Lecture 4
Introduction to hands-on
TRIQS : A Toolbox for Research in Interacting Quantum Systems

Olivier Parcollet

Institut de Physique Théorique

CEA, Université Paris-Saclay,

France

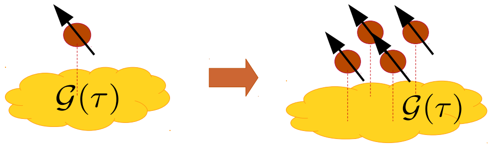
DE LA RECHERCHE À L'INDUSTRIE
cea

erc
European Research Council
Established by the European Commission

SIMONS FOUNDATION
Advancing Research in Basic Science and Mathematics

DMFT is quite versatile

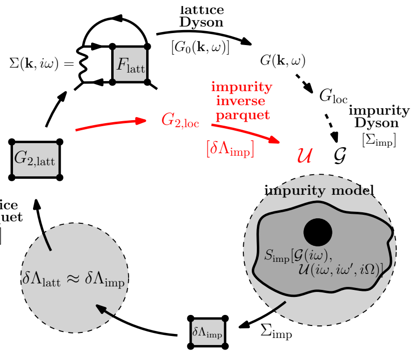
- Cluster DMFT



Control, short range correlation

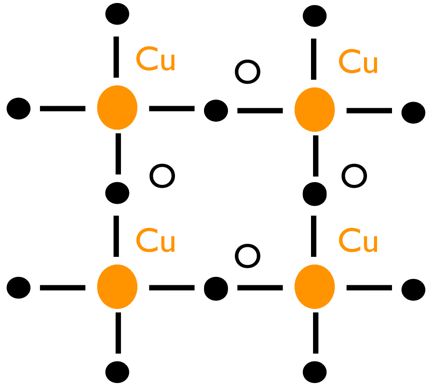
- Beyond cluster DMFT

Self-consistency on vertex
Dual fermions/bosons, Trilex, DΓA


lattice
Dyson
$[G_0(\mathbf{k},\omega)]$
$G(\mathbf{k},\omega)$
$\Sigma(\mathbf{k},i\omega) =$
F_{latt}
G_{loc}
impurity
Dyson
$[\Sigma_{\text{imp}}]$
impurity
inverse
parquet
$G_{2,\text{loc}}$
$[\delta\Lambda_{\text{imp}}]$
$\mathcal{U}$
$\mathcal{G}$
$G_{2,\text{latt}}$
impurity model
$S_{\text{imp}}[\mathcal{G}(i\omega), \mathcal{U}(i\omega, i\omega', i\Omega)]$
lattice
parquet
$[U]$
$\delta\Lambda_{\text{latt}} \approx \delta\Lambda_{\text{imp}}$
$\delta\Lambda_{\text{imp}}$
Σ_{imp}

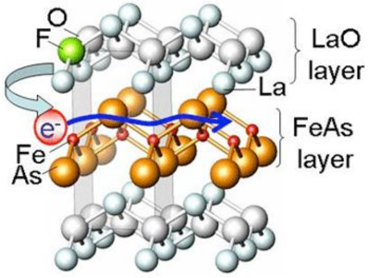
- Multiband/realistic systems

$$\Sigma(\omega) = \begin{pmatrix} \Sigma^{\text{imp}}(\omega) & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$



  - Self-consistency in large unit cell (Cu + 2 O)
    $\Sigma_{ab}(\omega)$ a 3x3 matrix
  - Impurity model on Cu, 1 band : $\Sigma^{\text{imp}}(\omega)$ 1x1 matrix

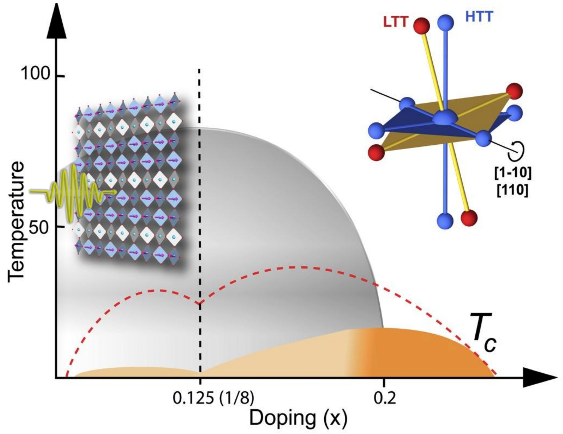
- DFT + DMFT



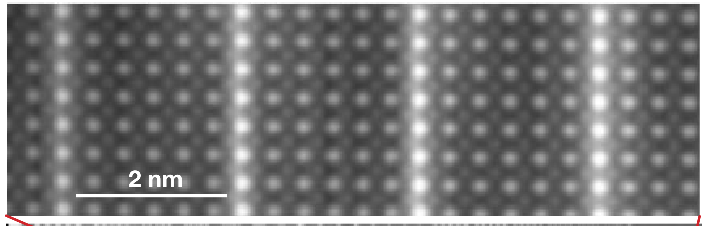
  - Interface with electronic structure codes (project on Wannier functions, etc).

DMFT is quite versatile

- Non equilibrium



- Correlated interfaces.



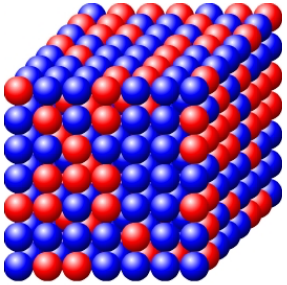
$SrTiO_3/LaTiO_3$

Ohtomo et al, Nature 2002

- One impurity per layer

- Disordered systems

Alloy



- Two impurity models

Need for a library

- No "general" DMFT code.
- Better to have a simple language to express your calculation.

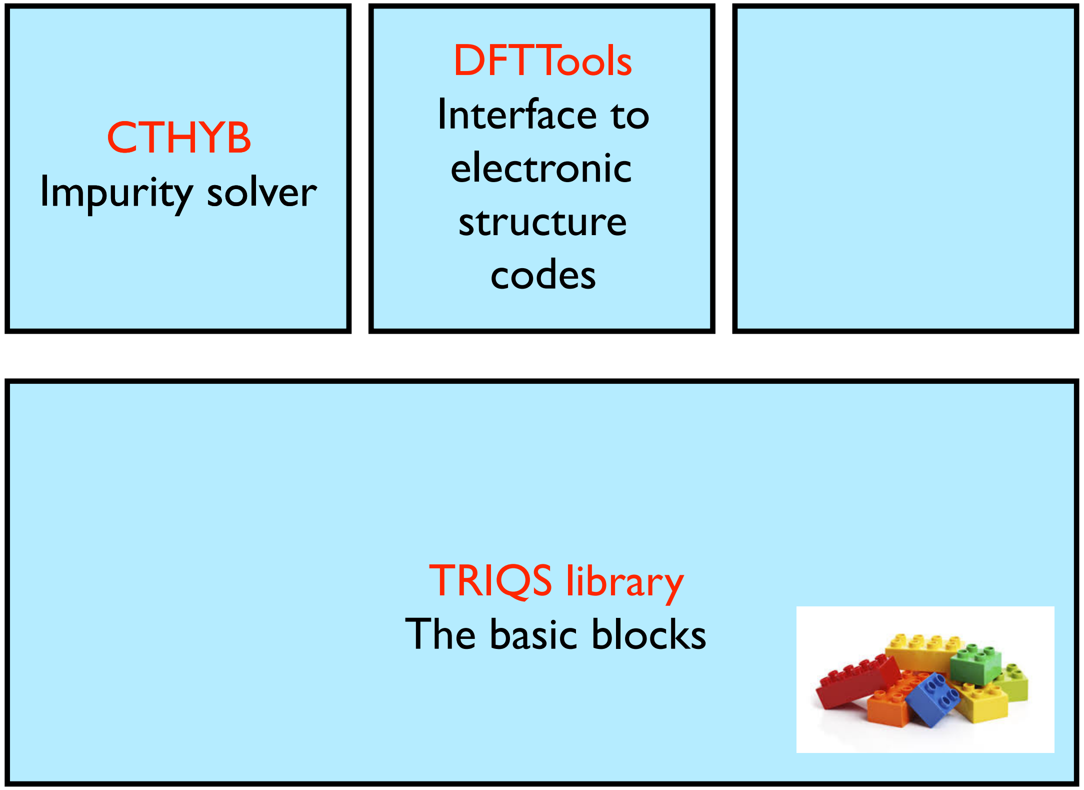
- Design goals
  - Basic blocks for DMFT and beyond, diagrammatic methods
  - Simplicity : what is simple should be coded simply !
  - High performance :
    - Human time : reduce the cost of writing codes.
    - Machine time : run quickly.

TRIQS structure

- A Library in Python & C++
- Applications
  - State of the art “impurity solvers” for DMFT.
  - Interface with electronic structure codes.

https://triqs.ipht.cnrs.fr

https://github.com/TRIQS



CTHYB
Impurity solver

DFTTools
Interface to electronic structure codes

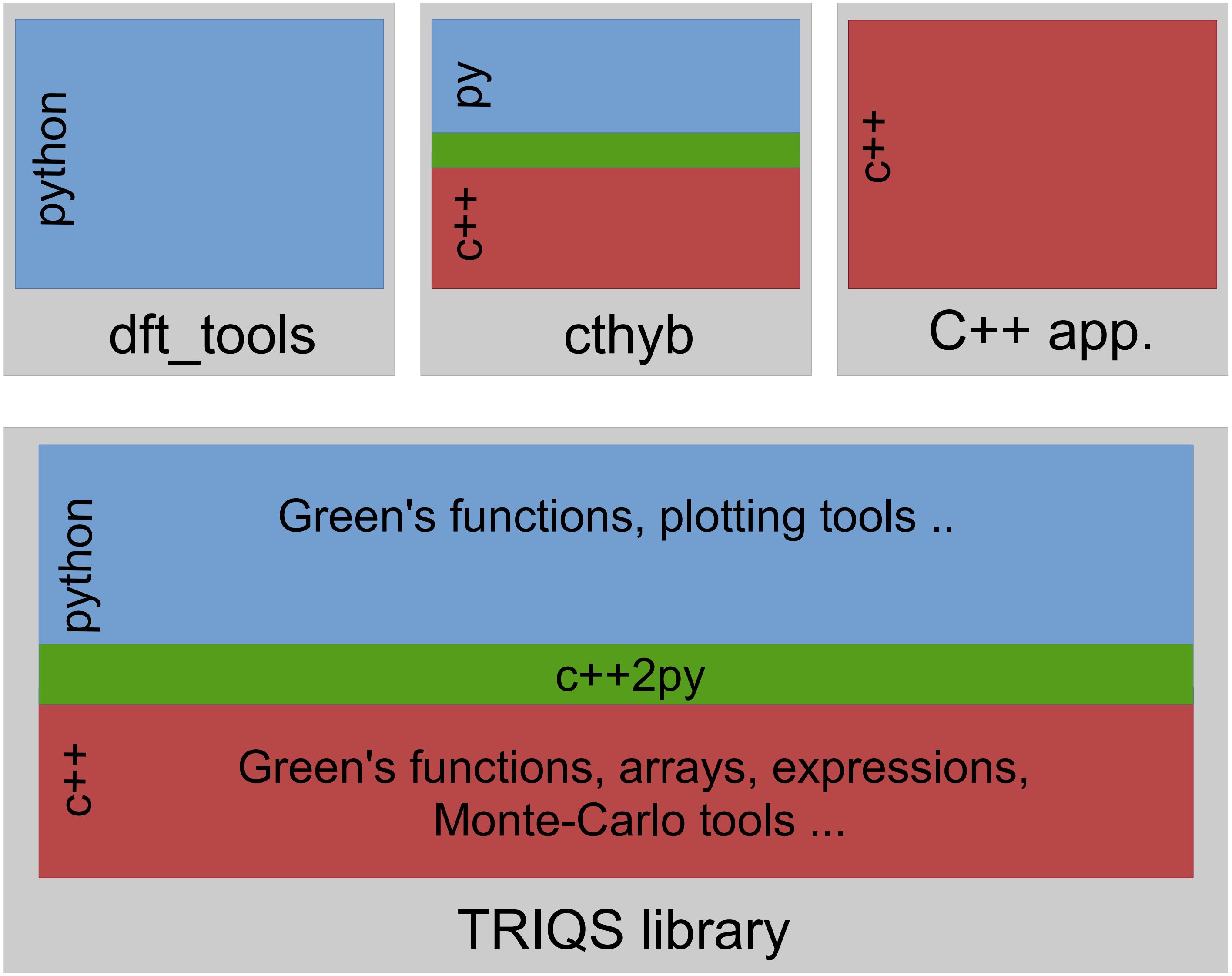
TRIQS library
The basic blocks

TRIQS library: contents

TRIQS library
The basic blocks

- Green functions containers
  G(ω), G(k,ω), Vertex Γ(ω,ν,ν').
- Generic Monte Carlo class & error analysis tools.
- Determinant manipulations (for QMCs).
- Lattice tools: Bravais Lattices, Brillouin zone, ...
- Many-body operators.
- More general tools, e.g.
  - Multidimensional array class
  - HDF5 light interface in Python & C++
  - Python/C++ light interfacing tool

Python/C++ toolkit

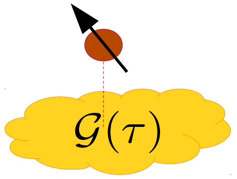

python
dft_tools
py
c++
cthyb
c++
C++ app.
python
Green's functions, plotting tools ..
c++2py
c++
Green's functions, arrays, expressions, Monte-Carlo tools ...
TRIQS library

A full DMFT computation in 1 slide

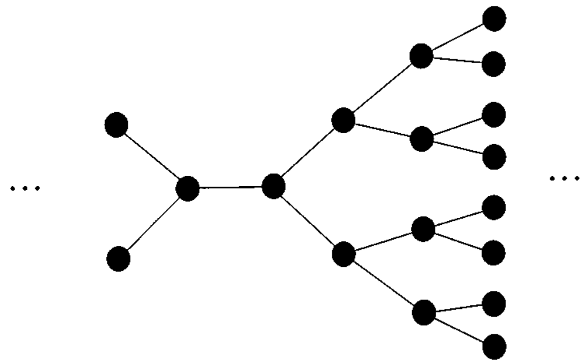
DMFT on Bethe lattice

- DMFT equations : 1 band, Hubbard model, Bethe lattice

$$S_{\mathrm{eff}} = -\iint_0^\beta d\tau d\tau' \, c_\sigma^\dagger(\tau)\mathcal{G}_\sigma^{-1}(\tau-\tau')c_\sigma(\tau') + \int_0^\beta d\tau \; U n_\uparrow(\tau) n_\downarrow(\tau)$$



$$G_{\sigma\mathrm{imp}}(\tau) \equiv -\left\langle T c_\sigma(\tau) c_\sigma^\dagger(0)\right\rangle_{S_{\mathrm{eff}}}$$



$$\boxed{\mathcal{G}_\sigma^{-1}(i\omega_n) = i\omega_n + \mu - \underbrace{t^2 G_{\sigma\mathrm{imp}}(i\omega_n)}_{\Delta_\sigma(i\omega_n)}}$$

- Goal: Solve DMFT equations, self-consistently with an impurity solver packaged in TRIQS (a quantum Monte Carlo)

How to do it ?

- Break the DMFT computation into small parts and assemble the computation.

- Which parts ?
  - Local Green functions
  - An impurity solver: e.g. the CT-INT solver.
  - Save the result.
  - Plot it.

Assemble a DMFT computation in 1 slide

- A complete code, using a QMC impurity solver (a TRIQS app).
- In Python, with parallelization included (mpi).

- Do not worry about the details of the syntax at this stage
  Get an idea of how to use TRIQS by example.

DMFT computation in 1 slide

```
from pytriqs.gf.local import *
from pytriqs.applications.impurity_solvers.ctint_tutorial import CtintSolver


U = 2.5                # Hubbard interaction
mu = U/2.0             # Chemical potential
half_bandwidth=1.0 # Half bandwidth (energy unit)
beta = 40.0            # Inverse temperature
n_iw = 128             # Number of Matsubara frequencies
n_cycles = 10000       # Number of MC cycles
delta = 0.1            # delta parameter
n_iterations = 21      # Number of DMFT iterations

S = CtintSolver(beta, n_iw) # Initialize the solver

S.G_iw << SemiCircular(half_bandwidth) # Initialize the Green's function
```

- Import some basic blocks (Green function, a solver).
- Define some parameters and declare a CT-INT solver S
- All TRIQS solvers contains G, G_0, Σ as members
  with the correct β, dimensions, etc.
- Initialize S.G_iw to a (the Hilbert transform of a) semi-circular dos.

DMFT computation in 1 slide

```
from pytriqs.gf.local import *
from pytriqs.applications.impurity_solvers.ctint_tutorial import CtintSolver


U = 2.5             # Hubbard interaction
mu = U/2.0          # Chemical potential
half_bandwidth=1.0 # Half bandwidth (energy unit)
beta = 40.0         # Inverse temperature
n_iw = 128          # Number of Matsubara frequencies
n_cycles = 10000    # Number of MC cycles
delta = 0.1         # delta parameter
n_iterations = 21   # Number of DMFT iterations

S = CtintSolver(beta, n_iw) # Initialize the solver

S.G_iw << SemiCircular(half_bandwidth) # Initialize the Green's function

for sigma, G0 in S.G0_iw: # sigma = ‘up’, ‘down’
  G0 << inverse(iOmega_n + mu - (half_bandwidth/2.0)**2 * S.G_iw[sigma] ) # Set G0
```

$$G_{0\sigma}^{-1}(i\omega_n) = i\omega_n + \mu - t^2 G_{c\sigma}(i\omega_n), \quad \text{for } \sigma = \uparrow, \downarrow$$

- Implement DMFT self-consistency condition

DMFT computation in 1 slide

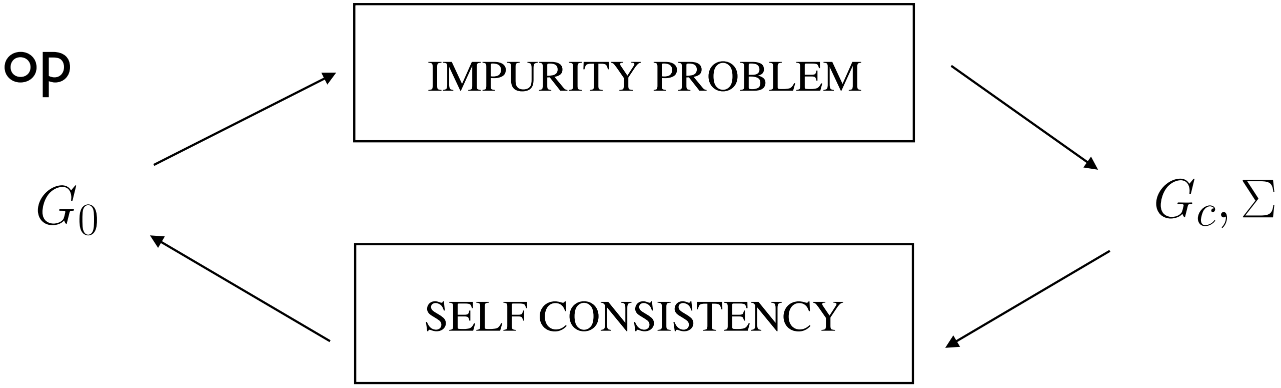
```
from pytriqs.gf.local import *
from pytriqs.applications.impurity_solvers.ctint_tutorial import CtintSolver


U = 2.5                # Hubbard interaction
mu = U/2.0             # Chemical potential
half_bandwidth=1.0 # Half bandwidth (energy unit)
beta = 40.0            # Inverse temperature
n_iw = 128             # Number of Matsubara frequencies
n_cycles = 10000       # Number of MC cycles
delta = 0.1            # delta parameter
n_iterations = 21      # Number of DMFT iterations

S = CtintSolver(beta, n_iw) # Initialize the solver

S.G_iw << SemiCircular(half_bandwidth) # Initialize the Green's function

for sigma, G0 in S.G0_iw:
  G0 << inverse(iOmega_n + mu - (half_bandwidth/2.0)**2 * S.G_iw[sigma] ) # Set G0

S.solve(U, delta, n_cycles) # Solve the impurity problem
```

- Call the solver.
- From $G_{0\sigma}(i\omega_n)$ (and various parameters),
  it computes $G_\sigma(i\omega_n)$ for $\sigma=\uparrow,\downarrow$

DMFT computation in 1 slide

```
from pytriqs.gf.local import *
from pytriqs.applications.impurity_solvers.ctint_tutorial import CtintSolver


U = 2.5            # Hubbard interaction
mu = U/2.0         # Chemical potential
half_bandwidth=1.0 # Half bandwidth (energy unit)
beta = 40.0        # Inverse temperature
n_iw = 128         # Number of Matsubara frequencies
n_cycles = 10000   # Number of MC cycles
delta = 0.1        # delta parameter
n_iterations = 21  # Number of DMFT iterations

S = CtintSolver(beta, n_iw) # Initialize the solver

S.G_iw << SemiCircular(half_bandwidth) # Initialize the Green's function

for it in range(n_iterations): # DMFT loop
  for sigma, G0 in S.G0_iw:
    G0 << inverse(iOmega_n + mu - (half_bandwidth/2.0)**2 * S.G_iw[sigma] ) # Set G0

  S.solve(U, delta, n_cycles) # Solve the impurity problem
```

- DMFT iteration loop



IMPURITY PROBLEM

SELF CONSISTENCY

G_0

G_c, Σ

DMFT computation in 1 slide

```
from pytriqs.gf.local import *
from pytriqs.applications.impurity_solvers.ctint_tutorial import CtintSolver



U = 2.5            # Hubbard interaction
mu = U/2.0         # Chemical potential
half_bandwidth=1.0 # Half bandwidth (energy unit)
beta = 40.0        # Inverse temperature
n_iw = 128         # Number of Matsubara frequencies
n_cycles = 10000   # Number of MC cycles
delta = 0.1        # delta parameter
n_iterations = 21  # Number of DMFT iterations

S = CtintSolver(beta, n_iw) # Initialize the solver

S.G_iw << SemiCircular(half_bandwidth) # Initialize the Green's function

for it in range(n_iterations): # DMFT loop
  for sigma, G0 in S.G0_iw:
    G0 << inverse(iOmega_n + mu - (half_bandwidth/2.0)**2 * S.G_iw[sigma] ) # Set G0

  S.solve(U, delta, n_cycles) # Solve the impurity problem

  G_sym = (S.G_iw['up'] + S.G_iw['down'])/2 # Impose paramagnetic solution
  S.G_iw << G_sym
```

- Enforce the fact that the solution is paramagnetic.
  (noise in the QMC would lead to a AF solution after iterations).

DMFT computation in 1 slide

```
from pytriqs.gf.local import *
from pytriqs.applications.impurity_solvers.ctint_tutorial import CtintSolver
from pytriqs.archive import HDFArchive


U = 2.5               # Hubbard interaction
mu = U/2.0            # Chemical potential
half_bandwidth=1.0 # Half bandwidth (energy unit)
beta = 40.0           # Inverse temperature
n_iw = 128            # Number of Matsubara frequencies
n_cycles = 10000      # Number of MC cycles
delta = 0.1           # delta parameter
n_iterations = 21     # Number of DMFT iterations

S = CtintSolver(beta, n_iw) # Initialize the solver

S.G_iw << SemiCircular(half_bandwidth) # Initialize the Green's function

for it in range(n_iterations): # DMFT loop
  for sigma, G0 in S.G0_iw:
    G0 << inverse(iOmega_n + mu - (half_bandwidth/2.0)**2 * S.G_iw[sigma] ) # Set G0

  S.solve(U, delta, n_cycles) # Solve the impurity problem

  G_sym = (S.G_iw['up'] + S.G_iw['down'])/2 # Impose paramagnetic solution
  S.G_iw << G_sym

  with HDFArchive("dmft_bethe.h5",'a') as A:
    A['G%i'%it] = G_sym # Save G from every iteration to file as G1, G2, G3....
```

- Accumulate the various iterations in a (hdf5) file

DMFT computation in 1 slide

```
from pytriqs.gf.local import *
from pytriqs.applications.impurity_solvers.ctint_tutorial import CtintSolver
from pytriqs.archive import HDFArchive


U = 2.5              # Hubbard interaction
mu = U/2.0           # Chemical potential
half_bandwidth=1.0 # Half bandwidth (energy unit)
beta = 40.0          # Inverse temperature
n_iw = 128           # Number of Matsubara frequencies
n_cycles = 10000     # Number of MC cycles
delta = 0.1          # delta parameter
n_iterations = 21    # Number of DMFT iterations

S = CtintSolver(beta, n_iw) # Initialize the solver

S.G_iw << SemiCircular(half_bandwidth) # Initialize the Green's function

for it in range(n_iterations): # DMFT loop
  for sigma, G0 in S.G0_iw:
    G0 << inverse(iOmega_n + mu - (half_bandwidth/2.0)**2 * S.G_iw[sigma] ) # Set G0

  # Change random number generator on final iteration
  random_name = 'mt19937' if it < n_iterations-1 else 'lagged_fibonacci19937'

  S.solve(U, delta, n_cycles, random_name=random_name) # Solve the impurity problem

  G_sym = (S.G_iw['up']+S.G_iw['down'])/2 # Impose paramagnetic solution
  S.G_iw << G_sym

  with HDFArchive("dmft_bethe.h5",'a') as A:
     A['G%i'%it] = G_sym # Save G from every iteration to file
```

- Change the random generator at the last iteration !

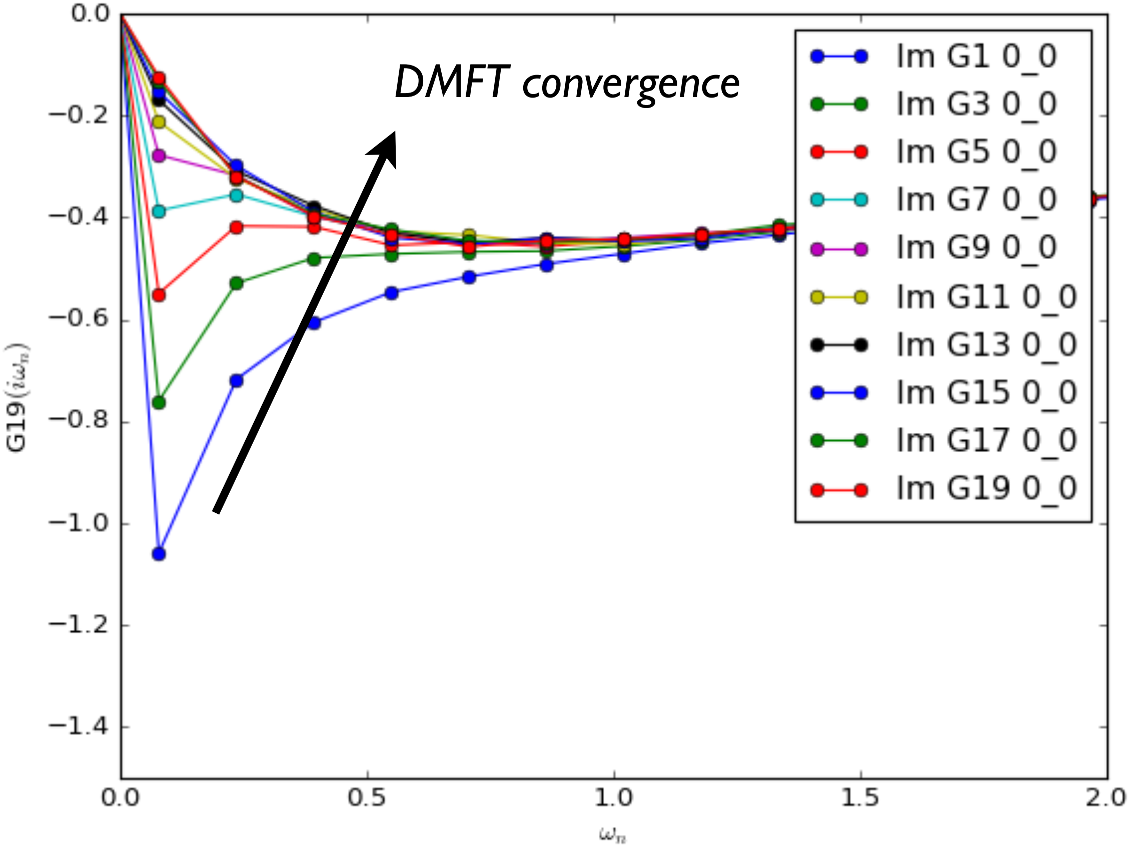
Look at the result (in IPython notebook)

```
A = HDFArchive("dmft_bethe.h5",'r') # Open file in read mode
for it in range(21):
    if it%2: # Plot every second result
    oplot(A['G%i'%it], '-o', mode='I', name='G%i'%it)
```

oplot can plot many TRIQS objects via matplotlib

Retrieve Gi from the file, and use it at once

Imaginary part only



DMFT convergence

NB
lines are guide to the eyes, only Matsubara frequency point matters

HDF5 file format

- De facto standard file format.
- Language agnostic (python, C/C++, F90).
- Binary format hence compact, but also portable.
- Dump & reload objects in one line.
  Forget worrying about format, reading files, conventions.
- $G(\omega)(n_1,n_2)$ a 3d array of complex numbers, i.e. 4d array of reals.
  No natural convention in a 2d text file.

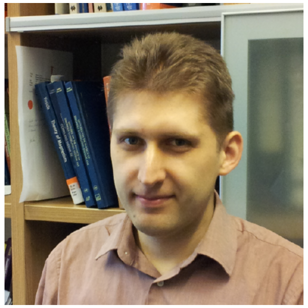
Collaborators/contributors

Michel Ferrero

Markus Aichhorn

Nils Wentzell

Hugo Strand



Igor Krivenko

Priyanka Seth

Thomas Ayral

Oleg Peil

Gernot Kraberger

Manuel Zingl

Leonid Pourovskii

Jernej Mravlje

Veronica Vildosola

Thank you for your attention

Hands-on

- Assemble a DMFT computation yourself.
- First, I will do an example : DMFT, bethe lattice, CT-INT.
- Second, you will do a few DMFT computations:
  - IPT solution of DMFT
  - Use CTHYB solver for 1 band, DMFT.
  - 2 bands Kanamori model with CTHYB.
    Effect of J on U_c
  - 2 patch DCA computation with CTHYB.
    Selective Mott transition in k space.

CT-INT demo code

https://github.com/TRIQS/ctint_tutorial.git

- In addition, the CT-INT code used earlier is available as a demo code.
- < 200 lines of C++.
  With Python interface, MPI, …
- Cf intro in TRIQS paper, arXiv:1504.01952, Appendix A.