Binding detailed example: Minuit2

Let's try a non-trivial example: Minuit2 (6.14.0 standalone edition)

Requirements

- Minuit2 6.14.0 standalone edition (included)
- Pybind11 (included)
- NumPy
- C++11 compatible compiler
- CMake 3

Expectations

- Be able to minimize a very simple function and get some parameters

Step 1: Get source

- Download Minuit2 source (provided in `minuit2src`)
- Install Pybind11 or add as submodule (provided in `pybind11`)

Step 2: Plan interface

- You should know what the C++ looks like, and know what you want the Python to look like
- For now, let's replicate the C++ experience

For example: a simple minimizer for $f(x) = x^2$ (should quickly find 0 as minimum):

- Define FCN
- Setup parameters
- Minimize
- Print result

Will use print out for illustration (instead of `MnPrint::SetLevel`)

In [1]:

```
%%writefile SimpleFCN.h
#pragma once

#include <Minuit2/FCNBase.h>
#include <Minuit2/FunctionMinimum.h>
#include <Minuit2/MnPrint.h>
#include <Minuit2/MnMigrad.h>

using namespace ROOT::Minuit2;

class SimpleFCN : public FCNBase {
    double Up() const override {return 0.5;}
    double operator()(const std::vector<double> &v) const override {
        std::cout << "val = " << v.at(0) << std::endl;
        return v.at(0)*v.at(0);
    }
};
```

```
Overwriting SimpleFCN.h
```

In [2]:

```
%%writefile simpleminuit.cpp
#include "SimpleFCN.h"

int main() {
    SimpleFCN fcn;
    MnUserParameters upar;
    upar.Add("x", 1., 0.1);
    MnMigrad migrad(fcn, upar);
    FunctionMinimum min = migrad();
    std::cout << min << std::endl;
}
```

```
Overwriting simpleminuit.cpp
```

In [3]:
```
%%writefile CMakeLists.txt

cmake_minimum_required(VERSION 3.4)
project(Minuit2SimpleExamle LANGUAGES CXX)

add_subdirectory(minuit2src)
add_executable(simpleminuit simpleminuit.cpp SimpleFCN.h)
target_link_libraries(simpleminuit PRIVATE Minuit2::Minuit2)
```

```
Overwriting CMakeLists.txt
```

Standard CMake configure and build (using Ninja instead of Make for speed)

In [4]:
```
!cmake -GNinja .
!cmake --build .
```

```
-- Configuring done
-- Generating done
-- Build files have been written to: /eos/user/h/hschrein/SWAN_projects/pybi
ndings_cc
[2/2] Linking CXX executable simpleminuitinuit.dir/simpleminuit.cpp.o
```

In [5]:
```
!./simpleminuit
```

```
val = 1
val = 1.001
val = 0.999
val = 1.0006
val = 0.999402
val = -8.23008e-11
val = 0.000345267
val = -0.000345267
val = -8.23008e-11
val = 0.000345267
val = -0.000345267
val = 6.90533e-05
val = -6.90535e-05

Minuit did successfully converge.
# of function calls: 13
minimum function Value: 6.773427082119e-21
minimum edm: 6.773427081817e-21
minimum internal state vector: LAVector parameters:
 -8.230083281546e-11

minimum internal covariance matrix: LASymMatrix parameters:
              1


# ext. ||   Name    ||   type  ||     Value     ||  Error +/-

   0   ||         x ||  free   || -8.230083281546e-11 ||0.7071067811865
```

Step 3: Bind parts we need

- subclassable FCNBase
- MnUserParameters (constructor and `Add(string, double, double)`)
- MnMigrad (constructor and operator())
- FunctionMinimum (cout)

Recommended structure of a Pybind11 program

main.cpp

- Builds module
- Avoids imports (fast compile)

```
include <pybind11/pybind11.h>
namespace py = pybind11;

void init_part1(py::module &);
void init_part2(py::module &);

PYBIND11_MODULE(mymodule, m) {
    m.doc() = "Real code would never have such poor documentation...";
    init_part1(m);
    init_part2(m);
}
```

In [6]:
```
mkdir -p pyminuit2
```

In [7]:
```
%%writefile pyminuit2/pyminuit2.cpp

#include <pybind11/pybind11.h>

namespace py = pybind11;

void init_FCNBase(py::module &);
void init_MnUserParameters(py::module &);
void init_MnMigrad(py::module &);
void init_FunctionMinimum(py::module &);

PYBIND11_MODULE(minuit2, m) {
    init_FCNBase(m);
    init_MnUserParameters(m);
    init_MnMigrad(m);
    init_FunctionMinimum(m);
}
```

```
Overwriting pyminuit2/pyminuit2.cpp
```

We will put all headers in a collective header (not a good idea unless you are trying to show files one per slide).

In [8]:
```
%%writefile pyminuit2/PyHeader.h
#pragma once
#include <pybind11/pybind11.h>
#include <pybind11/functional.h>
#include <pybind11/numpy.h>
#include <pybind11/stl.h>

#include <Minuit2/FCNBase.h>
#include <Minuit2/MnMigrad.h>
#include <Minuit2/MnApplication.h>
#include <Minuit2/MnUserParameters.h>
#include <Minuit2/FunctionMinimum.h>

namespace py = pybind11;
using namespace pybind11::literals;
using namespace ROOT::Minuit2;
```

```
Overwriting pyminuit2/PyHeader.h
```

Overloads

- Pure virtual methods cannot be instantiated in C++
- Have to provide "Trampoline class" to provide Python class

In [9]:
```
%%writefile pyminuit2/FCNBase.cpp
#include "PyHeader.h"
class PyFCNBase : public FCNBase {
   public:
     using FCNBase::FCNBase;

     double operator()(const std::vector<double> &v) const override {
         PYBIND11_OVERLOAD_PURE_NAME(
             double, FCNBase, "__call__", operator(), v);}

     double Up() const override {
         PYBIND11_OVERLOAD_PURE(double, FCNBase, Up, );}
 };
void init_FCNBase(py::module &m) {
    py::class_<FCNBase, PyFCNBase>(m, "FCNBase")
         .def(py::init<>())
         .def("__call__", &FCNBase::operator())
         .def("Up", &FCNBase::Up);
}
```

```
Overwriting pyminuit2/FCNBase.cpp
```

Overloaded function signatures:

- C++11 syntax: `(bool (MnUserParameters::*)(const std::string &, double)) &MnUserParameters::Add`
- C++14 syntax: `py::overload_cast<const std::string &, double> (&MnUserParameters::Add)`

In [10]:
```
%%writefile pyminuit2/MnUserParameters.cpp
#include "PyHeader.h"

void init_MnUserParameters(py::module &m) {
    py::class_<MnUserParameters>(m, "MnUserParameters")
        .def(py::init<>())
        .def("Add", (bool (MnUserParameters::*)(const std::string &, double)) &M
nUserParameters::Add)
        .def("Add", (bool (MnUserParameters::*)(const std::string &, double, dou
ble)) &MnUserParameters::Add)
    ;
}
```

```
Overwriting pyminuit2/MnUserParameters.cpp
```

Adding default arguments (and named arguments)

- Using " "_a literal, names and even defaults can be added

In [11]:
```
%%writefile pyminuit2/MnMigrad.cpp
#include "PyHeader.h"

void init_MnMigrad(py::module &m) {
    py::class_<MnApplication>(m, "MnApplication")
        .def("__call__",
             &MnApplication::operator(),
             "Minimize the function, returns a function minimum",
             "maxfcn"_a    = 0,
             "tolerance"_a = 0.1);

    py::class_<MnMigrad, MnApplication>(m, "MnMigrad")
        .def(py::init<const FCNBase &, const MnUserParameters &, unsigned int>()
,
             "fcn"_a, "par"_a, "stra"_a = 1)
    ;
}
```

```
Overwriting pyminuit2/MnMigrad.cpp
```

Lambda functions

- Pybind11 accepts lambda functions, as well

In [12]:
```
%%writefile pyminuit2/FunctionMinimum.cpp
#include "PyHeader.h"

#include <sstream>
#include <Minuit2/MnPrint.h>

void init_FunctionMinimum(py::module &m) {
    py::class_<FunctionMinimum>(m, "FunctionMinimum")
        .def("__str__", [](const FunctionMinimum &self) {
            std::stringstream os;
            os << self;
            return os.str();
        })
    ;
}
```

```
Overwriting pyminuit2/FunctionMinimum.cpp
```

In [13]:

```
%%writefile CMakeLists.txt
cmake_minimum_required(VERSION 3.4)
project(Minuit2SimpleExamle LANGUAGES CXX)

set(CMAKE_POSITION_INDEPENDENT_CODE ON)
add_subdirectory(minuit2src)
add_executable(simpleminuit simpleminuit.cpp SimpleFCN.h)
target_link_libraries(simpleminuit PRIVATE Minuit2::Minuit2)

add_subdirectory(pybind11)

file(GLOB OUTPUT pyminuit2/*.cpp)
pybind11_add_module(minuit2 ${OUTPUT})
target_link_libraries(minuit2 PUBLIC Minuit2::Minuit2)
```

```
Overwriting CMakeLists.txt
```

In [14]:
```
!cmake .
!cmake --build .
```

```
-- pybind11 v2.2.3
-- Configuring done
-- Generating done
-- Build files have been written to: /eos/user/h/hschrein/SWAN_projects/pybi
ndings_cc
[85/85] Linking CXX shared module minuit2.cpython-36m-x86_64-linux-gnu.so[Ko
```

Usage

We can now use our module! (Built in the current directory by CMake)

In [15]:
```
import sys
if '.' not in sys.path:
    sys.path.append('.')
```

In [16]:
```
import minuit2

class SimpleFCN (minuit2.FCNBase):
    def Up(self):
        return 0.5
    def __call__(self, v):
        print("val =", v[0])
        return v[0]**2;
```

In [17]:

```
fcn = SimpleFCN()
upar = minuit2.MnUserParameters()
upar.Add("x", 1., 0.1)
migrad = minuit2.MnMigrad(fcn, upar)
min = migrad()
```

```
val = 1.0
val = 1.001
val = 0.999
val = 1.0005980198587356
val = 0.9994019801412644
val = -8.230083281546285e-11
val = 0.00034526688527999595
val = -0.00034526704988816616
val = -8.230083281546285e-11
val = 0.00034526688527999595
val = -0.00034526704988816616
val = 6.905331121533294e-05
val = -6.905347581699857e-05
```

In [18]:
```
print(min)
```

```
Minuit did successfully converge.
# of function calls: 13
minimum function Value: 6.773427082119e-21
minimum edm: 6.773427081817e-21
minimum internal state vector: LAVector parameters:
 -8.230083281546e-11

minimum internal covariance matrix: LASymMatrix parameters:
              1


# ext. ||   Name    ||   type  ||     Value     ||  Error +/-

   0   ||         x ||  free   || -8.230083281546e-11 ||0.7071067811865
```

Done

- See GooFit's built in Minuit2 bindings (https://github.com/GooFit/GooFit/tree/master/python/Minuit2) for a more complete example
- Pybind11 bindings can talk to each other at the C level!

Overall topics covered:

- ctypes, CFFI : Pure Python, C only
- CPython: How all bindings work
- SWIG: Multi-language, automatic
- Cython: New language
- Pybind11: Pure C++11
- CPPYY: From ROOT's JIT engine
- An advanced binding in Pybind11

Backup:

This is the setup.py file for the Miniut2 bindings. With this, you can use the standard Python tools to build! (but slower and more verbose than CMake)

In [19]:
```
%%writefile setup.py

from setuptools import setup, Extension
from setuptools.command.build_ext import build_ext
import sys
import setuptools
from pathlib import Path # Python 3 or Python 2 backport: pathlib2
import pybind11 # Real code should defer this import
```

```
sources = set(str(p) for p in Path('Minuit2-6.14.0-Source/src').glob('**/*.cxx')
)
sources.remove('Minuit2-6.14.0-Source/src/TMinuit2TraceObject.cxx')

## Add your sources to `sources`
sources |= set(str(p) for p in Path('pyminuit2').glob('*.cpp'))

ext_modules = [
    Extension(
        'minuit2',
        list(sources),
        include_dirs=[
            pybind11.get_include(False),
            pybind11.get_include(True),
            'Minuit2-6.14.0-Source/inc',
        ],
        language='c++',
        define_macros=[('WARNINGMSG', None),
                       ('MATH_NO_PLUGIN_MANAGER', None),
                       ('ROOT_Math_VecTypes', None)
                      ],
    ),
]

class BuildExt(build_ext):
    """A custom build extension for adding compiler-specific options."""
    c_opts = {
        'msvc': ['/EHsc'],
        'unix': [],
    }

    if sys.platform == 'darwin':
        c_opts['unix'] += ['-stdlib=libc++', '-mmacosx-version-min=10.7']

    def build_extensions(self):
        ct = self.compiler.compiler_type
        opts = self.c_opts.get(ct, [])
        if ct == 'unix':
```

```
            opts.append('-DVERSION_INFO="%s"' % self.distribution.get_version())
            opts.append('-std=c++14')
            opts.append('-fvisibility=hidden')
        elif ct == 'msvc':
            opts.append('/DVERSION_INFO=\\"%s\\"' % self.distribution.get_versio
n())
        for ext in self.extensions:
            ext.extra_compile_args = opts
        build_ext.build_extensions(self)

setup(
    name='minuit2',
    version='6.14.0',
    author='Henry Schriener',
    author_email='hschrein@cern.ch',
    url='https://github.com/GooFit/Minuit2',
    description='A Pybind11 Minuit2 binding',
    long_description='',
    ext_modules=ext_modules,
    install_requires=['pybind11>=2.2', 'numpy>=1.10'],
    cmdclass={'build_ext': BuildExt},
    zip_safe=False,
)
```

```
Overwriting setup.py
```

In [20]:
```
#!python setup.py build_ext
```