TRIQS

A Toolbox for Research in Interacting Quantum Systems

Olivier Parcollet

Institut de Physique Théorique

CEA Saclay,

France

European Research Council

Established by the European Commission

SIMONS FOUNDATION

Advancing Research in Basic Science and Mathematics

Team for the hands-on

O. Parcollet

Michel Ferrero

Priyanka Seth

Gernot Kraberger

Manuel Zingl

What is TRIQS ?

- A Toolbox in Python & C++ to build modern many body computations:
  - DMFT methods [Today's topic]
    - Ab-initio strongly correlated materials (DFT+ DMFT).
    - Cluster DMFT methods.
    - Third generation methods: self-consistency with two particles Green functions.
  - TRIQS is not limited to DMFT methods.
    - Other many-body computations.
      e.g. Eliashberg type equations
      (superconductivity, spin-fluctuations...)
    - Some tools are very general, beyond condensed matter physics.

Outline

- Start with an example :
  1 band DMFT with CT-INT QMC as an impurity solver.
- Overview of the TRIQS project
- A few highlights of the contents of the library in Python & C++
- Hands-on:
  - You will build a little IPT solver at half-filling with TRIQS and run it to recover a classical result for Mott transition in DMFT.
  - Then compare with TRIQS exact and general CT-HYB quantum impurity solver
  - Finish with a 2 orbital model, study the role of U vs J_{hund}

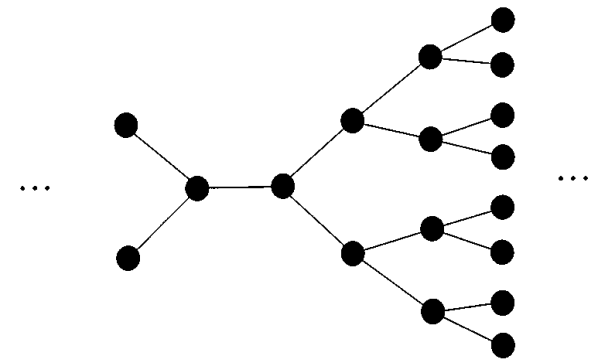
Motivating example

Let's do a DMFT computation in Python using TRIQS

Reminder : DMFT on Bethe lattice

- DMFT equations (Cf D. Sénéchal's lecture yesterday).

$$H = -\sum_{ij\sigma} t_{ij} c^{\dagger}_{i\sigma} c_{j\sigma} + U n_{i\uparrow} n_{i\downarrow}$$



$$G_c(\tau) = -\langle T c(\tau) c^{\dagger}(0) \rangle_{S_{eff}}$$

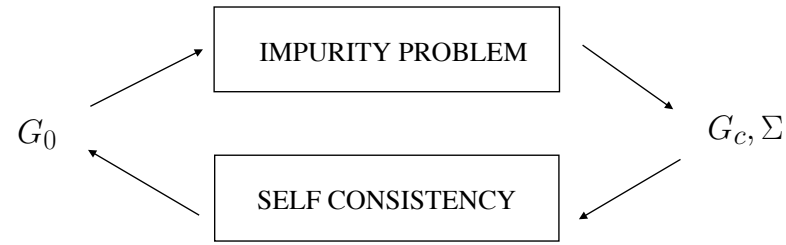
$$S_{\rm eff} = -\int_0^\beta c^{\dagger}_\sigma(\tau) G_0^{-1}(\tau - \tau') c_\sigma(\tau') + \int_0^\beta d\tau U n_\uparrow(\tau) n_\downarrow(\tau)$$

- Bethe lattice with infinite connectivity. Semi-circular d.o.s.

A. Georges, G. Kotliar, W. Krauth and M. Rozenberg, Rev. Mod. Phys. 68, 13, (1996)

$$\Delta(i\omega_n) = t^2 G_c(i\omega_n)$$

$$G_0^{-1}(i\omega_n) = i\omega_n + \mu - \Delta(i\omega_n)$$



- Goal: Solve DMFT equations, self-consistently with CT-INT.

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 CT-INT solver (one of the TRIQS apps).
- In Python, with parallelization included (mpi).

- Do not worry about the details of the syntax at this stage (Cf the hands-on).
  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

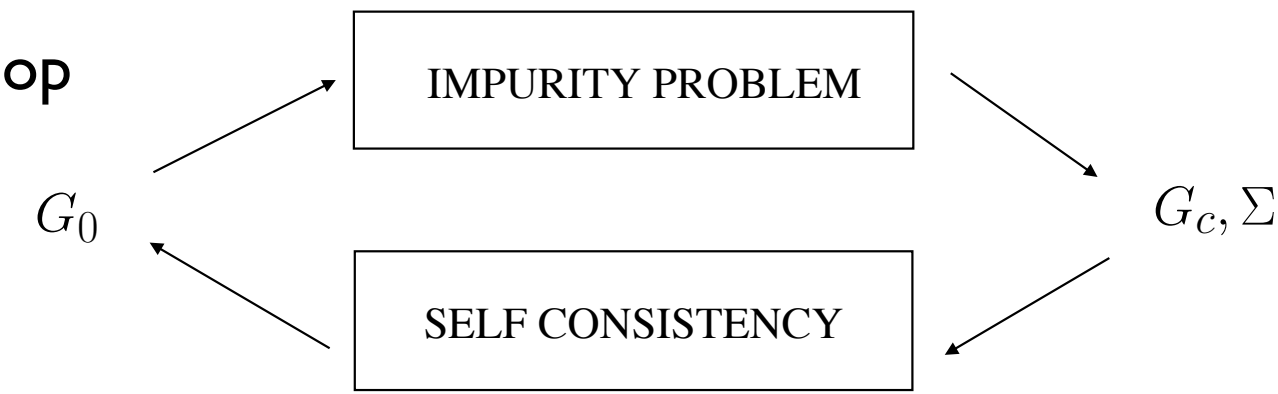
```
from pytriqs.gf.local import *
from pytriqs.applications.impurity_solvers.ctint_tutorial import CtintSolver


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



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, cf DMFT lecture. (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)
```

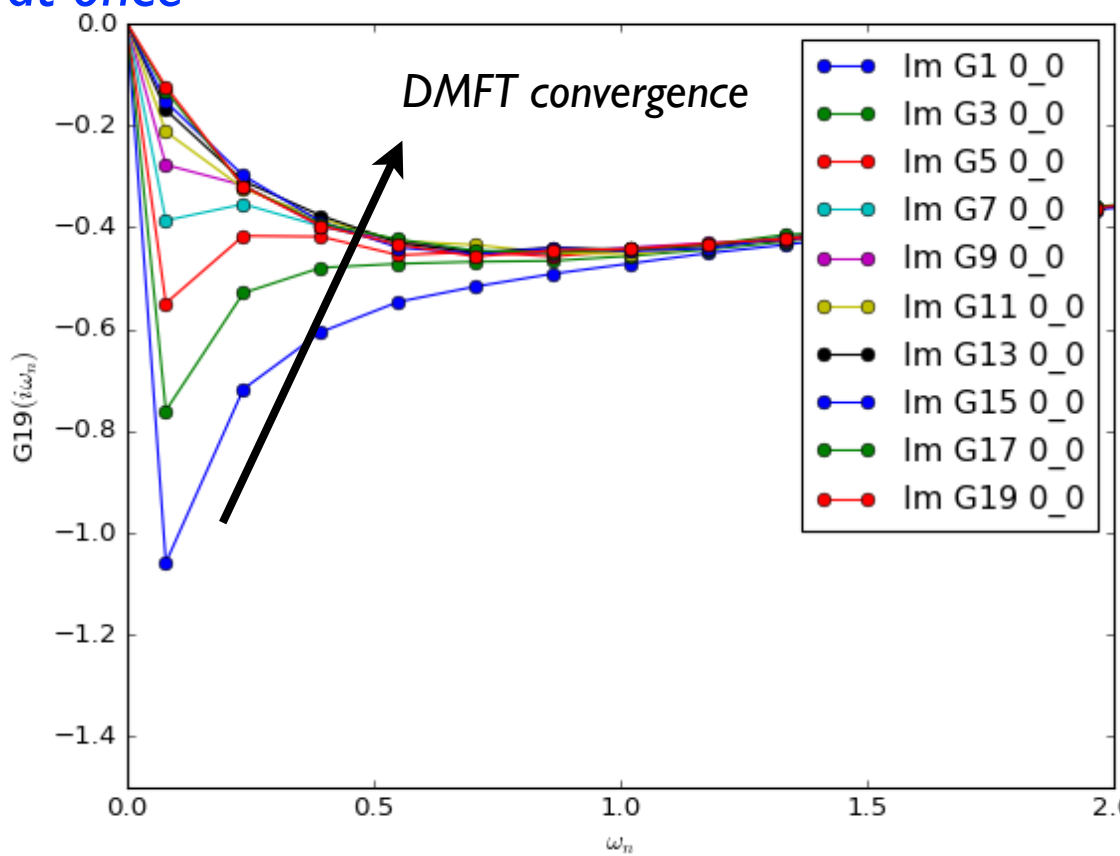
oplot can plot many TRIQS objects via matplotlib

Retrieve Gi from the file, and use it at once

Imaginary part only

NB

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



Summarize what we have done so far

- A fully functional DMFT code.
- Computation & data analysis: all in Python.
- Green functions, solvers are used as Python classes.
- Since it is a script, it is easy to change various details, e.g.
  - Self-consistency condition: enforce paramagnetism
  - Change random generator
  - Change starting point (e.g. reload G from a file).
  - Improve convergence (e.g. mixing quantities over iterations).
  - Measure e.g. susceptibilities only at the end of the DMFT loop

FAQ : Where was the input file ?

- Traditional way: a monolithic program, input files, output files.
- Python/TRIQS way : write your own script with full control
- No input file as text : no need to parse it,
  we can do operations on the fly, use Python to prepare data ...

```
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
```

• • •

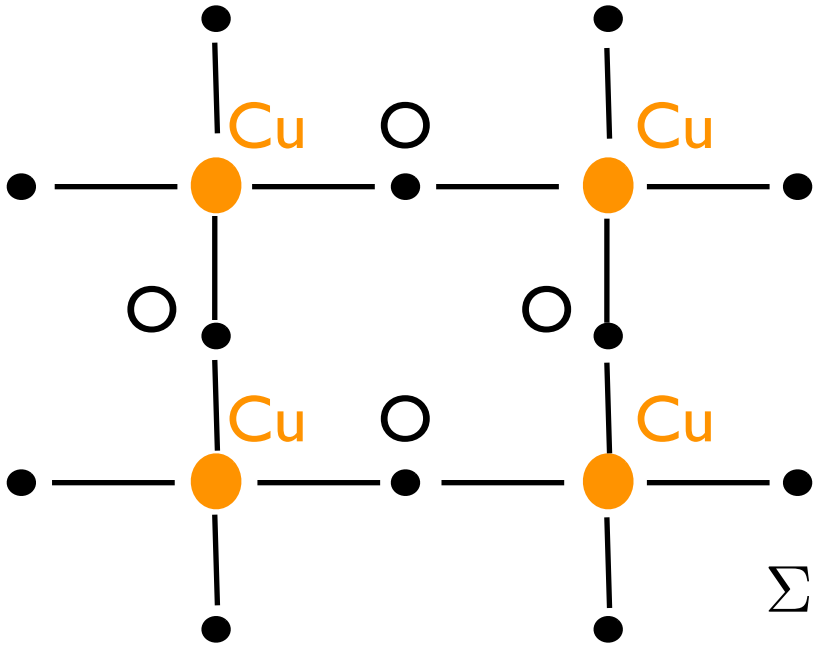
Why a library rather than a monolithic program ?

Library vs monolithic program

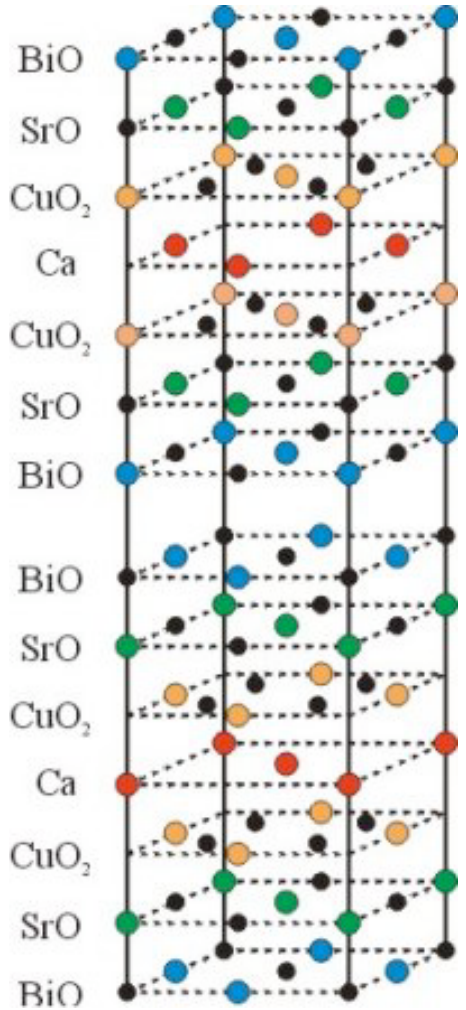
- Better to have a simple language to express your calculation.
- Since many-body approaches are quite versatile.
- Let me illustrate this point with examples of DMFT-like methods
  - Many impurity solvers (e.g. CT-QMC, ED, IPT, DMRG, NCA, ...)
  - Various impurity models (# of orbitals, symmetries, clusters)
  - Many self-consistency conditions ...

DMFT: correlated vs uncorrelated orbitals

- Select some correlated orbitals, even in models for cuprates, e.g.



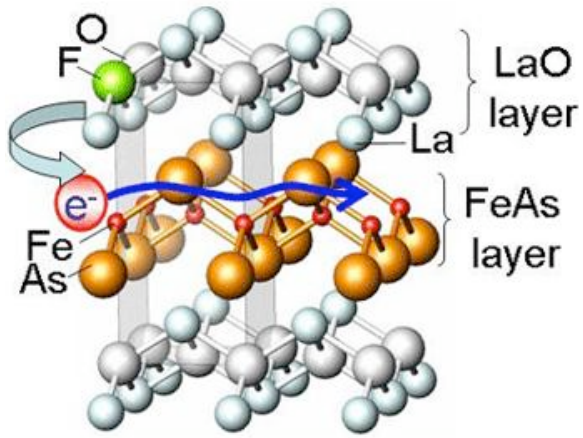
$$\Sigma(\omega) = \begin{pmatrix} \Sigma^{\mathrm{imp}}(\omega) & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$



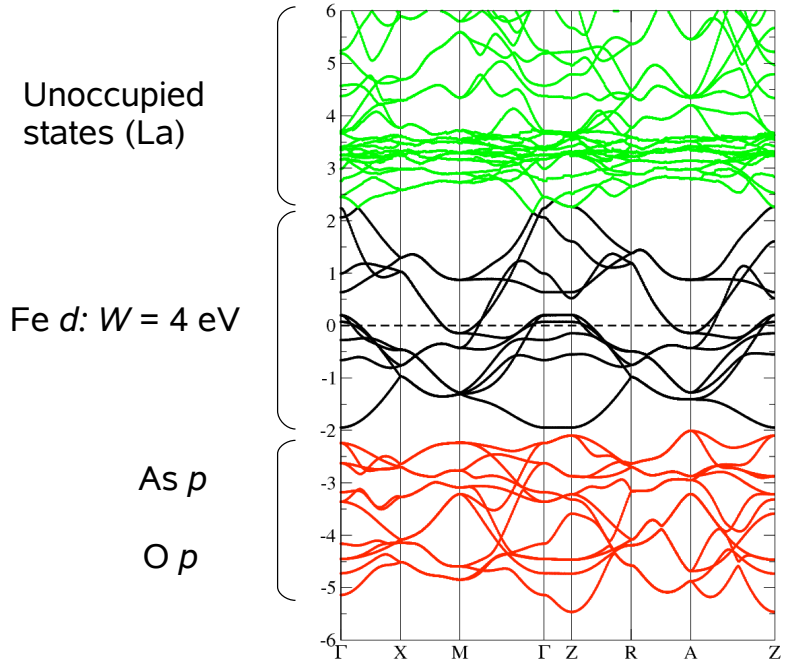
- Self-consistency in large unit cell (Cu + 2 O)
  $\Sigma_{ab}(\omega)$ a 3x3 matrix
- Interaction only on Cu.
  Impurity model is one band, with $\Sigma^{\mathrm{imp}}(\omega)$ a 1x1 matrix

Mix with electronic structure codes

- Cf Lectures by Kotliar next week.
- (Much) more of the same kind of manipulations:
  Extract the Green function of the correlated orbitals.
  E.g. project on Wannier functions.
  Embed the self-energy of the correlated orbital (downfolding).

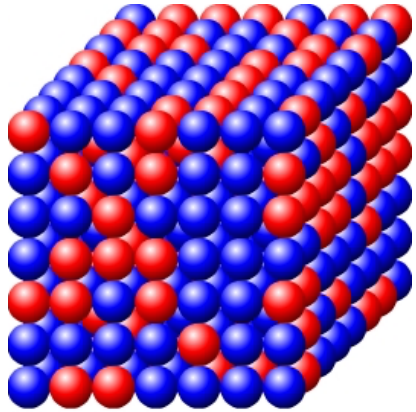


Fe-Based (2008)



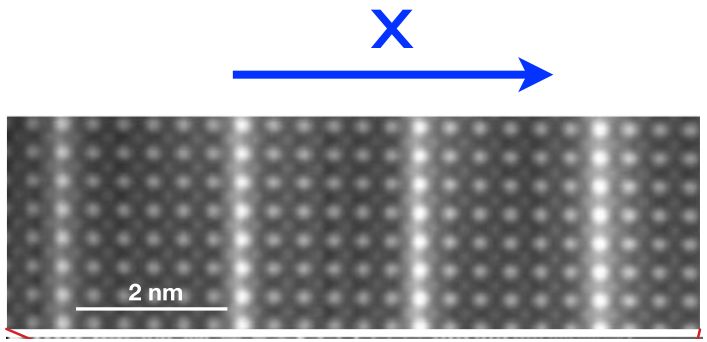
Role of geometry

- Real space DMFT : ultra-cold atoms
  Cf lecture by D. Sénéchal.



Alloy

- Disordered systems *Cf e.g. work of Dobrosavljević et al.*
  - One DMFT impurity for each part (red, blue).
  - Linked by self-consistency condition, taking into account disorder



SrTiO3/LaTiO3
Ohtomo et al, Nature 2002

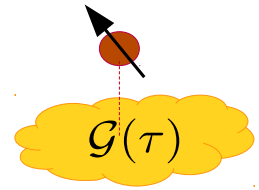
- Correlated interfaces. *Cf e.g. Okamoto & Millis*
  - One DMFT impurity for each layer x
  - Coupled by self-consistency condition

Cluster DMFT

Cf lecture by D. Sénéchal

- DMFT: 1 atom (Anderson impurity) + effective self-consistent bath

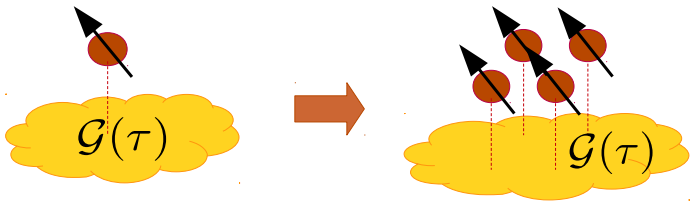
$$\Sigma(k,\omega) \approx \Sigma_{\text{impurity}}(\omega)$$



- Clusters = a systematic expansion around DMFT.
- Control parameter = size of cluster / momentum resolution.
  Systematic benchmarks, cf J. LeBlanc et al., PRX 5 (2015)

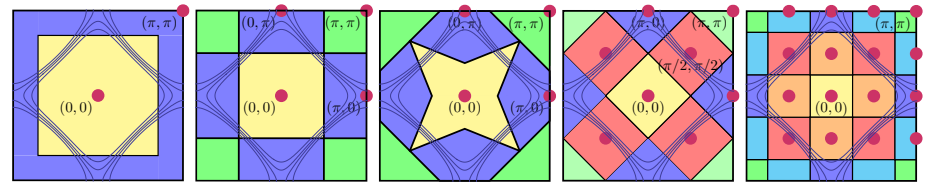
Real space clusters (C-DMFT)

Lichtenstein, Katsnelson 2000
Kotliar et al. 2001



Reciprocal space (DCA)
clusters Brillouin zone patching

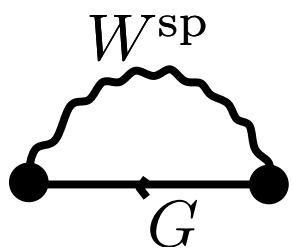
Hettler et al. '98, ...

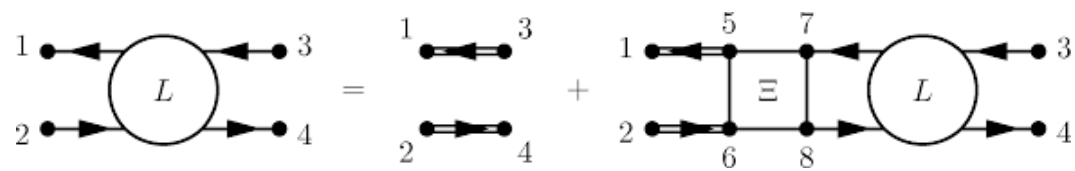


Beyond DMFT

- Mix diagrammatic approximations, e.g. Eliasberg or spin-fluctuation theory, Bethe Salpeter equation, Parquet equations with local atomic (Mott) physics à la DMFT.
  Dual fermions/bosons. *Rubtsov '08,'12,* DΓA *Toschi '07,* Trilex, *T.Ayral, O.Parcollet, 2015-2016*
- Study, compare these methods.
- Need to assemble more complex objects : G(k,ω), Vertex Γ(ω,ν,ν').

$$\Sigma(k, i\omega) \approx$$





Need for a library

- No "general" DMFT code.
- Same for other approaches (e.g. DMRG)
- I want a simple language able to write all of these, and much more.
- Even variants which you have not yet invented !
- Hence an open approach : a library
  Extending Python/C++ to express the basic concepts of our field
  (Green functions, impurity solvers, ...).

Reproducibility

- Methods & algorithms are more and more complex
- Do we have all the elements to reproduce easily a paper ?
  Do we really do "Reproducible Science" ?
- Tools :
  - Jupyter/IPython notebook: a simple tool for reproducibility
    - Analysis script in the notebook, with text, latex formula.
    - A new kind of "paper" where the script from the raw data to the figure is provided, can be changed, reanalyzed.
  - Version control (GitHub)
- But the problem is deeper than that...

Status of numerics

- Numerical & analytical computations on the same footing.
- Algorithmic Physics : study algorithms e.g. for many-body problem.
- Scientific method requires openness as for analytical computations.
  - Codes should be published along with papers.
  - Therefore they should be written to be read !
  - Code review by peers. Improvements, reuse.
- In practice, not the case: writing/testing a code is long, costly
- Need efficient open source toolkits to quickly build computations, enable team work on codes, code review.

The TRIQS project
Goals and organization

TRIQS goals

- Basic blocks for many-body calculations, starting with DMFT and co.
- Simplicity : what is simple should be coded simply !
- High performance :
  - Human time : reduce the cost of writing codes.
  - Machine time : run fast.

TRIQS project : a modular structure

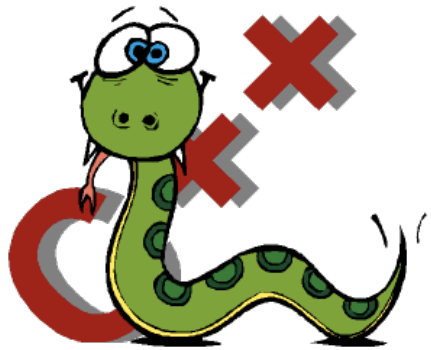
CTHYB
Impurity solver

DFTTools
Interface to electronic structure codes

Your app.

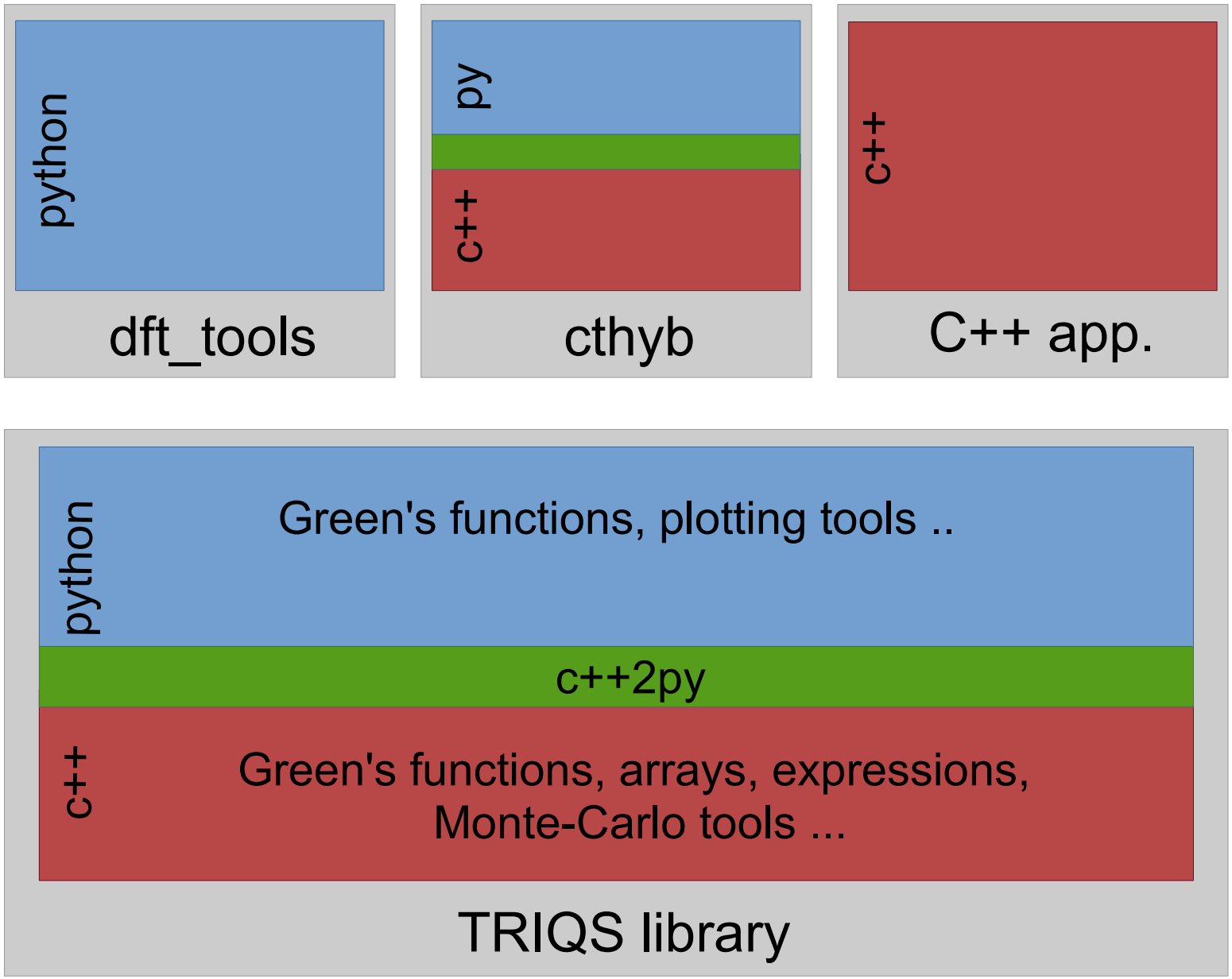
TRIQS library
The basic blocks

Python/C++



- Python
  - Script language, easy to learn.
  - High level part of the code (e.g. assemble a DMFT computation)
  - Data analysis with the same objects/language
    ipython notebook
  - Glue with other codes (e.g. electronic structure codes)
- C++
  - Computational intensive part, e.g. quantum impurity solvers (CTHYB, CTINT).
    When Python is too slow.

TRIQS Python and C++ layers.



• • •

TRIQS: different levels of usage.

- Simplest usage
  - Take an existing template script, e.g. our DMFT computation, change parameters, run, plot, do physics ...
- With Python programming [THIS LECTURE & HANDS-ON]
  - Build your own DMFT self-consistency, using building blocks in Python, package solvers.
- With C++ programming
  - Use the C++ layer to develop a high-performance application.
  - Use the TRIQS tools to wrap it in Python and collaborate using Python as a common language.

TRIQS library.

Olivier Parcollet, Michel Ferrero, Thomas Ayral, Hartmut Hafermann, Igor Krivenko, Laura Messio, Priyanka Seth
Comp. Phys. Comm. 196, 39 (2015), arXiv1504.01952

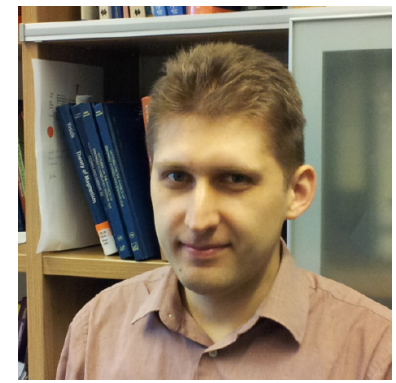
Main developers & contributors

O. Parcollet

Michel Ferrero

Thomas Ayral

Priyanka Seth

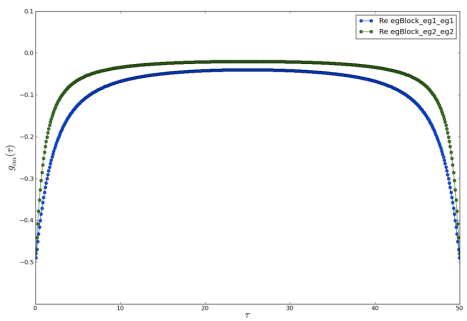


Igor Krivenko

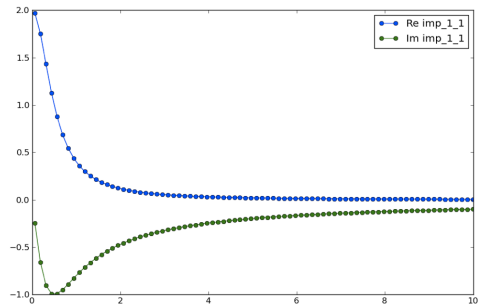
TRIQS library: Python

- Various kinds of Green functions and the associated functions.

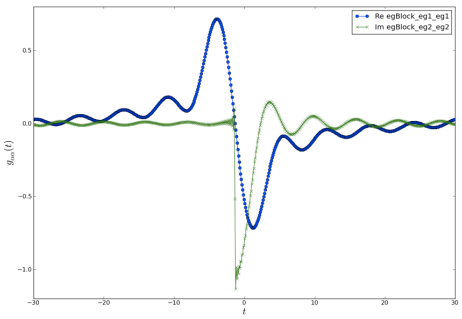
Matsubara time



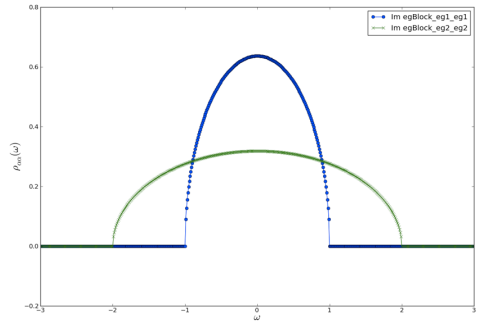
Matsubara frequencies



Real time



Real frequencies



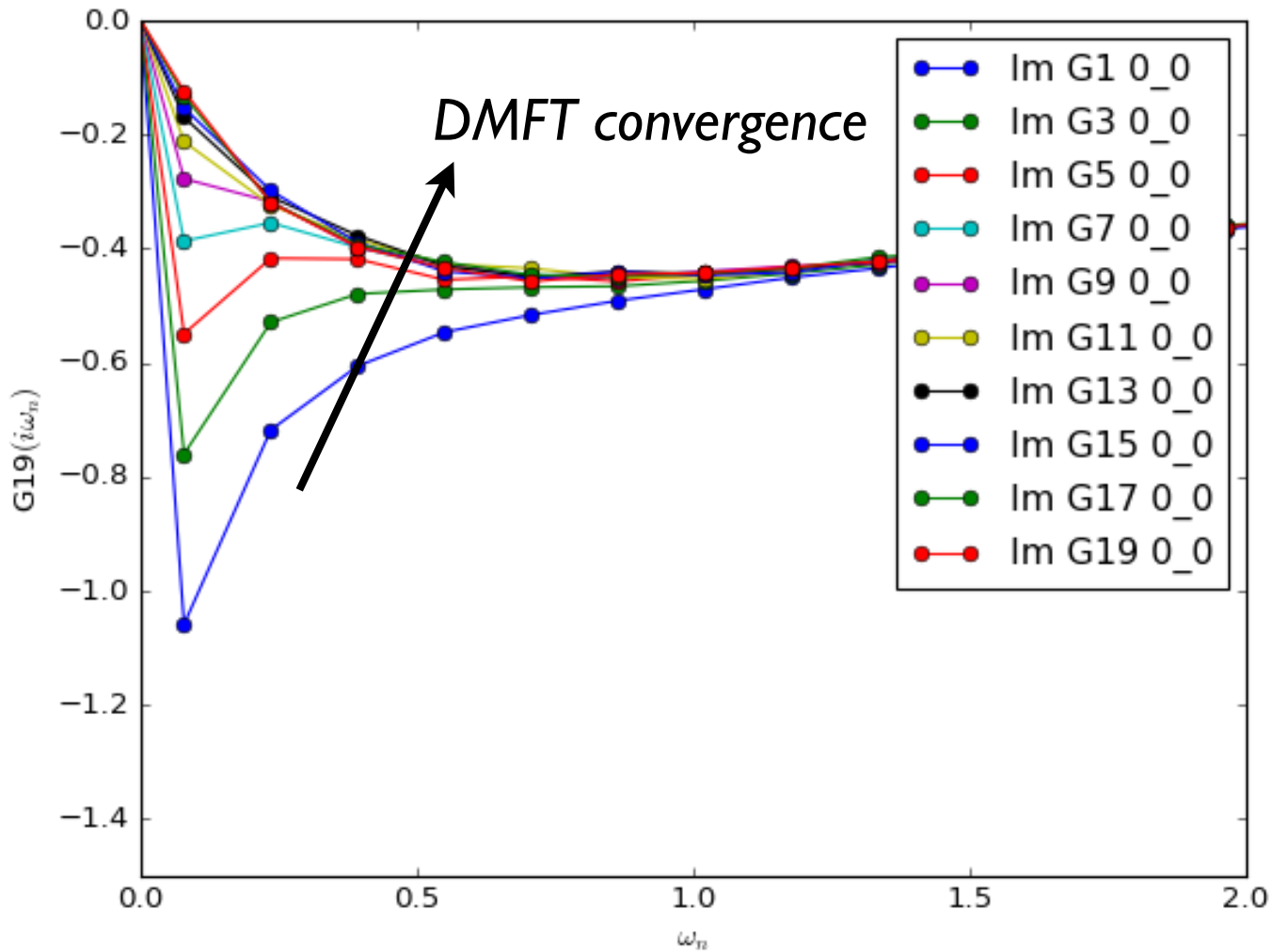
- Many-body operators to write Hamiltonians.
- Simple Bravais Lattices, Brillouin zone, density of states, Hilbert transform.
- Interfaces to save/load in HDF5 files, plot interactively in the ipython notebook.

TRIQS library: C++

- More libraries components:
  - "Green functions" containers (any function on a mesh) $G(\omega)$, $G(k,\omega)$, Vertex $\Gamma(\omega,\nu,\nu')$.
  - Generic Monte Carlo class & error analysis tools.
  - Determinant manipulations (for QMCs).
  - Lattice tools: Bravais Lattices, Brillouin zone, ...
  - Many-body operators.
- Basic blocks and tools , e.g.
  - Multidimensional array class
  - HDF5 light interface in C++

HDF5 file format ...

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



HDF5 file format

- Standard file format used by many projects, e.g. pytables, ALPS.
- 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.
- Also used by ALPS project
  In progress : "standardize" details for even simpler data exchange between TRIQS & ALPS.

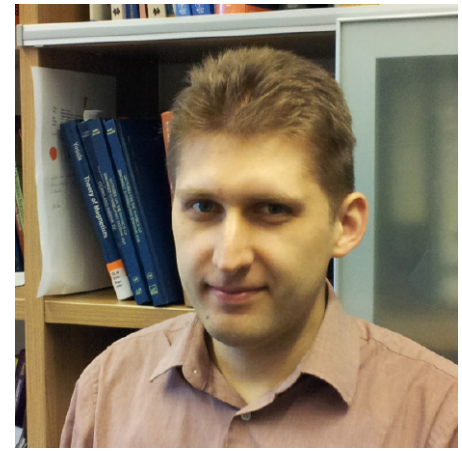
TRIQS/CTHYB.

A Continuous-Time Quantum Monte Carlo Hybridization Expansion Solver for Quantum Impurity Problems

Priyanka Seth, Igor Krivenko, Michel Ferrero, Olivier Parcollet
Comp. Phys. Comm. 200, 274 (2016), arXiv:1507.00175

Developers

Priyanka Seth



Igor Krivenko

Michel Ferrero

O. Parcollet

TRIQS/CTHYB

https://triqs.ipht.cnrs.fr/1.x/applications/cthyb/

cthyb

a generic quantum impurity solver based on the triqs library

Install Documentation Issues About CTHYB

The hybridization-expansion solver

The TRIQS-based hybridization-expansion solver allows to solve the generic problem of a quantum impurity embedded in a conduction bath for an arbitrary local interaction vertex. The "impurity" can be any set of orbitals, on one or several atoms. To be more specific, the Hamiltonian of the problem has the form:

$$\hat{H} = \sum_{k,\alpha} \epsilon_{k,\alpha} c^\dagger_{k,\alpha} c_{k,\alpha} + \sum_{k,\alpha} V_{k,\alpha} (c^\dagger_{k,\alpha} d_\alpha + h.c.) - \mu \sum_\alpha d^\dagger_\alpha d_\alpha + \hat{\sum}_{\alpha\beta} h_{\alpha\beta} d^\dagger_\alpha d_\beta + \frac{1}{2} \sum_{\alpha\beta\gamma\delta} U_{\alpha\beta\gamma\delta} d^\dagger_\alpha d^\dagger_\beta d_\delta d_\gamma .$$

Here the operators $c^\dagger$ construct a fermion in the bath, while the operators $d^\dagger$ construct a fermion on the impurity. In this problem, the hybridization function Δ between the bath and the impurity is given by:

$$\Delta_{\alpha,\beta}(i\omega_n) = \sum_k \frac{V_{k,\alpha} V^*_{k,\beta}}{i\omega_n - \epsilon_{k,\alpha}},$$

so that the non-interacting Green's function of the impurity is:

$$\hat{G}_0^{-1}(i\omega_n) = i\omega_n + \mu - \hat{h} - \hat{\Delta}(i\omega_n).$$

With the knowledge of G_0 and the matrix $U_{\alpha\beta\gamma\delta}$, the quantum impurity solvers find the interacting Green's function G of the problem. Learn how to use it in the Documentation.

Quick search

Go

Enter search terms or a module, class or function name.

CT-HYB :Expansion in hybridization

P. Werner et al, PRL (2006)

$$S_{\rm eff} = -\int_0^\beta c_a^\dagger(\tau) G^{-1}_{0ab}(\tau-\tau') c_b(\tau') + \int_0^\beta d\tau H_{\rm local}(\{c_a^\dagger, c_a\})(\tau)$$

$$G^{-1}_{0ab}(i\omega_n) = (i\omega_n + \mu)\delta_{ab} - \Delta_{ab}(i\omega_n)$$

a,b = 1,N

- Expansion in hybridization :

$$Z = \sum_{n\geq 0} \int_< \prod_{i=1}^n d\tau_i d\tau_i' \sum_{a_i,b_i=1,N} \underbrace{\det_{1\leq i,j\leq n} \left[\Delta_{a_i,b_j}(\tau_i - \tau_j')\right] \mathrm{Tr}\left(\mathcal{T} e^{-\beta H_{\rm local}} \prod_{i=1}^n c_{a_i}^\dagger(\tau_i) c_{b_i}(\tau_i')\right)}_{w(n,\{a_i,b_i\},\{\tau_i\})}$$

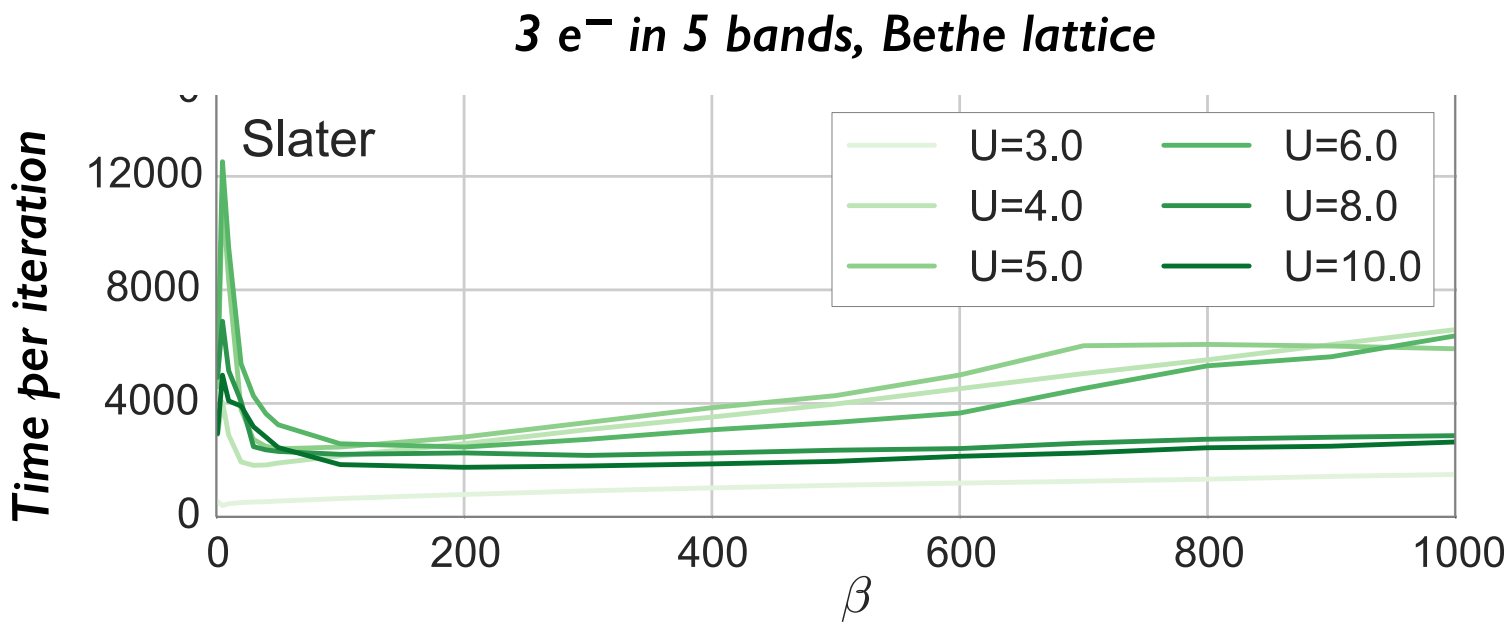
$$G_{ab}(\tau) = \frac{1}{Z}\frac{\delta Z}{\delta \Delta_{ba}(-\tau)}$$

Determinant manipulations

Atomic correlators

Key features of TRIQS/CTHYB

- Treat general Hamiltonian, not only density-density interaction.
  ≠ "segment picture code"
- New 'auto-partitioning' algorithm divides the impurity Hilbert space efficiently without quantum numbers, i.e. no additional user input.
- Use balanced tree (Gull 2008) and truncations on it (Yee, Sémon et al. 2014) for atomic correlators. Much better scaling at low T



Many-body operators

- Write Hamiltonian (or any operator) in a natural way, as polynomials of c, c^+.
- Simple manipulation of fermionic second quantized operators with normal ordering and (basic) simplifications, Cf Tutorial.
- Part of TRIQS library.

```
from pytriqs.operators.operators import Operator, n, c_dag, c

# Spin operators
Sp = c_dag("up",0)*c("dn",0)          # S_+
Sz = 0.5*(n("up",0) - n("dn",0))      # S_z
S2 = Sz*Sz + (Sp*Sm + Sm*Sp)/2        # S^2

# The Hamiltonian of a half-filled Hubbard atom
U = 1.0
H = U*(n("up",0) - 0.5)*(n("dn",0) - 0.5) - U/4
```

Write more complex local Hamiltonians

- Atomic hamiltonian (impurities) can becomes quite complex, e.g.

$$H = \frac{1}{2} \sum_{ijkl,\sigma\sigma'} U_{ijkl} d^\dagger_{i\sigma} d^\dagger_{j\sigma'} d_{l\sigma'} d_{k\sigma}$$

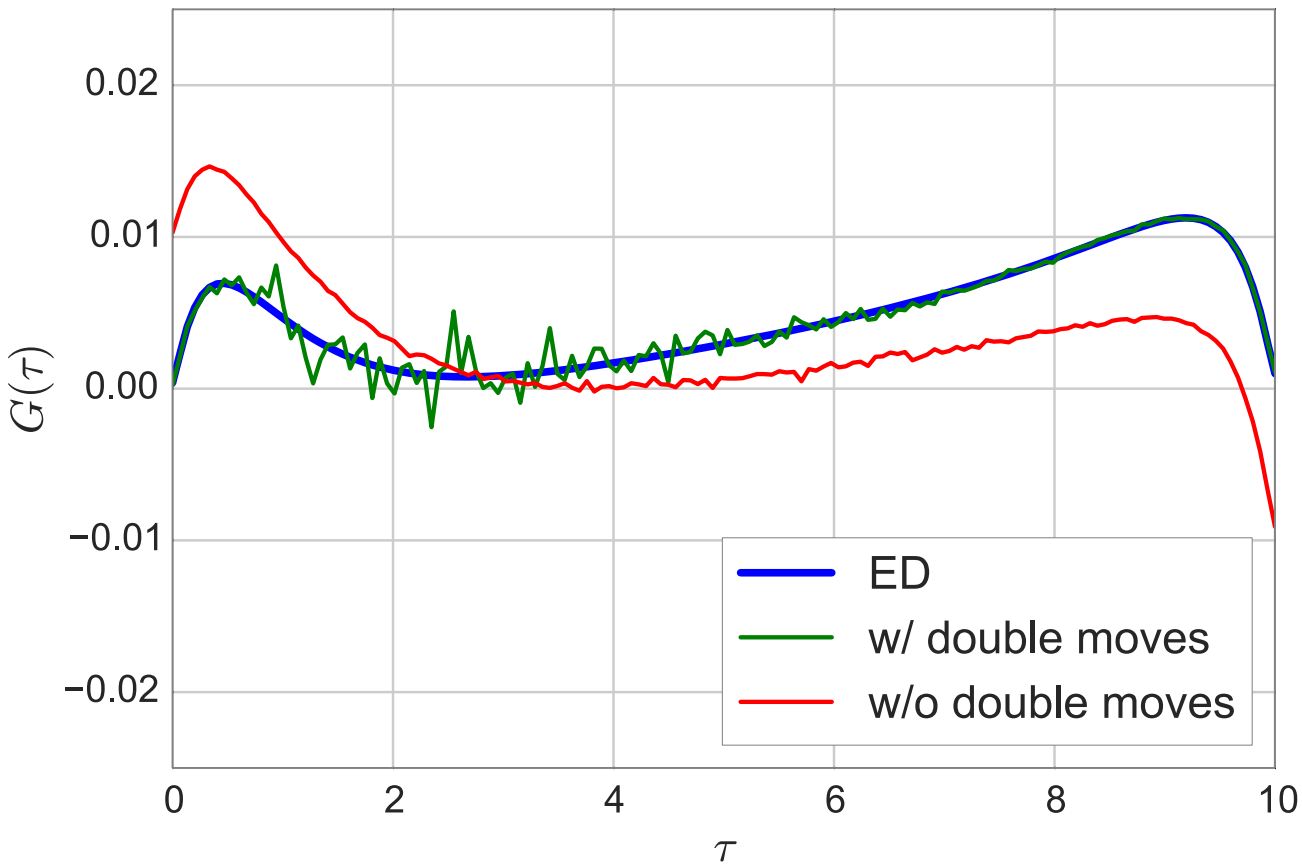
- Corresponding code

```
from pytriqs.operators import Operator, c, c_dag
spin, orb = ['up','down'], ['d0','d1','d2','d3','d5']

H = Operator()
for s1, s2 in product(spin, spin):
    for a1, a2, a3, a4 in product(orb, orb, orb, orb):
        U = U_matrix[orb.index(a1),orb.index(a2),
                     orb.index(a3),orb.index(a4)]
        H += 0.5 * U * c_dag(s1,a1) * c_dag(s2,a2) * c(s2,a4) * c(s1,a3)
```

- H is an input of the CTHYB solver

Features of TRIQS/CTHYB

- Four-operator insert/delete ensure ergodicity of common Hamiltonians (e.g. Kanamori), even in disordered phases (≠ Sémon et al. 2014)



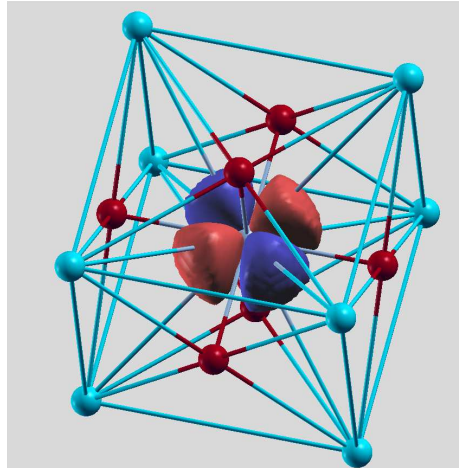
TRIQS/DFTTools.

Interface to electronic structure codes

Markus Aichhorn, Leonid Pourovskii, Priyanka Seth, Veronica Vildosola, Manuel Zingl, Oleg E. Peil, Xiaoyu Deng, Jernej Mravlje, Gernot J. Kraberger, Cyril Martins, M. Ferrero, O. Parcollet
Comp. Phys. Comm. 204, 200 (2016), arXiv:1511.01302

TRIQS / DFTTools

- Ab-initio DMFT + DFT.
  Cf lectures by Kotliar & Haule next week.
  Not covered in these hands-on.
- A TRIQS python interface with a growing number of electronic structure codes:
  - Wien2k
  - Wannier90
  - Vasp (in progress, to be published).
  - Fleur



$SrVO_3$

Developers & some contributors

Leonid Pourovskii

Markus Aichhorn

Veronica Vildosola

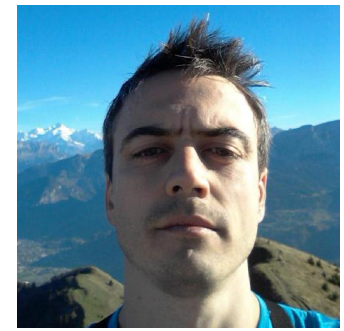
Michel Ferrero

Priyanka Seth

Jernej Mravlje

Gernot Kraberger

Manuel Zingl



Oleg Peil

Web sites

TRIQS web site

https://triqs.ipht.cnrs.fr

triqs

a Toolbox for Research on Interacting Quantum Systems

Install Reference Tutorials Applications Issues About TRIQS

Welcome

TRIQS (**T**oolbox for **R**esearch on **I**nteracting **Q**uantum **S**ystems) is a scientific project providing a set of C++ and Python libraries to develop new tools for the study of interacting quantum systems.

The goal of this toolkit is to provide high level, efficient and simple to use libraries in C++ and Python, and to promote the use of modern programming techniques.

TRIQS is free software distributed under the GPL license.

TRIQS 1.4-dev

This is the homepage of the TRIQS release 1.4 (in development). For the changes in 1.4, Cf changelog page

TRIQS applications

Based on the TRIQS toolkit, several full-fledged applications are also available. They allow for example to solve a generic quantum impurity model or to run a complete LDA+DMFT calculation.

Developed in a collaboration between IPhT Saclay and Ecole Polytechnique since 2005, the TRIQS library and applications have allowed us to address questions as diverse as:

- Momentum-selective aspects on cuprate superconductors (with various cluster DMFT methods)
- Degree of correlation in iron-based superconductors (within an LDA+DMFT approach)
- Fermionic Mott transition and exploration of Sarma phase in cold-atoms

Python & C++

The libraries exist at two complementary levels: on the one hand, C++ libraries allow to quickly develop high-performance low-level codes; on the other hand python libraries implement the most common many-body objects, like Green's functions, that can be manipulated easily in python scripts.

This duality is a real advantage in the development of new many-body tools. Critical parts where performance is essential can be written in C++ (e.g. a quantum impurity solver) while the data analysis, preparation of the inputs or interface with other programs can be done at the very user-friendly python level.

cea

ÉCOLE POLYTECHNIQUE

cnrs

erc

European Research Council

Established by the European Commission

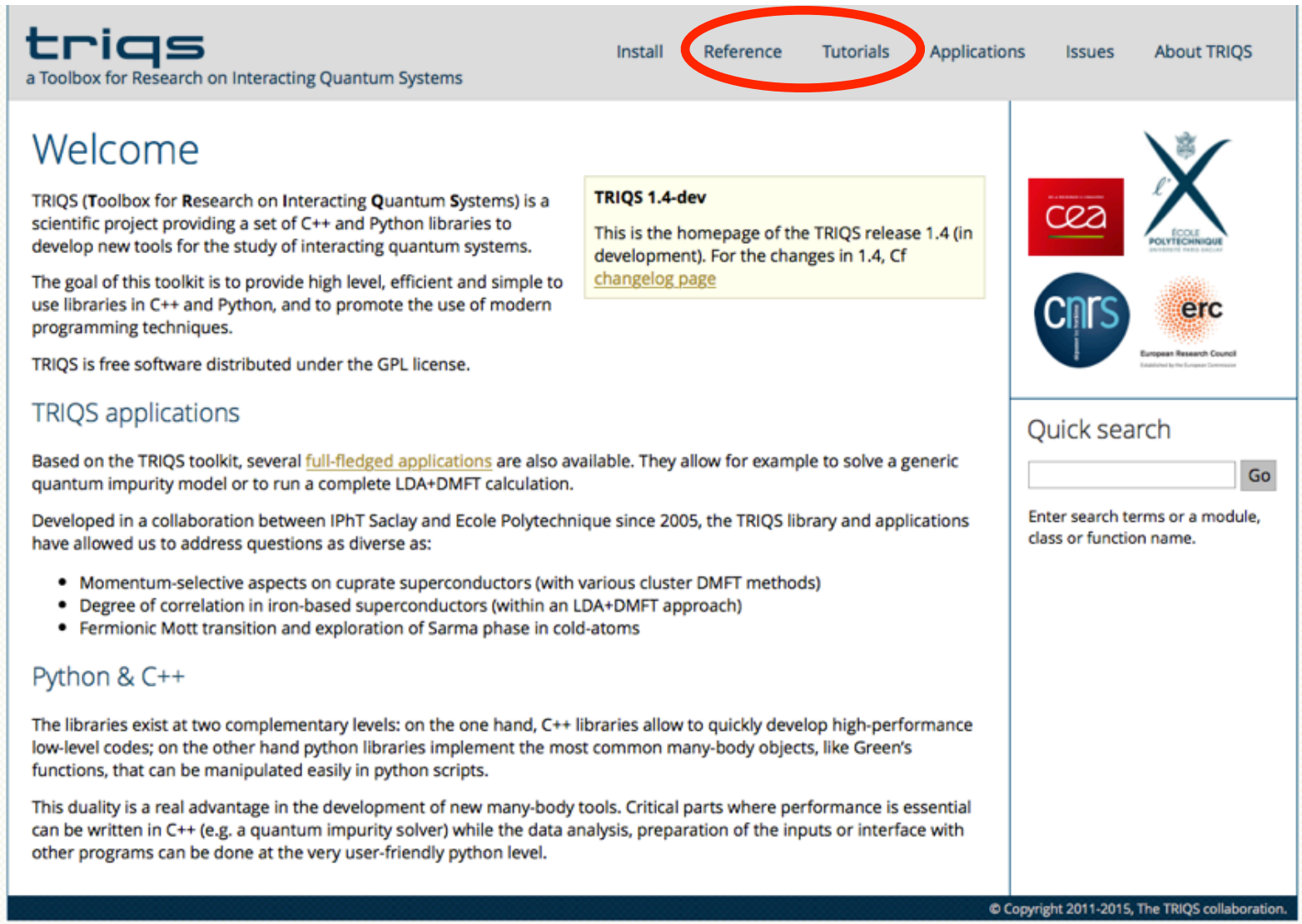
Quick search

Go

Enter search terms or a module, class or function name.

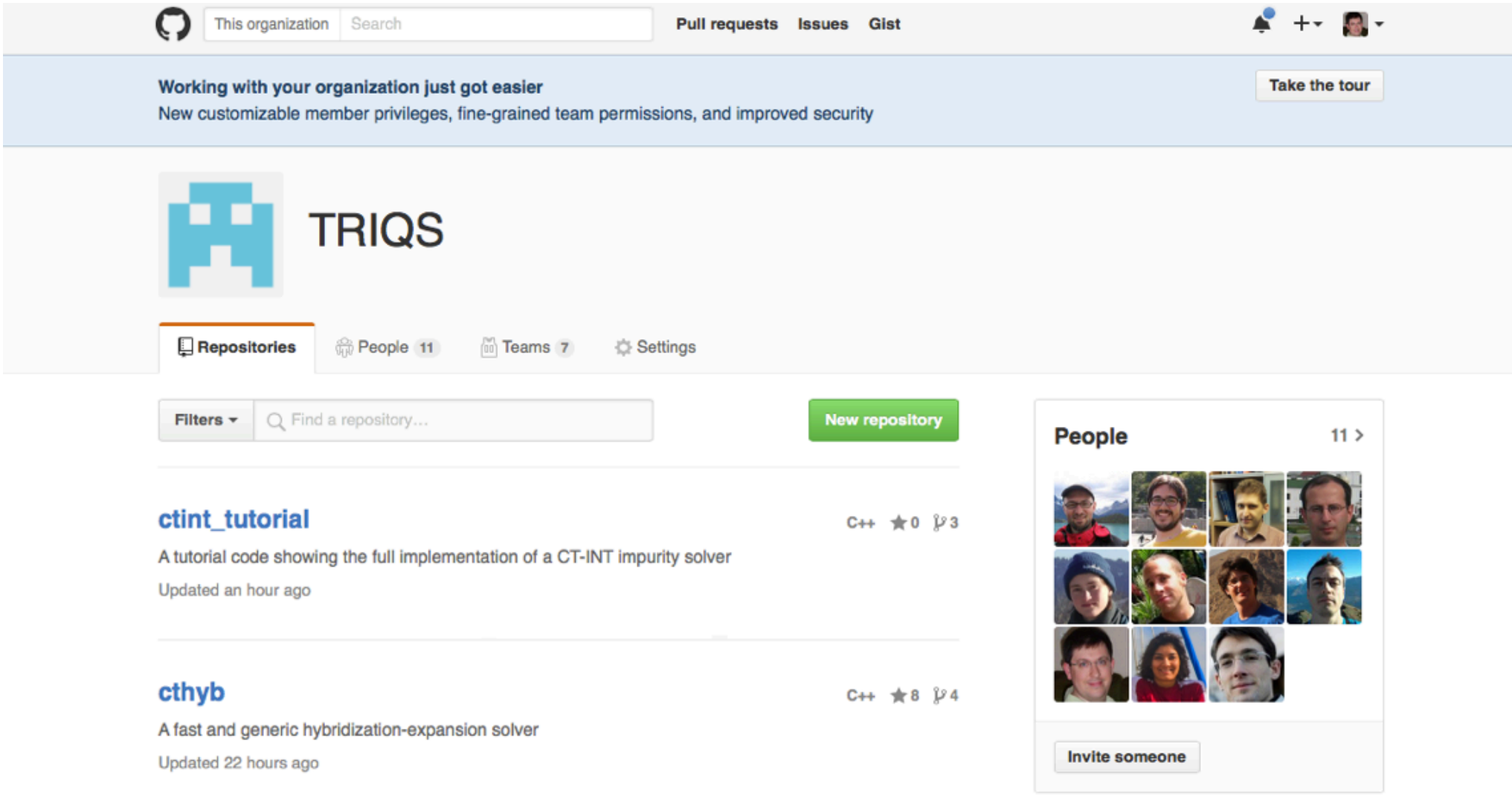
Documentation

- Python & C++. Mainly reference doc, with examples.
- Python tools: sphinx, rst (wiki + code + latex), automatic doc.



The Github site

https://github.com/TRIQS



How to contribute ?

- We are still a small team, help is welcome, in various ways
  - Cite TRIQS papers if you used some parts of it.
  - Issue reporting.
  - Short contribution : “pull request” on github
  - Larger contribution : coordinate with the developers.

Bug reporting

- When a problem is found in the library :
  bugs, bad or unnatural behavior (API), speed issue, lack of doc.
  Then :
  - Try to narrow the problem,
    with a little test for us to reproduce.
  - Open an issue on github.
  - “It does not work” is ***not*** a bug report, it is ***useless***.
- We will make a little exercise in hands-on #2
- We then fix the problem at its source (bug, design flaw) and add a non regression test.

TRIQS library for C++ programmers

The core of the TRIQS project

A few highlights

Zero cost abstraction

- What is simple should be coded simply
- Code should be as simple as math, high level and yet fast.
- High abstraction ***and*** speed.
- Contrary to a common wrong idea, it is possible.
- Let's illustrate this with 2 examples

C++ example with a Green function

- A Green function in Matsubara frequency, on a Brillouin zone.

$$G(k, i\omega_n) = \frac{1}{i\omega_n - 2t\big(\cos(k_x) + \cos(k_y)\big) - \dfrac{1}{i\omega_n + 2}}$$

- C++ code snippet

```
// A Brillouin zone of a Bravais lattice
auto bz = brillouin_zone{bravais_lattice{{{1, 0}, {0, 1}}}};

// A Green function G(k,omega)
double beta = 10;
auto g_k_iw = gf<cartesian_product<brillouin_zone, imfreq>>{
              {{bz, 100}, {beta, Fermion}},
               {1, 1}
              };

clef::placeholder<0> iw_; // Dummy variables
clef::placeholder<1> k_;

g_k_iw(k_, iw_) << 1 / (iw_ - 2*t*(cos(k_(0)) + cos(k_(1))) - 1 / (iw_ + 2));
```

mesh

matrix size

- Loops are implicit, optimized, with proper memory traversal, ...

Lazy assignment techniques

- More complex example (from a ***real*** code in my group).

$$\chi^{0\,\sigma\sigma'}_{\nu\nu'\omega} = \beta(g^{0\,\sigma}_{\nu} g^{0\,\sigma'}_{\nu'} \delta_{\omega} - g^{0\,\sigma}_{\nu} g^{0\,\sigma}_{\nu+\omega} \delta_{\nu\nu'} \delta_{\sigma\sigma'})$$

- C++ code snippet

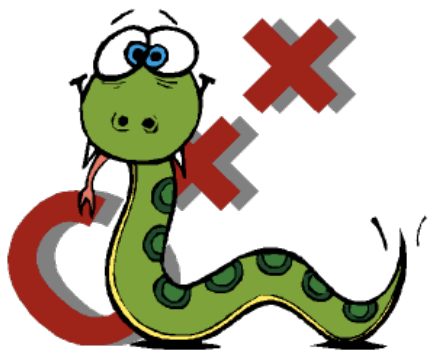
```
chi0(s_, sp_)(nu_, nup_, om_) <<
    beta * (g[s_](nu_) * g[sp_](nup_) * kronecker(om_))
  - beta * (g[s_](nu_) * g[s_](nu_ + om_)  * kronecker(nu_, nup_) * kronecker(s_, sp_));
```

- Express intent, leave the details to the compiler & library writer.
- Save 5 loops, no need to think about bounds, memory traversal order.
- Easy parallelization, like parallel_foreach, gpu_foreach (TBB, AMP, ...)

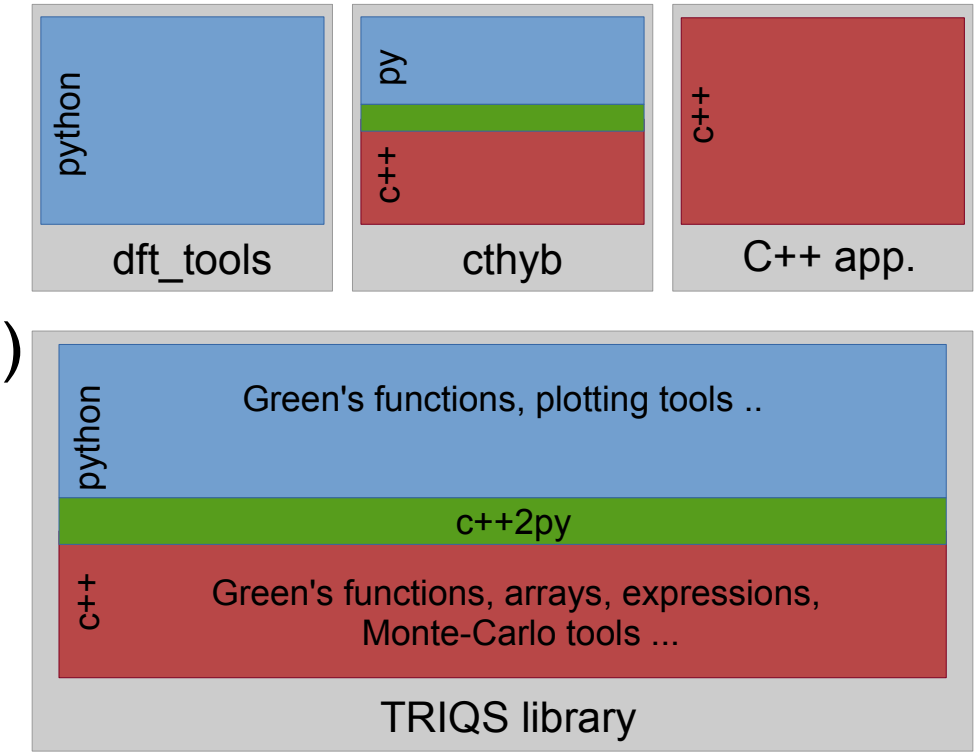
CT-INT

- A demo code given with the TRIQS paper.
- A full C++ implementation for CT-INT, for 1 band.
- Parallel (MPI), with its Python interface.
- Illustration of the use of TRIQS C++ lib, 200 lines of C++.
- git@github.com:TRIQS/ctint_tutorial.git

Gluing Python and C++



How to efficiently glue C++ & Python ?



- Issue: pass objects Python $\leftrightarrow$ C++ (e.g. Green's functions, slices of them)
- It is critical for the TRIQS project.
- Not an easy problem in software engineering. Python & C++ are quite different.
- TRIQS solution : c++2py
- In most cases, it is entirely automatic.
  - Takes C++ code, extract the functions, classes, their documentation (thanks to libClang & Google compiler team !) ...
  - ... write glueing code, the documentation

Questions ?

Let us start the hands-on ...

Hands on: the menu

Files Running Clusters

Select items to perform actions on them.

- other
- results_two_bands
- 00a-Introducing_the_ipython_notebook.ipynb
- 00b-Matplotlib_Examples.ipynb
- 01-Greens_functions.ipynb
- 02-Greens_functions_solution.ipynb
- 03-Archiving_your_data.ipynb
- 04-IPT_and_DMFT.ipynb
- 05-IPT_and_DMFT_solution.ipynb
- 06-Operators.ipynb
- 07-Introduction_to_the_CTHYB_solver.ipynb
- 08-Single-orbital_Hubbard_with_CTQMC.ipynb
- 09-Single-orbital_Hubbard_with_CTQMC_solution.ipynb
- 10-Two-orbital_Hubbard_with_CTQMC.ipynb
- 11-Two-orbital_Hubbard_with_CTQMC_solution.ipynb

Practical details

- TRIQS and apps are already installed on the server
- Everyone run his/her own ipython on one core on the server.
- Launch a ipython on a node (with 6h time limit).

```
qsub /soft/bin/run_jupyter
```

- Find the node attributed to you

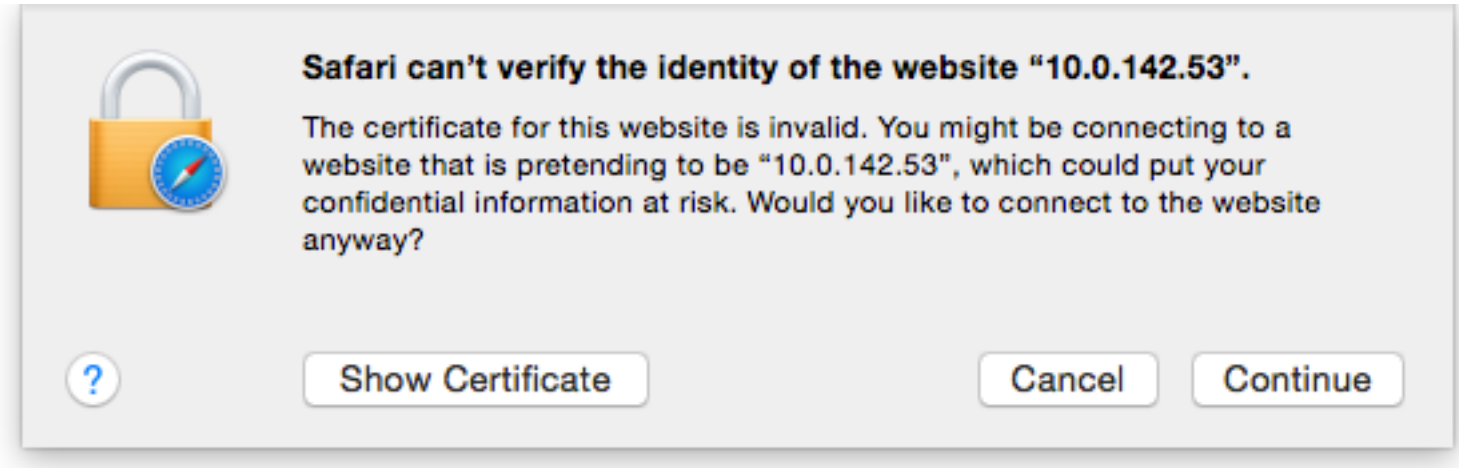
```
more ~/jupyter_url.txt

> Starting jupyter sesion on https://10.0.XXX.XX:YYYY
```

- Clone locally the tutorials (local copy of https://github.com/TRIQS/tutorials.git)

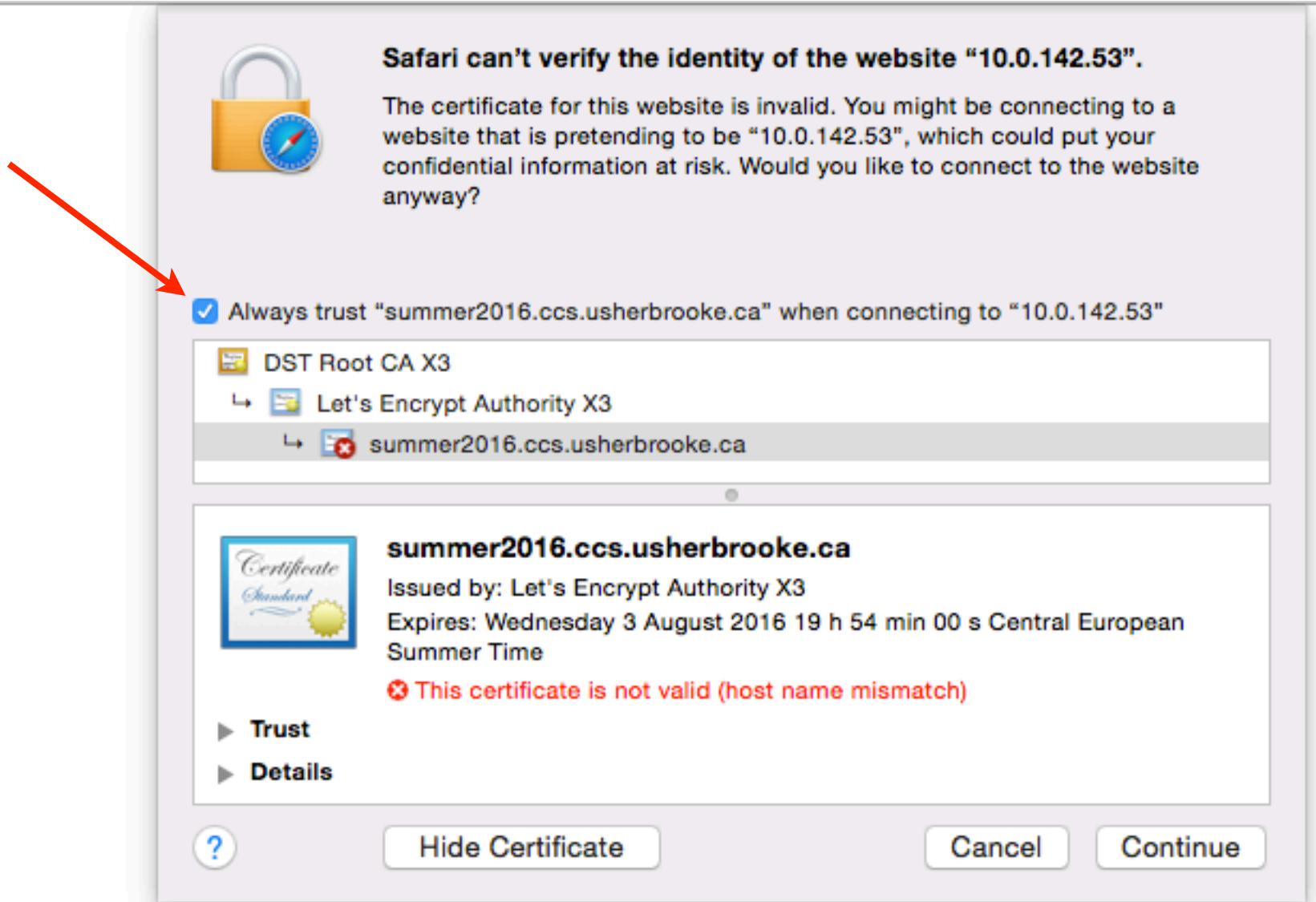
```
git clone /soft/public_soft/triqs/src/tutorials tutorials
```

- Open browser at https://10.0.XXX.XX:YYYY
  You need to reenter your password.
- At end, qdel job_id your ipython job (use qstat to the job id).

Safari on OS X



Safari on OS X



+ *password to change Keychain Access*

In the browser, you should see something like

- Click on tutorials, then python

jupyter

Logout

Files | Running | Clusters

Select items to perform actions on them.

Upload | New | ↻

/ tutorials / python

..

- 00a-Introducing_the_ipython_notebook.ipynb
- 00b-Matplotlib_Examples.ipynb
- 01-Greens_functions.ipynb
- 02-Greens_functions_solution.ipynb
- 03-Archiving_your_data.ipynb
- 04-IPT_and_DMFT.ipynb
- 05-IPT_and_DMFT_solution.ipynb
- 06-Operators.ipynb
- 07-Introduction_to_the_CTHYB_solver.ipynb
- 08-Single-orbital_Hubbard_with_CTQMC.ipynb
- 09-Single-orbital_Hubbard_with_CTQMC_solution.ipynb
- 10-Two-orbital_Hubbard_with_CTQMC.ipynb
- 11-Two-orbital_Hubbard_with_CTQMC_solution.ipynb