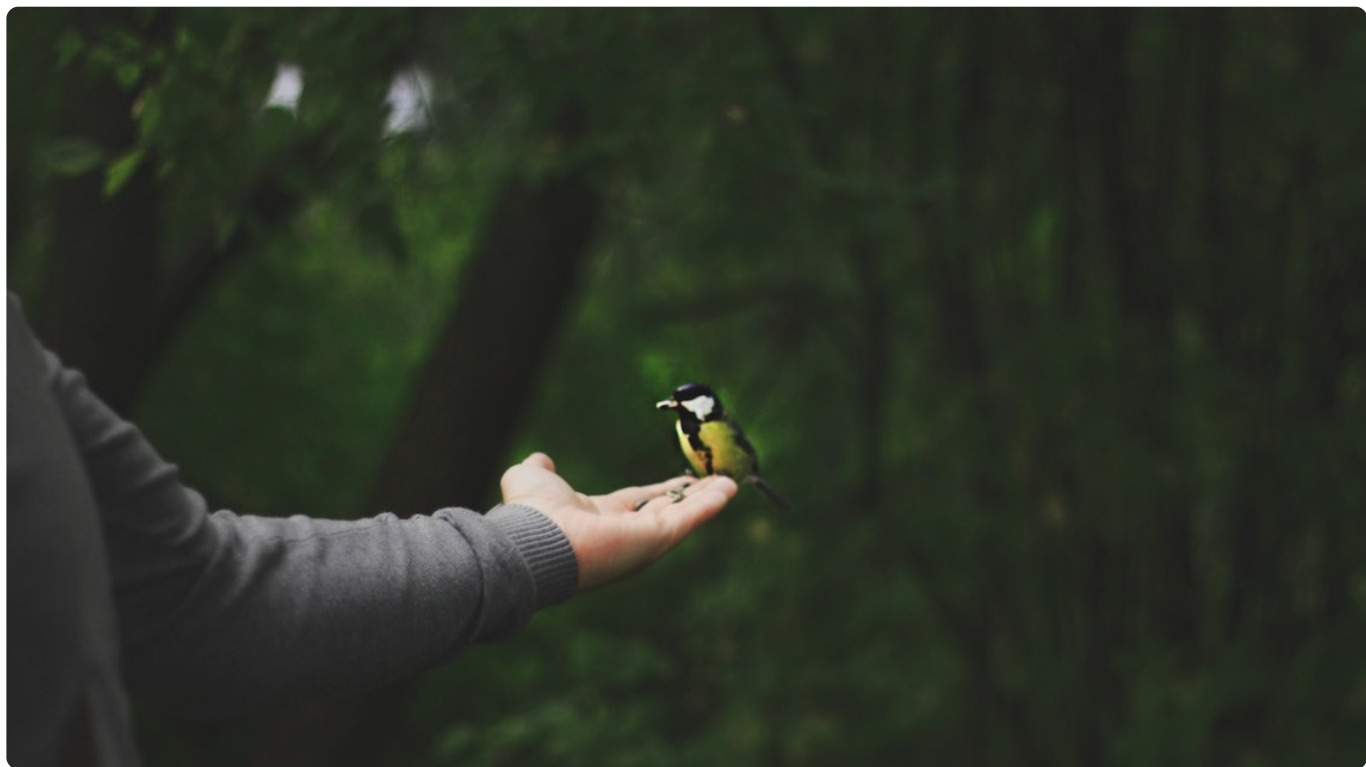


加餐 | 带你上手SWIG：一份清晰好用的SWIG编程实践指南

2019-08-16 卢誉声

Python核心技术与实战

[进入课程 >](#)



讲述：冯永吉

时长 14:16 大小 13.07M



你好，我是卢誉声，Autodesk 数据平台和计算平台资深软件工程师，也是《移动平台深度神经网络实战》和《分布式实时处理系统：原理架构与实现》的作者，主要从事 C/C++、JavaScript 开发工作和平台架构方面的研发工作，对 SWIG 也有比较深的研究。很高兴受极客时间邀请来做本次分享，今天，我们就来聊一聊 SWIG 这个话题。

我们都知道，Python 是一门易于上手并实验友好的胶水语言。现在有很多机器学习开发或研究人员，都选择 Python 作为主力编程语言；流行的机器学习框架也都会提供 Python 语言的支持作为调用接口和工具。因此，相较于学习成本更高的 C++ 来说，把 Python 作为进入机器学习世界的首选编程语言，就再合适不过了。

不过，像 TensorFlow 或 PyTorch 这样的机器学习框架的核心，是使用 Python 编写的吗？

显然不是。这里面的原因比较多，但最为显著的一个原因就是“性能”。通过 C++ 编写的机器学习框架内核，加上编译器的优化能力，为系统提供了接近于机器码执行的效率。这种得天独厚的优势，让 C++ 在机器学习的核心领域站稳了脚跟。我们前面所说的 TensorFlow 和 PyTorch 的核心，便都是使用 C/C++ 开发的。其中，TensorFlow 的内核，就是由高度优化的 C++ 代码和 CUDA 编写而成。

因此，我们可以理解为，TensorFlow 通过 Python 来描述模型，而实际的运算则是由高性能 C++ 代码执行的。而且，在绝大多数情况下，不同操作之间传递的数据，并不会拷贝回 Python 代码的执行空间。机器学习框架，正是通过这样的方式确保了计算性能，同时兼顾了对框架易用性方面的考虑。

因此，当 Python 和 C++ 结合使用的时候，Python 本身的性能瓶颈就不那么重要了。它足够胜任我们给它的任务就可以了，至于对计算有更高要求的任务，就交给 C++ 来做吧！

今天，我们就来讨论下，如何通过 SWIG 对 C++ 程序进行 Python 封装。我会先带你编写一段 Python 脚本，来执行一个简单的机器学习任务；接着，尝试将计算密集的部分改写成 C++ 程序，再通过 SWIG 对其进行封装。最后的结果就是，Python 把计算密集的任务委托给 C++ 执行。

我们会对性能做一个简单比较，并在这个过程中，讲解使用 SWIG 的方法。同时，在今天这节课的最后，我会为你提供一个学习路径，作为日后提高的参考。

明确了今天的学习目的，也就是使用 SWIG 来实现 Python 对 C++ 代码的调用，那么，我们今天的内容，其实可以看成一份**关于 SWIG 的编程实践指南**。学习这份指南之前，我们先来简单了解一下 SWIG。

SWIG 是什么？

SWIG，是一款能够连接 C/C++ 与多种高级编程语言（我们在这里特别强调 Python）的软件开发工具。SWIG 支持多种不同类型的目标语言，这其中，支持的常见脚本语言包括 JavaScript、Perl、PHP、Tcl、Ruby 和 Python 等，支持的高级编程语言则包括 C#、D、Go 语言、Java（包括对 Android 的支持）、Lua、OCaml、Octave、Scilab 和 R。

我们通常使用 SWIG 来创建高级解释或编译型的编程环境和接口，它也常被用来当作 C/C++ 编写原型的测试工具。一个典型的应用场景，便是解析和创建 C/C++ 接口，生成

胶水代码供像 Python 这样的高级编程语言调用。近期发布的 4.0.0 版本，更是带来了 C++ 的显著改进和支持，这其中包括（不局限于）下面几点。

针对 C#、Java 和 Ruby 而改进的 STL 包装器。

针对 Java、Python 和 Ruby，增加 C++11 标准下的 STL 容器的支持。

改进了对 C++11 和 C++14 代码的支持。

修正了 C++ 中对智能指针 `shared_ptr` 的一系列 bug 修复。

一系列针对 C 预处理器的极端 case 修复。

一系列针对成员函数指针问题的修复。


低支持的 Python 版本为 2.7、3.2-3.7。

使用 Python 实现 PCA 算法

借助于 SWIG，我们可以简单地实现用 Python 调用 C/C++ 库，甚至可以用 Python 继承和使用 C++ 类。接下来，我们先来看一个你十分熟悉的使用 Python 编写的 PCA (Principal Component Analysis，主成分分析) 算法。

因为我们今天的目标不是讲解 PCA 算法，所以如果你对这个算法还不是很熟悉，也没有关系，我会直接给出具体的代码，我们把焦点放在如何使用 SWIG 上就可以了。下面，我先给出代码清单 1。

代码清单 1，基于 Python 编写的 PCA 算法 `testPCAPurePython.py`：

 复制代码


```
1 import numpy as np
2
3 def compute_pca(data):
4     m = np.mean(data, axis=0)
5     datac = np.array([obs - m for obs in data])
6     T = np.dot(datac, datac.T)
7     [u,s,v] = np.linalg.svd(T)
8
9     pcs = [np.dot(datac.T, item) for item in u.T ]
10
11     pcs = np.array([d / np.linalg.norm(d) for d in pcs])
12
13     return pcs, m, s, T, u
14
```

```

15 def compute_projections(I, pcs, m):
16     projections = []
17     for i in I:
18         w = []
19         for p in pcs:
20             w.append(np.dot(i - m, p))
21         projections.append(w)
22     return projections
23
24 def reconstruct(w, X, m, dim = 5):
25     return np.dot(w[:dim], X[:dim,:]) + m
26
27 def normalize(samples, maxs = None):
28     if not maxs:
29         maxs = np.max(samples)
30     return np.array([np.ravel(s) / maxs for s in samples])

```

现在，我们保存这段编写好的代码，并通过下面的命令来执行：


 复制代码

```
1 python3 testPCAPurePython.py
```

准备 SWIG

这样，我们已经获得了一些进展——使用 Python 编写了一个 PCA 算法，并得到了一些结果。接下来，我们看一下如何开始 SWIG 的开发工作。我会先从编译相关组件开始，再介绍一个简单使用的例子，为后续内容做准备。

首先，我们从 SWIG 的网站 (<http://swig.org/download.html>) 下载源代码包，并开始构建：

 复制代码


```

1 $ wget https://newcontinuum.dl.sourceforge.net/project/swig/swig/swig-4.0.0/swig-4.0.0.1
2 $ tar -xvf swig-4.0.0.tar.gz
3 $ cd swig-4.0.0
4 $ wget https://ftp.pcre.org/pub/pcre/pcre-8.43.tar.gz # SWIG 需要依赖 pcre 工作
5 $ sh ./Tools/pcre-build.sh # 该脚本会将 pcre 自动构建成 SWIG 使用的静态库
6 $ ./configure # 注意需要安装 bison，如果没有安装需要读者手动安装
7 $ make
8 $ sudo make install

```

一切就绪后，我们就来编写一个简单的例子吧。这个例子同样来源于 SWIG 网站（<http://swig.org/tutorial.html>）。我们先来创建一个简单的 c 文件，你可以通过你习惯使用的文本编辑器（比如 vi），创建一个名为 `example.c` 的文件，并编写代码。代码内容我放在了代码清单 2 中。


代码清单 2，`example.c`：

 复制代码

```
1 #include <time.h>
2 double My_variable = 3.0;
3
4 int fact(int n) {
5     if (n <= 1) return 1;
6     else return n*fact(n-1);
7 }
8
9 int my_mod(int x, int y) {
10     return (x%y);
11 }
12
13 char *get_time()
14 {
15     time_t ltime;
16     time(&ltime);
17     return ctime(&ltime);
18 }
```

接下来，我们编写一个名为 `example.i` 的接口定义文件，和稍后用作测试的 Python 脚本，内容如代码清单 3 和代码清单 4 所示。

代码清单 3，`example.i`：


 复制代码

```
1 %module example
2 %{
3 /* Put header files here or function declarations like below */
4 extern double My_variable;
5 extern int fact(int n);
```

```
6 extern int my_mod(int x, int y);
7 extern char *get_time();
8 %}
9
10 extern double My_variable;
11 extern int fact(int n);
12 extern int my_mod(int x, int y);
13 extern char *get_time();
```


我来解释下清单 3 这段代码。第 1 行，我们定义了模块的名称为 `example`。第 2-8 行，我们直接指定了 `example.c` 中的函数定义，也可以定义一个 `example.h` 头文件，并将这些定义加入其中；然后，在 `%{ ... %}` 结构体中包含 `example.h`，来实现相同的功能。第 10-13 行，则是定义了导出的接口，以便你在 Python 中直接调用这些接口。

代码清单 4，`testExample.py`：

 复制代码

```
1 import example
2 print(example.fact(5))
3 print(example.my_mod(7,3))
4 print(example.get_time())
```

好了，到现在为止，我们已经准备就绪了。现在，我们来执行下面的代码，创建目标文件和最后链接的文件吧：

 复制代码

```
1 swig -python example.i
2 gcc -c -fPIC example.c example_wrap.c -I/usr/include/python3.6
3 gcc -shared example.o example_wrap.o -o _example.so
4 python3 testExample.py # 测试调用
```

其实，从代码清单 4 中你也能够看到，通过导入 `example`，我们可以直接在 Python 脚本中，调用使用 C 实现的函数接口，并获得返回值。


通过 SWIG 封装基于 C++ 编写的 Python 模块

到这一步，我们已经准备好了一份使用 C++ 编写的 PCA 算法，接下来，我们就要对其进行一个简单的封装。由于 C++ 缺少线性代数的官方支持，因此，为了简化线性代数运算，我这里用了一个第三方库 Armadillo。在 Ubuntu 下，它可以使用 `apt-get install libarmadillo-dev` 安装支持。

另外，还是要再三说明一下，我们今天这节课的重点并不是讲解 PCA 算法本身，所以希望你不要困于此处，而错过了真正的使用方法。当然，为了完整性考虑，我还是会对代码做出最基本的解释。

封装正式开始。我们先来编写一个名为 `pca.h` 的头文件定义，内容我放在了代码清单 5 中。

代码清单 5，`pca.h`：

 复制代码

```
1 #pragma once
2
3 #include <vector>
4 #include <string>
5 #include <armadillo>
6
7 class pca {
8 public:
9     pca();
10    explicit pca(long num_vars);
11    virtual ~pca();
12
13    bool operator==(const pca& other);
14
15    void set_num_variables(long num_vars);
16    long get_num_variables() const;
17    void add_record(const std::vector<double>& record);
18    std::vector<double> get_record(long record_index) const;
19    long get_num_records() const;
20    void set_do_normalize(bool do_normalize);
21    bool get_do_normalize() const;
22    void set_solver(const std::string& solver);
23    std::string get_solver() const;
24
25    void solve();
26
27    double check_eigenvectors_orthogonal() const;
28    double check_projection_accurate() const;
29
```

```

30 void save(const std::string& basename) const;
31 void load(const std::string& basename);
32
33 void set_num_retained(long num_retained);
34 long get_num_retained() const;
35 std::vector<double> to_principal_space(const std::vector<double>& record) const;
36 std::vector<double> to_variable_space(const std::vector<double>& data) const;
37 double get_energy() const;
38 double get_eigenvalue(long eigen_index) const;
39 std::vector<double> get_eigenvalues() const;
40 std::vector<double> get_eigenvector(long eigen_index) const;
41 std::vector<double> get_principal(long eigen_index) const;
42 std::vector<double> get_mean_values() const;
43 std::vector<double> get_sigma_values() const;
44
45 protected:
46     long num_vars_;
47     long num_records_;
48     long record_buffer_;
49     std::string solver_;
50     bool do_normalize_;
51     long num_retained_;
52     arma::Mat<double> data_;
53     arma::Col<double> energy_;
54     arma::Col<double> eigval_;
55     arma::Mat<double> eigvec_;
56     arma::Mat<double> proj_eigvec_;
57     arma::Mat<double> princomp_;
58     arma::Col<double> mean_;
59     arma::Col<double> sigma_;
60     void initialize_();
61     void assert_num_vars_();
62     void resize_data_if_needed_();
63 };

```

接着，我们再来编写具体实现 `pca.cpp`，也就是代码清单 6 的内容。

代码清单 6，`pca.cpp`：

 复制代码

```

1 #include "pca.h"
2 #include "utils.h"
3 #include <stdexcept>
4 #include <random>
5
6 pca::pca()

```



```

7      : num_vars_(0),
8      num_records_(0),
9      record_buffer_(1000),
10     solver_("dc"),
11     do_normalize_(false),
12     num_retained_(1),
13     energy_(1)
14 {}
15
16 pca::pca(long num_vars)
17     : num_vars_(num_vars),
18     num_records_(0),
19     record_buffer_(1000),
20     solver_("dc"),
21     do_normalize_(false),
22     num_retained_(num_vars_),
23     data_(record_buffer_, num_vars_),
24     energy_(1),
25     eigval_(num_vars_),
26     eigvec_(num_vars_, num_vars_),
27     proj_eigvec_(num_vars_, num_vars_),
28     princomp_(record_buffer_, num_vars_),
29     mean_(num_vars_),
30     sigma_(num_vars_)
31 {
32     assert_num_vars_();
33     initialize_();
34 }
35
36 pca::~pca()
37 {}
38
39 bool pca::operator==(const pca& other) {
40     const double eps = 1e-5;
41     if (num_vars_ == other.num_vars_ &&
42         num_records_ == other.num_records_ &&
43         record_buffer_ == other.record_buffer_ &&
44         solver_ == other.solver_ &&
45         do_normalize_ == other.do_normalize_ &&
46         num_retained_ == other.num_retained_ &&
47         utils::is_approx_equal_container(eigval_, other.eigval_, eps) &&
48         utils::is_approx_equal_container(eigvec_, other.eigvec_, eps) &&
49         utils::is_approx_equal_container(princomp_, other.princomp_, eps) &&
50         utils::is_approx_equal_container(energy_, other.energy_, eps) &&
51         utils::is_approx_equal_container(mean_, other.mean_, eps) &&
52         utils::is_approx_equal_container(sigma_, other.sigma_, eps) &&
53         utils::is_approx_equal_container(proj_eigvec_, other.proj_eigvec_, eps))
54         return true;
55     else
56         return false;
57 }
58

```

```

59 void pca::resize_data_if_needed_() {
60     if (num_records_ == record_buffer_) {
61         record_buffer_ += record_buffer_;
62         data_.resize(record_buffer_, num_vars_);
63     }
64 }
65
66 void pca::assert_num_vars_() {
67     if (num_vars_ < 2)
68         throw std::invalid_argument("Number of variables smaller than two.");
69 }
70
71 void pca::initialize_() {
72     data_.zeros();
73     eigval_.zeros();
74     eigvec_.zeros();
75     princomp_.zeros();
76     mean_.zeros();
77     sigma_.zeros();
78     energy_.zeros();
79 }
80
81 void pca::set_num_variables(long num_vars) {
82     num_vars_ = num_vars;
83     assert_num_vars_();
84     num_retained_ = num_vars_;
85     data_.resize(record_buffer_, num_vars_);
86     eigval_.resize(num_vars_);
87     eigvec_.resize(num_vars_, num_vars_);
88     mean_.resize(num_vars_);
89     sigma_.resize(num_vars_);
90     initialize_();
91 }
92
93 void pca::add_record(const std::vector<double>& record) {
94     assert_num_vars_();
95
96     if (num_vars_ != long(record.size()))
97         throw std::domain_error(utils::join("Record has the wrong size: ", record.size(
98
99     resize_data_if_needed_();
100     arma::Row<double> row(&record.front(), record.size());
101     data_.row(num_records_) = std::move(row);
102     ++num_records_;
103 }
104
105 std::vector<double> pca::get_record(long record_index) const {
106     return std::move(utils::extract_row_vector(data_, record_index));
107 }
108
109 void pca::set_do_normalize(bool do_normalize) {
110     do_normalize_ = do_normalize;

```

```

111 }
112
113 void pca::set_solver(const std::string& solver) {
114     if (solver!="standard" && solver!="dc")
115         throw std::invalid_argument(utils::join("No such solver available: ", solver));
116     solver_ = solver;
117 }
118
119 void pca::solve() {
120     assert_num_vars_();
121
122     if (num_records_ < 2)
123         throw std::logic_error("Number of records smaller than two.");
124
125     data_.resize(num_records_, num_vars_);
126
127     mean_ = utils::compute_column_means(data_);
128     utils::remove_column_means(data_, mean_);
129
130     sigma_ = utils::compute_column_rms(data_);
131     if (do_normalize_) utils::normalize_by_column(data_, sigma_);
132
133     arma::Col<double> eigval(num_vars_);
134     arma::Mat<double> eigvec(num_vars_, num_vars_);
135
136     arma::Mat<double> cov_mat = utils::make_covariance_matrix(data_);
137     arma::eig_sym(eigval, eigvec, cov_mat, solver_.c_str());
138     arma::uvec indices = arma::sort_index(eigval, 1);
139
140     for (long i=0; i<num_vars_; ++i) {
141         eigval_(i) = eigval(indices(i));
142         eigvec_.col(i) = eigvec.col(indices(i));
143     }
144
145     utils::enforce_positive_sign_by_column(eigvec_);
146     proj_eigvec_ = eigvec_;
147
148     princomp_ = data_ * eigvec_;
149
150     energy_(0) = arma::sum(eigval_);
151     eigval_ *= 1./energy_(0);
152 }
153
154 void pca::set_num_retained(long num_retained) {
155     if (num_retained<=0 || num_retained>num_vars_)
156         throw std::range_error(utils::join("Value out of range: ", num_retained));
157
158     num_retained_ = num_retained;
159     proj_eigvec_ = eigvec_.submat(0, 0, eigvec_.n_rows-1, num_retained_-1);
160 }
161
162 std::vector<double> pca::to_principal_space(const std::vector<double>& data) const {

```

```

163     arma::Col<double> column(&data.front(), data.size());
164     column -= mean_;
165     if (do_normalize_) column /= sigma_;
166     const arma::Row<double> row(column.t() * proj_eigvec_);
167     return std::move(utils::extract_row_vector(row, 0));
168 }
169
170 std::vector<double> pca::to_variable_space(const std::vector<double>& data) const {
171     const arma::Row<double> row(&data.front(), data.size());
172     arma::Col<double> column(arma::trans(row * proj_eigvec_.t()));
173     if (do_normalize_) column %= sigma_;
174     column += mean_;
175     return std::move(utils::extract_column_vector(column, 0));
176 }
177
178 double pca::get_energy() const {
179     return energy_(0);
180 }
181
182 double pca::get_eigenvalue(long eigen_index) const {
183     if (eigen_index >= num_vars_)
184         throw std::range_error(utils::join("Index out of range: ", eigen_index));
185     return eigval_(eigen_index);
186 }
187
188 std::vector<double> pca::get_eigenvalues() const {
189     return std::move(utils::extract_column_vector(eigval_, 0));
190 }
191
192 std::vector<double> pca::get_eigenvector(long eigen_index) const {
193     return std::move(utils::extract_column_vector(eigvec_, eigen_index));
194 }
195
196 std::vector<double> pca::get_principal(long eigen_index) const {
197     return std::move(utils::extract_column_vector(princomp_, eigen_index));
198 }
199
200 double pca::check_eigenvectors_orthogonal() const {
201     return std::abs(arma::det(eigvec_));
202 }
203
204 double pca::check_projection_accurate() const {
205     if (data_.n_cols != eigvec_.n_cols || data_.n_rows != princomp_.n_rows)
206         throw std::runtime_error("No proper data matrix present that the projection col
207     const arma::Mat<double> diff = (princomp_ * arma::trans(eigvec_)) - data_;
208     return 1 - arma::sum(arma::sum( arma::abs(diff) )) / diff.n_elem;
209 }
210
211 bool pca::get_do_normalize() const {
212     return do_normalize_;
213 }
214

```

```

215 std::string pca::get_solver() const {
216     return solver_;
217 }
218
219 std::vector<double> pca::get_mean_values() const {
220     return std::move(utils::extract_column_vector(mean_, 0));
221 }
222
223 std::vector<double> pca::get_sigma_values() const {
224     return std::move(utils::extract_column_vector(sigma_, 0));
225 }
226
227 long pca::get_num_variables() const {
228     return num_vars_;
229 }
230
231 long pca::get_num_records() const {
232     return num_records_;
233 }
234
235 long pca::get_num_retained() const {
236     return num_retained_;
237 }
238
239 void pca::save(const std::string& basename) const {
240     const std::string filename = basename + ".pca";
241     std::ofstream file(filename.c_str());
242     utils::assert_file_good(file.good(), filename);
243     utils::write_property(file, "num_variables", num_vars_);
244     utils::write_property(file, "num_records", num_records_);
245     utils::write_property(file, "solver", solver_);
246     utils::write_property(file, "num_retained", num_retained_);
247     utils::write_property(file, "do_normalize", do_normalize_);
248     file.close();
249
250     utils::write_matrix_object(basename + ".eigval", eigval_);
251     utils::write_matrix_object(basename + ".eigvec", eigvec_);
252     utils::write_matrix_object(basename + ".princomp", princomp_);
253     utils::write_matrix_object(basename + ".energy", energy_);
254     utils::write_matrix_object(basename + ".mean", mean_);
255     utils::write_matrix_object(basename + ".sigma", sigma_);
256 }
257
258 void pca::load(const std::string& basename) {
259     const std::string filename = basename + ".pca";
260     std::ifstream file(filename.c_str());
261     utils::assert_file_good(file.good(), filename);
262     utils::read_property(file, "num_variables", num_vars_);
263     utils::read_property(file, "num_records", num_records_);
264     utils::read_property(file, "solver", solver_);
265     utils::read_property(file, "num_retained", num_retained_);
266     utils::read_property(file, "do_normalize", do_normalize_);

```

```

267     file.close();
268
269     utils::read_matrix_object(basename + ".eigval", eigval_);
270     utils::read_matrix_object(basename + ".eigvec", eigvec_);
271     utils::read_matrix_object(basename + ".princomp", princomp_);
272     utils::read_matrix_object(basename + ".energy", energy_);
273     utils::read_matrix_object(basename + ".mean", mean_);
274     utils::read_matrix_object(basename + ".sigma", sigma_);
275
276     set_num_retained(num_retained_);
277 }

```

这里要注意了，代码清单 6 中用到了 `utils.h` 这个文件，它是对部分矩阵和数学计算的封装，内容我放在了代码清单 7 中。

代码清单 7，`utils.h`：

 复制代码

```

1  #pragma once
2
3  #include <armadillo>
4  #include <sstream>
5
6  namespace utils {
7      arma::Mat<double> make_covariance_matrix(const arma::Mat<double>& data);
8      arma::Mat<double> make_shuffled_matrix(const arma::Mat<double>& data);
9      arma::Col<double> compute_column_means(const arma::Mat<double>& data);
10     void remove_column_means(arma::Mat<double>& data, const arma::Col<double>& means);
11     arma::Col<double> compute_column_rms(const arma::Mat<double>& data);
12     void normalize_by_column(arma::Mat<double>& data, const arma::Col<double>& rms);
13     void enforce_positive_sign_by_column(arma::Mat<double>& data);
14     std::vector<double> extract_column_vector(const arma::Mat<double>& data, long index);
15     std::vector<double> extract_row_vector(const arma::Mat<double>& data, long index);
16     void assert_file_good(const bool& is_file_good, const std::string& filename);
17     template<typename T>
18     void write_matrix_object(const std::string& filename, const T& matrix) {
19         assert_file_good(matrix.quiet_save(filename, arma::arma_ascii), filename);
20     }
21
22     template<typename T>
23     void read_matrix_object(const std::string& filename, T& matrix) {
24         assert_file_good(matrix.quiet_load(filename), filename);
25     }
26     template<typename T, typename U, typename V>
27     bool is_approx_equal(const T& value1, const U& value2, const V& eps) {
28         return std::abs(value1-value2)<eps ? true : false;

```

```

29 }
30 template<typename T, typename U, typename V>
31 bool is_approx_equal_container(const T& container1, const U& container2, const V& eps) {
32     if (container1.size()==container2.size()) {
33         bool equal = true;
34         for (size_t i=0; i<container1.size(); ++i) {
35             equal = is_approx_equal(container1[i], container2[i], eps);
36             if (!equal) break;
37         }
38         return equal;
39     } else {
40         return false;
41     }
42 }
43 double get_mean(const std::vector<double>& iter);
44 double get_sigma(const std::vector<double>& iter);
45
46 struct join_helper {
47     static void add_to_stream(std::ostream& stream) {}
48
49     template<typename T, typename... Args>
50     static void add_to_stream(std::ostream& stream, const T& arg, const Args&... args) {
51         stream << arg;
52         add_to_stream(stream, args...);
53     }
54 };
55
56 template<typename T, typename... Args>
57 std::string join(const T& arg, const Args&... args) {
58     std::ostringstream stream;
59     stream << arg;
60     join_helper::add_to_stream(stream, args...);
61     return stream.str();
62 }
63
64 template<typename T>
65 void write_property(std::ostream& file, const std::string& key, const T& value) {
66     file << key << "\t" << value << std::endl;
67 }
68
69 template<typename T>
70 void read_property(std::istream& file, const std::string& key, T& value) {
71     std::string tmp;
72     bool found = false;
73     while (file.good()) {
74         file >> tmp;
75         if (tmp==key) {
76             file >> value;
77             found = true;
78             break;
79         }
80     }

```


```

81     if (!found)
82         throw std::domain_error(join("No such key available: ", key));
83     file.seekg(0);
84 }
85
86 } //utils

```

至于具体的实现代码，我放在了在代码清单 8utils.cpp中。

代码清单 8，utils.cpp：

 复制代码

```

1  #include "utils.h"
2  #include <stdexcept>
3  #include <sstream>
4  #include <numeric>
5
6  namespace utils {
7
8  arma::Mat<double> make_covariance_matrix(const arma::Mat<double>& data) {
9      return std::move( (data.t()*data) * (1./(data.n_rows-1)) );
10 }
11
12 arma::Mat<double> make_shuffled_matrix(const arma::Mat<double>& data) {
13     const long n_rows = data.n_rows;
14     const long n_cols = data.n_cols;
15     arma::Mat<double> shuffle(n_rows, n_cols);
16     for (long j=0; j<n_cols; ++j) {
17         for (long i=0; i<n_rows; ++i) {
18             shuffle(i, j) = data(std::rand()%n_rows, j);
19         }
20     }
21     return std::move(shuffle);
22 }
23
24 arma::Col<double> compute_column_means(const arma::Mat<double>& data) {
25     const long n_cols = data.n_cols;
26     arma::Col<double> means(n_cols);
27     for (long i=0; i<n_cols; ++i)
28         means(i) = arma::mean(data.col(i));
29     return std::move(means);
30 }
31
32 void remove_column_means(arma::Mat<double>& data, const arma::Col<double>& means) {
33     if (data.n_cols != means.n_elem)
34         throw std::range_error("Number of elements of means is not equal to the number

```



```

35     for (long i=0; i<long(data.n_cols); ++i)
36         data.col(i) -= means(i);
37 }
38
39 arma::Col<double> compute_column_rms(const arma::Mat<double>& data) {
40     const long n_cols = data.n_cols;
41     arma::Col<double> rms(n_cols);
42     for (long i=0; i<n_cols; ++i) {
43         const double dot = arma::dot(data.col(i), data.col(i));
44         rms(i) = std::sqrt(dot / (data.col(i).n_rows-1));
45     }
46     return std::move(rms);
47 }
48
49 void normalize_by_column(arma::Mat<double>& data, const arma::Col<double>& rms) {
50     if (data.n_cols != rms.n_elem)
51         throw std::range_error("Number of elements of rms is not equal to the number of
52     for (long i=0; i<long(data.n_cols); ++i) {
53         if (rms(i)==0)
54             throw std::runtime_error("At least one of the entries of rms equals to zero
55     data.col(i) *= 1./rms(i);
56     }
57 }
58
59 void enforce_positive_sign_by_column(arma::Mat<double>& data) {
60     for (long i=0; i<long(data.n_cols); ++i) {
61         const double max = arma::max(data.col(i));
62         const double min = arma::min(data.col(i));
63         bool change_sign = false;
64         if (std::abs(max)>=std::abs(min)) {
65             if (max<0) change_sign = true;
66         } else {
67             if (min<0) change_sign = true;
68         }
69         if (change_sign) data.col(i) *= -1;
70     }
71 }
72
73 std::vector<double> extract_column_vector(const arma::Mat<double>& data, long index) {
74     if (index<0 || index >= long(data.n_cols))
75         throw std::range_error(join("Index out of range: ", index));
76     const long n_rows = data.n_rows;
77     const double* memptr = data.colptr(index);
78     std::vector<double> result(memptr, memptr + n_rows);
79     return std::move(result);
80 }
81
82 std::vector<double> extract_row_vector(const arma::Mat<double>& data, long index) {
83     if (index<0 || index >= long(data.n_rows))
84         throw std::range_error(join("Index out of range: ", index));
85     const arma::Row<double> row(data.row(index));
86     const double* memptr = row.memptr();

```

```

87     std::vector<double> result(memptr, memptr + row.n_elem);
88     return std::move(result);
89 }
90
91 void assert_file_good(const bool& is_file_good, const std::string& filename) {
92     if (!is_file_good)
93         throw std::ios_base::failure(join("Cannot open file: ", filename));
94 }
95
96 double get_mean(const std::vector<double>& iter) {
97     const double init = 0;
98     return std::accumulate(iter.begin(), iter.end(), init) / iter.size();
99 }
100
101 double get_sigma(const std::vector<double>& iter) {
102     const double mean = get_mean(iter);
103     double sum = 0;
104     for (auto v=iter.begin(); v!=iter.end(); ++v)
105         sum += std::pow(*v - mean, 2.);
106     return std::sqrt(sum/(iter.size()-1));
107 }
108
109 } //utils

```

最后，我们来编写 `pca.i` 接口文件，也就是代码清单 9 的内容。

代码清单 9，`pca.i`：

 复制代码


```

1 %module pca
2
3 %include "std_string.i"
4 %include "std_vector.i"
5
6 namespace std {
7     %template(DoubleVector) vector<double>;
8 }
9
10 %{
11 #include "pca.h"
12 #include "utils.h"
13 %}
14
15 %include "pca.h"
16 %include "utils.h"

```

这里需要注意的是，我们在 C++ 代码中使用了熟悉的顺序容器 `std::vector`，但由于模板类比较特殊，我们需要用 `%template` 声明一下。

一切就绪后，我们执行下面的命令行，生成 `_pca.so` 库供 Python 使用：

 复制代码

```
1 $ swig -c++ -python pca.i # 解释接口定义生成包 SWIG 装器代码
2 $ g++ -fPIC -c pca.h pca.cpp utils.h utils.cpp pca_wrap.cxx -I/usr/include/python3.7 # :
3 $ g++ -shared pca.o pca_wrap.o utils.o -o _pca.so -O2 -Wall -std=c++11 -pthread -shared
```

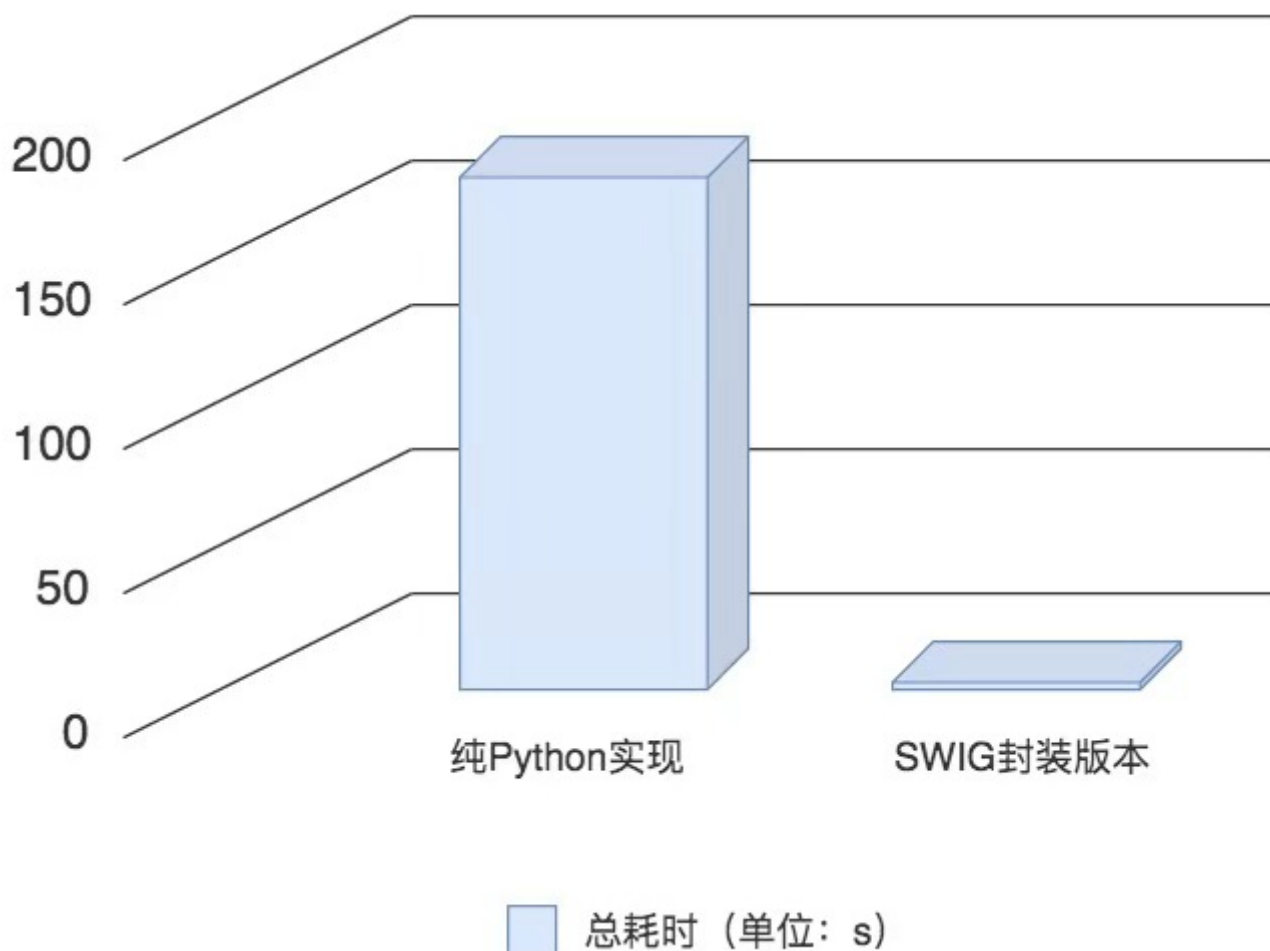
接着，我们使用 Python 脚本，导入我们创建好的 so 动态库；然后，调用相应的类的函数。这部分内容，我写在了代码清单 10 中。

代码清单 10，`testPCA.py`：

 复制代码

```
1 import pca
2
3 pca_inst = pca.pca(2)
4 pca_inst.add_record([1.0, 1.0])
5 pca_inst.add_record([2.0, 2.0])
6 pca_inst.add_record([4.0, 1.0])
7
8 pca_inst.solve()
9
10 energy = pca_inst.get_energy()
11 eigenvalues = pca_inst.get_eigenvalues()
12
13 print(energy)
14 print(eigenvalues)
```

最后，我们分别对纯 Python 实现的代码，和使用 SWIG 封装的版本来进行测试，各自都执行 1,000,000 次，然后对比执行时间。我用一张图表示了我的机器上得到的结果，你可以对比看看。



虽然这样粗略的比较并不够严谨，比如我们没有认真考虑 SWIG 接口类型转换的耗时，也没有考虑在不同编程语言下实现算法的逻辑等等。但是，通过这个粗略的结果，你仍然可以看出执行类似运算时，两者性能的巨大差异。

SWIG C++ 常用工具


到这里，你应该已经可以开始动手操作了，把上面的代码清单当作你的工具进行实践。不过，SWIG 本身非常丰富，所以这里我也再给你总结介绍几个常用的工具。

1. 全局变量

在 Python 中，我们可以通过 `cvar`，来访问 C++ 代码中定义的全局变量。

比如说，我们在头文件 `sample.h` 中定义了一个全局变量，并在 `sample.i` 中对其进行引用，也就是代码清单 11 和 12 的内容。

代码清单 11，`sample.h`：

 复制代码

```
1 #include <stdint>
2 int32_t score = 100;
```


代码清单 12, sample.i :

 复制代码

```
1 %module sample
2 %{
3 #include "sample.h"
4 %}
5
6 %include "sample.h"
```

这样，我们就可以直接在 Python 脚本中，通过 `cvar` 来访问对应的全局变量，如代码清单 13 所示，输出结果为 100。

代码清单 13, sample.py :


 复制代码

```
1 import sample
2 print sample.cvar.score
```

2. 常量

我们可以在接口定义文件中，使用 `%constant` 来设定常量，如代码清单 14 所示。

代码清单 14, sample.i :

 复制代码

```
1 %constant int foo = 100;
2 %constant const char* bar = "foobar2000";
```

3. Enumeration

我们可以在接口文件中，使用 enum 关键字来定义 enum。

4. 指针和引用

在 C++ 世界中，指针是永远也绕不开的一个概念。它无处不在，我们也无时无刻不需要使用它。因此，在这里，我认为很有必要介绍一下，如何对 C++ 中的指针和引用进行操作。


SWIG 对指针有着较为不错的支持，对智能指针也有一定的支持，而且在近期的更新日志中，我发现它对智能指针的支持一直在更新。下面的代码清单 15 和 16，就展示了针对指针和引用的使用方法。

代码清单 15，sample.h：

 复制代码

```
1 #include <cstdlib>
2
3 void passPointer(ClassA* ptr) {
4     printf("result= %d", ptr->result);
5 }
6
7 void passReference(const ClassA& ref) {
8     printf("result= %d", ref.result);
9 }
10
11 void passValue(ClassA obj) {
12     printf("result= %d", obj.result);
13 }
```

代码清单 16，sample.py：


 复制代码

```
1 import sample
2
3 a = ClassA() # 创建 ClassA 实例
4 passPointer(a)
5 passReference(a)
6 passValue(a)
```

5. 字符串

我们在工业级代码中，时常使用`std::string`。而在 SWIG 的环境下，使用标准库中的字符串，需要你在接口文件中声明`%include "std_string.i"`，来确保实现 C++ `std::string`到 Python `str`的自动转换。具体内容我放在了代码清单 17 中。

代码清单 17, `sample.i` :


 复制代码

```
1 %module sample
2
3 %include "std_string.i"
4
```

6. 向量

`std::vector`是 STL 中最常见也是使用最频繁的顺序容器，模板类比较特殊，因此，它的使用也比字符串稍微复杂一些，需要使用`%template`进行声明。详细内容我放在了代码清单 18 中。

代码清单 18, `sample.i` :


 复制代码

```
1 %module sample
2
3 %include "std_string.i"
4 %include "std_vector.i"
5
6 namespace std {
7   %template(DoubleVector) vector<double>;
8 }
9
```

7. 映射

`std::map` 同样是 STL 中最常见也是使用最频繁的容器。同样的，它的模板类也比较特殊，需要使用`%template`进行声明，详细内容可见代码清单 19。

代码清单 19, `sample.i` :

 复制代码

```
1 %module sample
2
3 %include "std_string.i"
4 %include "std_map.i"
5
6 namespace std {
7   %template(Int2strMap) map<int, string>;
8   %template(Str2intMap) map<string, int>;
9 }
10
```

学习路径

到此，SWIG 入门这个小目标，我们就已经实现了。今天内容可以当作一份 SWIG 的编程实践指南，我给你提供了 19 个代码清单，利用它们，你就可以上手操作了。当然，如果在这方面你还想继续精进，该怎么办呢？别着急，今天这节课的最后，我再和你分享下，我觉得比较高效的一条 SWIG 学习路径。

首先，任何技术的学习不要脱离官方文档。SWIG 网站上提供了难以置信的详尽文档，通过文档掌握 SWIG 的用法，显然是最好的一个途径。

其次，要深入 SWIG，对 C++ 有一个较为全面的掌握，就显得至关重要了。对于高性能计算来说，C++ 总是绕不开的一个主题，特别是对内存管理、指针和虚函数的应用，需要你实际上手编写 C++ 代码后，才能逐渐掌握。退一步讲，即便你只是为了封装其他 C++ 库供 Python 调用，也需要对 C++ 有一个基本了解，以便未来遇到编译或链接错误时，可以找到方向来解决问题。

最后，我再罗列一些学习素材，供你进一步学习参考。

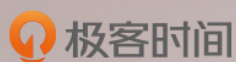
第一便是 SWIG 文档。

- a. <http://www.swig.org/doc.html>
- b. <http://www.swig.org/Doc4.0/SWIGPlus.html>
- c. PDF 版本：<http://www.swig.org/Doc4.0/SWIGDocumentation.pdf>

第二是《C++ Primer》这本书。作为 C++ 领域的经典书籍，这本书对你全面了解 C++ 有极大帮助。

第三则是《高级 C/C++ 编译技术》这本书。这本书的内容更为进阶，你可以把它作为学习 C++ 的提高和了解。

好了，今天的内容就到此结束了。关于 SWIG，你有哪些收获，或者还有哪些问题，都欢迎你留言和我分享讨论。也欢迎你把这篇文章分享给你的同事、朋友，我们一起学习和进步。



Python 核心技术与实战

系统提升你的 Python 能力

景霄

Facebook 资深工程师



新版升级：点击「👤 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 42 | 细数技术研发的注意事项

下一篇 43 | Q&A：聊一聊职业发展和选择

精选留言 (5)

写留言



gutentag

2019-08-16

对于单文件而言，用SWIG还是boost.python/py++感觉都好理解和实践，请问对于依赖关系复杂的大型C++项目（比如OpenCV, OpenSceneGraph之类的）的python binding 有没有比较完整的最佳实践呢？

C++编译的动态库python无法直接调用，C++项目的python binding本身等价于把本身编译时用到的所有的头文件中需要暴露的接口都extern成C的呢？对于头文件的相互各种i...
展开



3



许童童

2019-08-16

极客时间的C++课程快来了，期待一下，补一补我的C++。



1



安排

2019-08-16

类似于jni啊

展开



1



Ethan

2019-08-16

C++大法

展开



1



柒~龟虽寿！

2019-08-18

如何看python源代码，比如list.sort的实现

展开

