaction $U = 4.0$ and Hunds coupling $J = 0.9$ we get as a result an insulator, even in the paramagnetic case. This is qualitatively different to any DFT calculation, and is the paradigmatic example of a Mott insulator.

5.1 Further tasks

1. As a further problem you can look at smaller interaction values. Changing the values to $U = 1.0$ and $J = 0.2$ is not difficult in the scripts, but be aware that also the tail fit is sensitive to the interaction values and has to be done carefully again! At the end, you might up with different results as for the $U = 4.0$, $J = 0.9$ case.

2. You could try out whether the ferromagnetic state is stable in DMFT. For this, you have to initialize the Self energy for the first run with different values for up and down spins. Then you do all procedures as above and check, whether the iteration is stable and converges to a finite magnetic moment. This is already a quite advanced task.