

PythonからC++ライブラリを 呼び出す話

T. Aoyama (KEK)

第13回 HPC-Phys勉強会

2021/11/25

Outline

- はじめに
- Python extension を書く
- 事例紹介: PyBridge++
- 落とし穴
- まとめ

Python Programming Language

- Pythonとは
 - 動的型付けのオブジェクト指向言語。
 - 簡潔で表現力の高いコードでプログラムを記述できる。
 - 様々なライブラリ・フレームワークなどが開発され、エコシステムを形成している。
 - 近年は、AIや機械学習の文脈でよく利用されている。
 - 様々なプラットフォームで動作する。UNIX/Linux, Windows, MacOS など。intel Python などのdistributionもある。



Python Programming Language

- Python を”拡張”する:
 - ▶ Pythonプログラムの Hot Spot を C/C++ で高速化する
 - ▶ 既存の C/C++ ライブラリを Python から使えるようにする
 - 今回は後者の視点
- 数値シミュレーションの文脈では:
 - Rapid prototyping によるプログラム開発
 - 数値計算と解析を統合
 - 数値計算 → 結果の出力 → 統計処理・可視化
 - jupyter notebook 上で一連の処理を実行
 - パラメータファイルと計算ロジックを融合
 - cf. スクリプト言語をフロントエンドとして埋め込み

Writing Python Extension

- Python Extension を書くには:

- CPython (標準実装, C API を提供)
- Cython (Python-like な文法で extension を記述する言語)
- SWIG (グルーコードを生成する汎用ツール)
- Boost.Python (C++用の Python binding)
- **pybind11** (C++ テンプレートライブラリ)
- and others.



Writing Python Extension

C++ code

```
class Field {  
public:  
    Field(int size) { ... }  
    double norm() const { ... }  
};
```

Python binding
by pybind11

```
#include <pybind11/pybind11.h>  
  
namespace py = pybind11;  
  
PYBIND11_MODULE(pybridge, m) {  
    py::class_<Field>(m, "Field")  
        .def(py::init<int>())  
        .def("norm", Field::norm);  
}
```

using from Python

```
import pybridge as pyb  
  
f = pyb.Field(size)  
v = f.norm()
```

具体例:

C++ で記述された Field クラス (Vectorのようなもの) を pybind11 により Python から使えるようにする

Writing Python Extension

C++ code

```
class Field {  
public:  
    Field(int size) { ... }  
    double norm() const { ... }  
};
```

C++で"Field"クラスを定義。
int引数のコンストラクタと
doubleを返すnorm()メソッドを
導入。

Python binding
by pybind11

```
#include <pybind11/pybind11.h>  
  
namespace py = pybind11;  
  
PYBIND11_MODULE(pybridge, m) {  
    py::class_<Field>(m, "Field")  
        .def(py::init<int>())  
        .def("norm", Field::norm);  
}
```

using from Python

```
import pybridge as pyb  
  
f = pyb.Field(size)  
v = f.norm()
```

Writing Python Extension

C++ code

```
class Field {  
public:  
    Field(int size) { ... }  
    double norm() const { ... }  
};
```

まず pybind11.h ヘッダを
include

Python binding
by pybind11

```
#include <pybind11/pybind11.h>  
  
namespace py = pybind11;  
  
PYBIND11_MODULE(pybridge, m) {  
    py::class_<Field>(m, "Field")  
        .def(py::init<int>())  
        .def("norm", Field::norm);  
}
```

“pybridge”モジュールを定義
するマクロ

“Field”クラスを Python 側に
導入

コンストラクタとnorm()
メソッドを追加

using from Python

```
import pybridge as pyb  
  
f = pyb.Field(size)  
v = f.norm()
```

.def() 指示はチェーンできる

Writing Python Extension

C++ code

```
class Field {  
public:  
    Field(int size) { ... }  
    double norm() const { ... }  
};
```

Python binding
by pybind11

```
#include <pybind11/pybind11.h>  
  
namespace py = pybind11;  
  
PYBIND11_MODULE(pybridge, m) {  
    py::class_<Field>(m, "Field")  
        .def(py::init<int>())  
        .def("norm", Field::norm);  
}
```

using from Python

```
import pybridge as pyb  
  
f = pyb.Field(size)  
v = f.norm()
```

“pybridge”モジュールを
import

Fieldクラスのインスタンスを
作り、メソッドを呼び出す

Writing Python Extension

- Buildのしかた
 - (1) pybind11 をインストール
 - pip などのパッケージ管理ツール、または
 - github からソースを取得. (make は不要)
 - <https://github.com/pybind/pybind11>
 - (2) コンパイル

```
g++ -O3 -shared -fPIC -std=c++11 \  
    `python3 -m pybind11 --include` \  
    -o pybridge`python3-config --extension-suffix` \  
    pybridge.cpp [...]
```

pybridge.cpython-35m-x86_64-linux-gnu.so が生成される。(下線部は環境依存)

Writing Python Extension

- pybind11 とは
 - C++の関数やクラスをPythonから呼び出せるようにする header-only な template library。
- pybind11の機能
 - 関数 任意の引数型・戻り値型の関数を bind.
 - クラス Pythonのクラスとして bind.
 - インスタンスメソッド・クラスメソッドを expose.
 - 継承・仮想関数・オペレータオーバーロードに対応
 - 列挙型など
 - STLコンテナ
 - Python の Array, Dictionary, Tuple に map される
 - 例外処理

Writing Python Extension

- pybind11 とは
 - pybind11の機能 (con't)
 - Python のデータ型
 - 配列や辞書型は `py::array`, `py::dict` として import される。
STLコンテナのように扱える
 - docstring
 - docstring を記述できる。help で参照可
 - `py::class_<Field>(m, "Field", R"pydoc( description... )");`
 - numpyのサポート
 - numpyの多次元配列と相互利用可

Lattice QCD

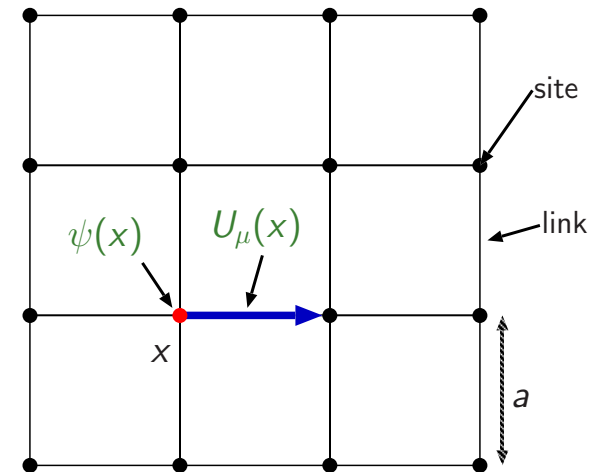
- QCD (Quantum ChromoDynamics)
 - クォーク・グルーオン間に働く「強い相互作用」の理論
 - 低エネルギー(長距離)で結合定数が増大, 解析的に解けない
 - 数値的手法による研究

- 格子QCD

- 離散化された時空上の場の理論
 - クォーク場: 格子点上の自由度 [$\psi(x)$]
 - グルーオン場: リンク上のSU(N_c) [$U_\mu(x)$]

- 経路積分量子化

- モンテカルロ法による数値シミュレーション

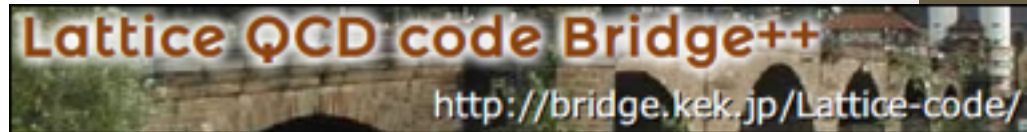


Python and Lattice QCD

- 格子QCDシミュレーションの Python ライブラリ:
 - “pyQCD: A Native Lattice Simulation Package for Python”, M. Spraggs, PoS LATTICE 2014 (2014) 050.
 - Pythonライブラリと拡張可能なAPIの定義。1ノードで格子QCDシミュレーションを実行。
 - Boost.Pythonを使って内部のC++コードをwrapする。GPU対応、Dirac演算子のinversionを高速化。
 - GPT (Grid Python Toolkit) by C. Lehner
 - Grid Lattice QCDライブラリ (P. Boyle) をベースにした Python toolkit。
 - Grid は MPI, OpenMP, SIMD, SIMT によるデータ並列環境を提供。
 - データ並列処理は CPython で書かれ、QCDのアルゴリズム部分は Python module として実装。 .
 - **PyBridge++** by Bridge++ project team
 - Bridge++格子QCDコードセットの Python binding。pybind11で実装。



Bridge++

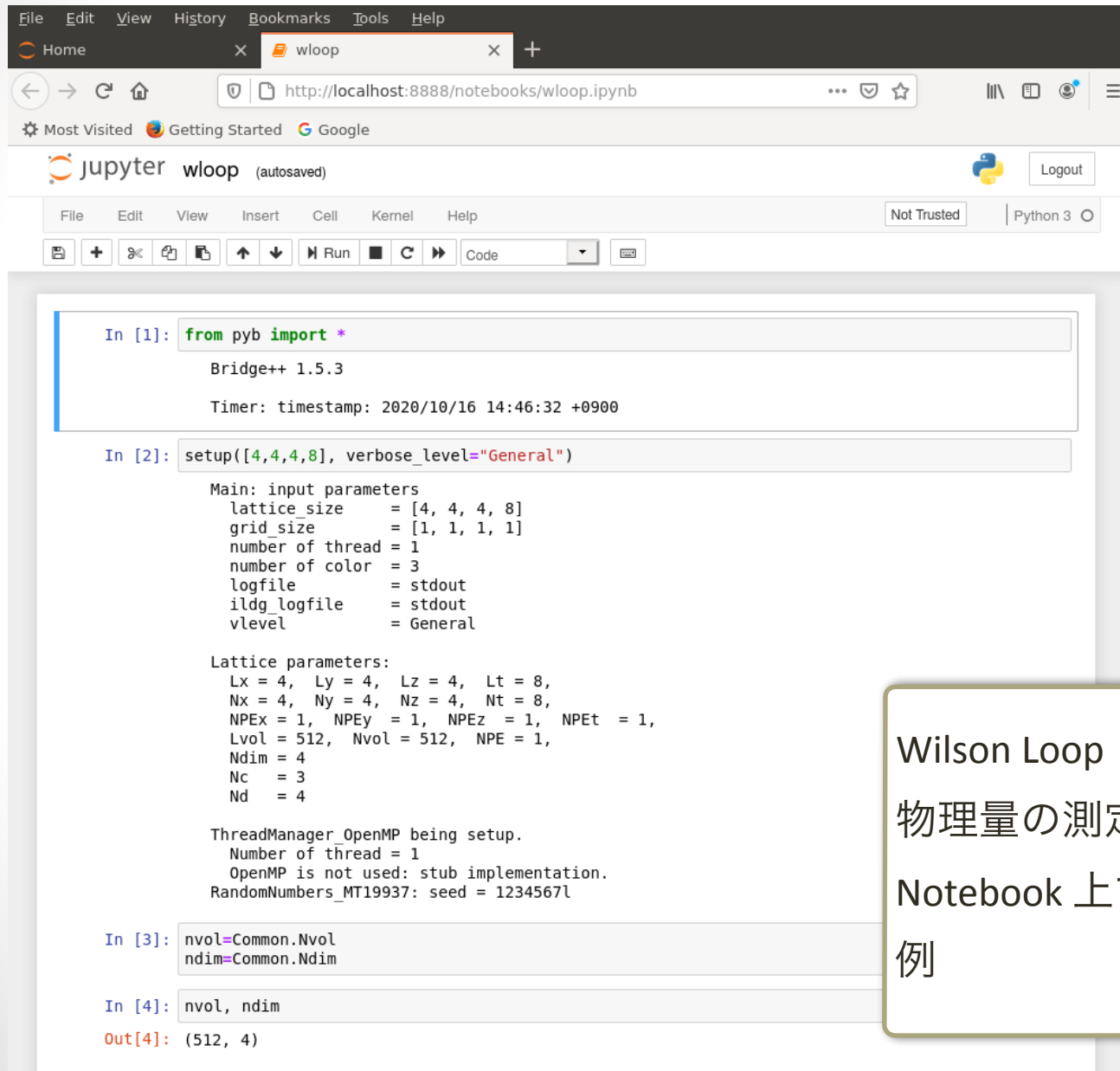


- Overview
 - 格子ゲージ理論シミュレーションのための汎用コードセット
 - オブジェクト指向デザイン、C++ で記述
 - 開発ポリシー
 - 可読性・拡張性・可搬性のある高速なコードセット
- History
 - 2009年スタート、2012年に version 1.0 を正式リリース
 - 最新版は version 1.6.1 (2021年6月).
 - 国内外の研究グループでの利用実績. 引用件数 55件.
- Members
 - Y. Akahoshi, S. Aoki, H. Nemura (YITP), T. Aoyama, H. Matsufuru (KEK), I. Kanamori (RIKEN), K. Kanaya, Y. Taniguchi (Tsukuba), Y. Namekawa (Kyoto), and contributors.

Bridge++ and Python

- Bridge++ の機能: C++ class として構成されている
 - Fields (時空上の自由度), 線形代数演算,
 - Dirac演算子の様々な定式化, 線形方程式ソルバ,
 - Monte Carlo アルゴリズム, ゲージおよびフェルミオン作用,
 - Plaquette, Wilson Loop, quark相関関数など,
 - ファイル入出力, 乱数生成, その他ツール,
 - MPIおよびOpenMP で並列化.
- Bridge++ のPython binding:
 - クラス・関数の “Wrapper” を pybind11 で作成。
 - 一般的なユースケースに対するサンプルプログラムをPythonに移植
 - Python向け拡張
 - Field クラスを numpy array として扱えるようにする。
 - Version 1.6.0 (2021年3月) リリースに導入

PyBridge++ on Jupyter notebook

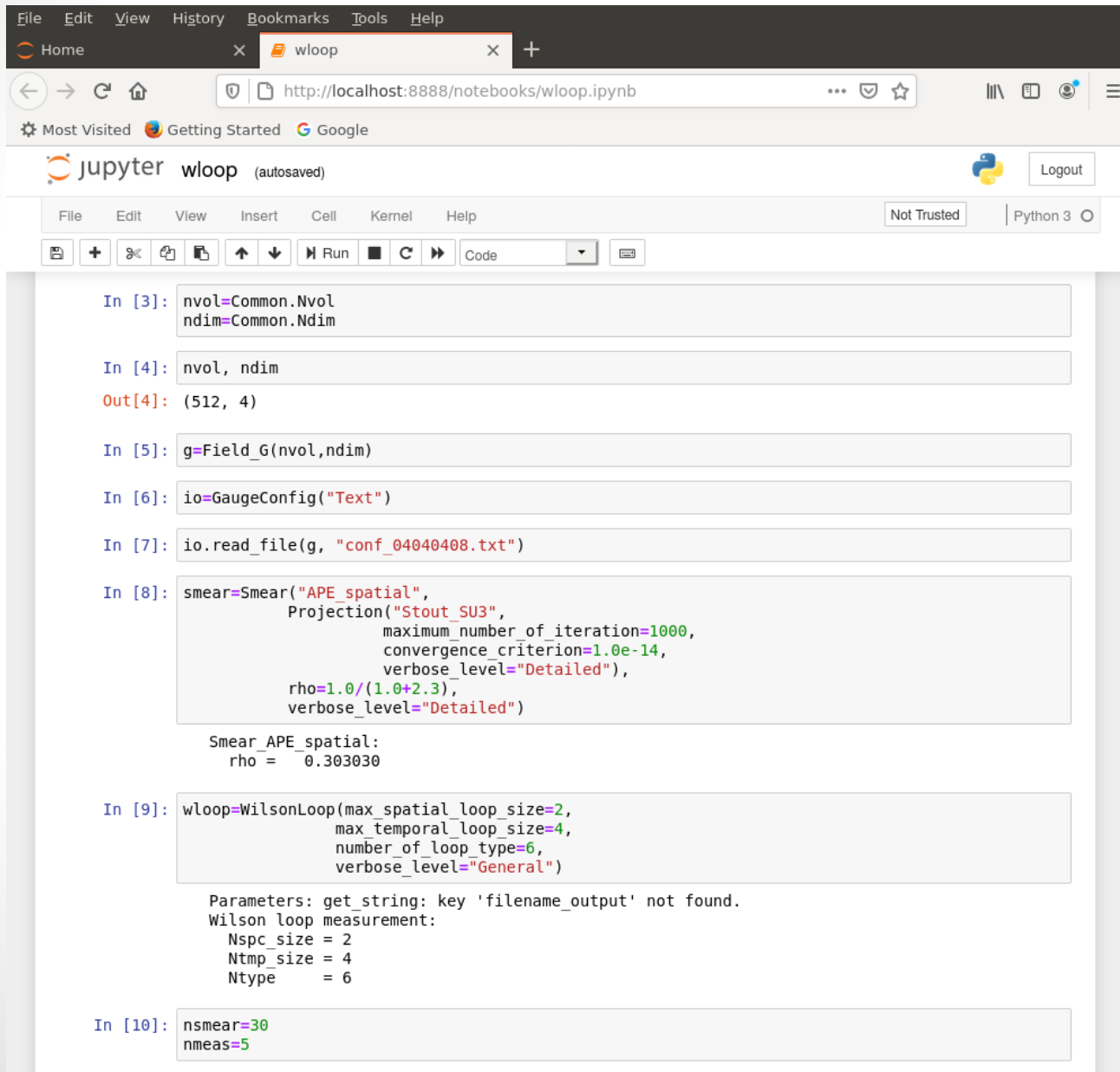


The screenshot shows a Jupyter Notebook interface in a web browser. The browser's address bar shows the URL `http://localhost:8888/notebooks/wloop.ipynb`. The notebook's title bar indicates it is named "wloop" and is "autosaved". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help) and a toolbar with icons for saving, running, and other actions. The code area contains four input cells:

```
In [1]: from pyb import *  
  
        Bridge++ 1.5.3  
  
        Timer: timestamp: 2020/10/16 14:46:32 +0900  
  
In [2]: setup([4,4,4,8], verbose_level="General")  
  
        Main: input parameters  
        lattice_size = [4, 4, 4, 8]  
        grid size    = [1, 1, 1, 1]  
        number of thread = 1  
        number of color = 3  
        logfile       = stdout  
        ildg_logfile   = stdout  
        vlevel        = General  
  
        Lattice parameters:  
        Lx = 4, Ly = 4, Lz = 4, Lt = 8,  
        Nx = 4, Ny = 4, Nz = 4, Nt = 8,  
        NPEx = 1, NPEy = 1, NPEz = 1, NPET = 1,  
        Lvol = 512, Nvol = 512, NPE = 1,  
        Ndim = 4  
        Nc = 3  
        Nd = 4  
  
        ThreadManager_OpenMP being setup.  
        Number of thread = 1  
        OpenMP is not used: stub implementation.  
        RandomNumbers_MT19937: seed = 12345671  
  
In [3]: nvol=Common.Nvol  
        ndim=Common.Ndim  
  
In [4]: nvol, ndim  
Out[4]: (512, 4)
```

Wilson Loop と呼ばれる
物理量の測定を Jupyter
Notebook 上で実行する
例

PyBridge++ on Jupyter notebook



The screenshot displays a Jupyter Notebook interface in a web browser. The browser's address bar shows the URL `http://localhost:8888/notebooks/wloop.ipynb`. The notebook's title bar indicates the file name is `wloop` and it has been autosaved. The interface includes a top menu bar with options like File, Edit, View, History, Bookmarks, Tools, and Help. Below this is a toolbar with icons for file operations and execution. The notebook content consists of several code cells, each with an input prompt (In [n]:) and an output (Out[n:]). The code cells contain Python code for initializing variables, creating a field, reading a configuration file, and running a Wilson loop measurement. The output of the measurement is displayed as a dictionary of parameters.

```
In [3]: nvol=Common.Nvol
        ndim=Common.Ndim

In [4]: nvol, ndim
Out[4]: (512, 4)

In [5]: g=Field_G(nvol,ndim)

In [6]: io=GaugeConfig("Text")

In [7]: io.read_file(g, "conf_04040408.txt")

In [8]: smear=Smear("APE_spatial",
                    Projection("Stout_SU3",
                                maximum_number_of_iteration=1000,
                                convergence_criterion=1.0e-14,
                                verbose_level="Detailed"),
                    rho=1.0/(1.0+2.3),
                    verbose_level="Detailed")

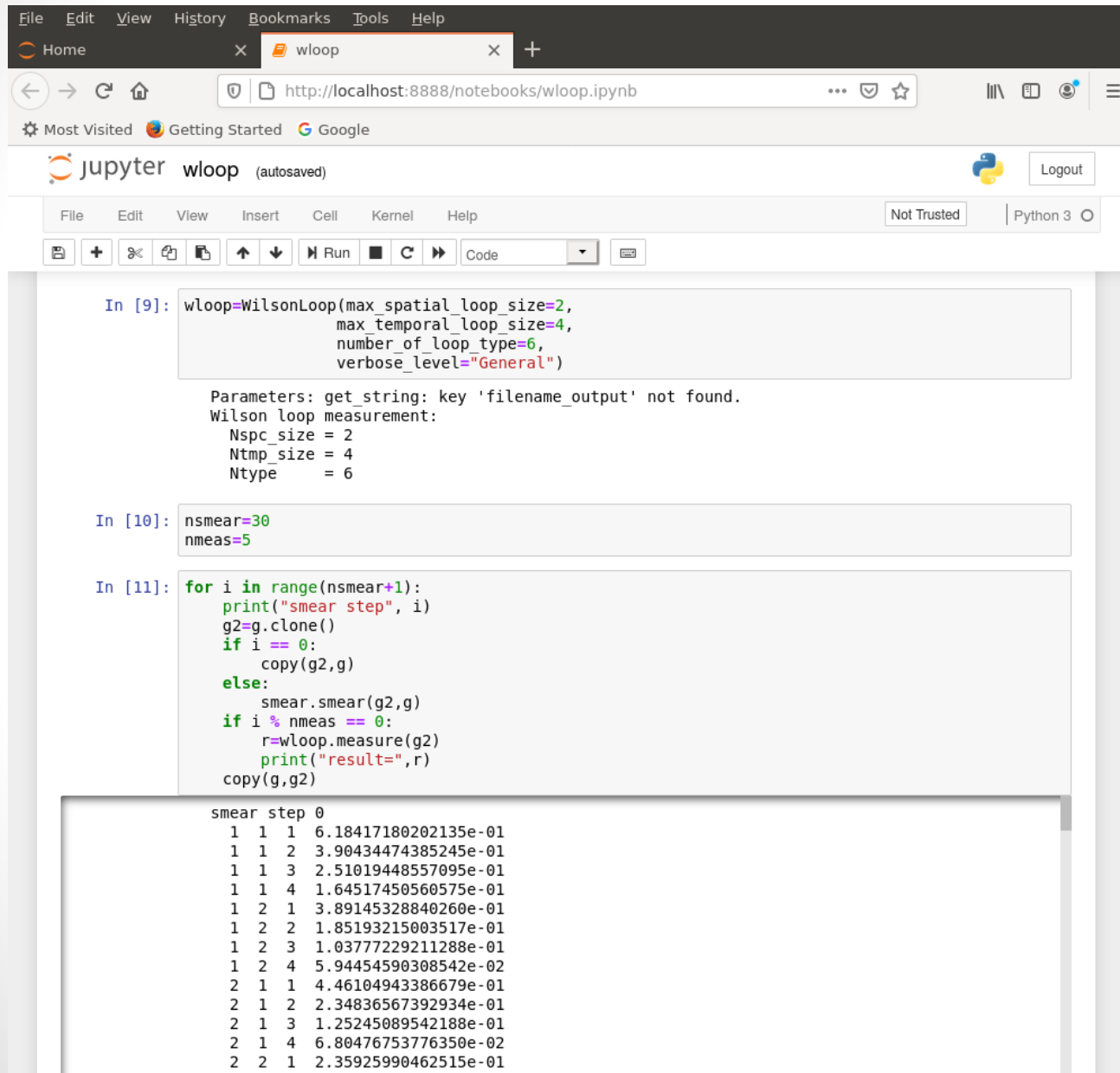
        Smear_APE_spatial:
        rho = 0.303030

In [9]: wloop=WilsonLoop(max_spatial_loop_size=2,
                          max_temporal_loop_size=4,
                          number_of_loop_type=6,
                          verbose_level="General")

        Parameters: get_string: key 'filename_output' not found.
        Wilson loop measurement:
        Nspc_size = 2
        Ntmp_size = 4
        Ntype     = 6

In [10]: nsmear=30
         nmeas=5
```

PyBridge++ on Jupyter notebook



The screenshot shows a Jupyter notebook titled 'wloop' running on a local server at `http://localhost:8888/notebooks/wloop.ipynb`. The interface includes a top menu bar with 'File', 'Edit', 'View', 'History', 'Bookmarks', 'Tools', and 'Help'. Below the menu is a toolbar with icons for file operations and a 'Run' button. The notebook content area displays three code cells. The first cell, labeled 'In [9]:', defines a `WilsonLoop` object with parameters `max_spatial_loop_size=2`, `max_temporal_loop_size=4`, `number_of_loop_type=6`, and `verbose_level="General"`. The output of this cell shows the parameters and the Wilson loop measurement results: `Nspc_size = 2`, `Ntmp_size = 4`, and `Ntype = 6`. The second cell, labeled 'In [10]:', defines `nsmeas=30` and `nmeas=5`. The third cell, labeled 'In [11]:', contains a `for` loop that iterates over `nsmeas+1` steps, printing the 'smear step' and the result of the Wilson loop measurement. The output of the third cell shows the results of the measurements, including the 'smear step' and the Wilson loop measurement results.

```
In [9]: wloop=WilsonLoop(max_spatial_loop_size=2,
                        max_temporal_loop_size=4,
                        number_of_loop_type=6,
                        verbose_level="General")

Parameters: get_string: key 'filename_output' not found.
Wilson loop measurement:
  Nspc_size = 2
  Ntmp_size = 4
  Ntype     = 6

In [10]: nsmeas=30
         nmeas=5

In [11]: for i in range(nsmeas+1):
         print("smear step", i)
         g2=g.clone()
         if i == 0:
             copy(g2,g)
         else:
             smear.smear(g2,g)
         if i % nmeas == 0:
             r=wloop.measure(g2)
             print("result=",r)
         copy(g,g2)

smear step 0
1 1 1 6.18417180202135e-01
1 1 2 3.90434474385245e-01
1 1 3 2.51019448557095e-01
1 1 4 1.64517450560575e-01
1 2 1 3.89145328840260e-01
1 2 2 1.85193215003517e-01
1 2 3 1.03777229211288e-01
1 2 4 5.94454590308542e-02
2 1 1 4.46104943386679e-01
2 1 2 2.34836567392934e-01
2 1 3 1.25245089542188e-01
2 1 4 6.80476753776350e-02
2 2 1 2.35925990462515e-01
```

PyBridge++

- Pythonは遅い?
- PyBridge++ は Bridge++ のクラスを wrap し、実際の計算は native な C++ コードが動く。

実行例: Clover Dirac演算子の線形方程式を解く

(格子サイズ $8^3 \times 16$)

native Bridge++	PyBridge++
78.85 sec	79.62 sec

- Performance への影響はわずか。

Tips

- ObjectのLifetime
 - ▶ C++: Objectの new ~ delete はユーザが管理
 - ▶ Python: 参照カウントと garbage collection
- 他のobjectを参照する場合に Lifetime に注意する必要がある

```
C++  Fopr *fop = new Fopr_Wilson;           //演算子 D
      Solver *solver = new Solver_CG(fop); // 線形方程式ソルバ
      solver->solve(x, b);                  //  $x = D^{-1} b$ 
```

```
Python fop = pyb.Fopr_Wilson()
        solver = pyb.Solver_CG(fop)
        solver.solve(x, b)                # fopの存在を保証する必要
```

→Solver が fop の参照カウントをインクリメントする

Tips

- 並列化
 - OpenMP スレッド並列
 - C/C++ライブラリがスレッド並列化されていればそのまま multithread で実行される。
 - (Pythonと並行処理する方法は?)
 - MPIノード並列
 - C/C++ライブラリがMPI化されていれば、`mpiexec python` で MPI並列で動作する。
 - PythonのMPI環境 `mpi4py` と相互利用できる。MPIの初期化はどちらかで行う。

Summary

- Python から C/C++ のライブラリを呼び出す extension を template library の pybind11 を用いて作成。
- 格子QCDコードセット Bridge++ の Python binding である PyBridge++ を紹介。
- Jupyter notebook の利用など、Python のエコシステムを活用した数値シミュレーション環境の構築。