RcppCNPy: Reading and writing NumPy binary files

Dirk Eddelbuettel[a] **and Wush Wu**[b]

[a]http://dirk.eddelbuettel.com; [b]https://github.com/wush978

This version was compiled on September 17, 2017

This vignette introduces the RcppCNPy package for reading and writing files created by or for the NumPy module for Python.

1. Motivation

Python[1] is a widely-used and popular programming language. It is deployed in use cases ranging from simple scripting to larger-scale application development. Python is also popular for quantitative and scientific application due to the existence of extension modules such as **NumPy**[2] (which is shorthand for Numeric Python) and many other packages for data analysis.

NumPy is used to efficiently represent N-dimensional arrays, and provides an efficient binary storage model for these files. In practice, N is often equal to two, and matrices processed or generated in Python can be stored in this form. As **NumPy** is popular, many project utilize this file format.

R has no dedicated reading or writing functionality for these type of files. However, Carl Rogers has provided a small Cpp library called **cnpy**[3] which is released under the MIT license. Using the 'Rcpp modules' feature in **Rcpp** (Eddelbuettel and François, 2011; Eddelbuettel, 2013; Eddelbuettel et al., 2017), we provide (some) features of this library to R.

2. Examples

Data creation in Python. The first code example simply creates two files in Python: a two-dimensional rectangular array as well as a vector.

```
import numpy as np

mat = np.arange(12).reshape(3,4) * 1.1
np.save("fmat.npy", mat)
print mat

vec = np.arange(5) * 1.1
np.save("fvec.npy", vec)
print vec
```

```
#  [[  0.    1.1   2.2   3.3]
#   [  4.4   5.5   6.6   7.7]
#   [  8.8   9.9  11.   12.1]]
#  [ 0.   1.1  2.2  3.3  4.4]
```

As illustrated, Python uses the Fortran convention for storing matrices and higher-dimensional arrays: a matrix constructed from a single sequence has its first consecutive elements in its first row—whereas R, following the C convention, has these first few values in its first column. This shows that to go back and forth we need to transpose these matrices (which represented internally as two-dimensional arrays).

Data reading in R . We can read the same data in R using the `npyLoad()` function provided by the **RcppCNPy** package:

```
library(RcppCNPy)
mat <- npyLoad("fmat.npy")
mat
```

```
#       [,1] [,2] [,3] [,4]
#  [1,]  0.0  1.1  2.2  3.3
#  [2,]  4.4  5.5  6.6  7.7
#  [3,]  8.8  9.9 11.0 12.1
```

```
vec <- npyLoad("fvec.npy")
vec
```

```
#  [1] 0.0 1.1 2.2 3.3 4.4
```

The Fortran-order of the matrix is preserved; we obtain the exact same data as we stored.

Reading compressed data in R . A useful extension to the **cnpy** library is the support of **gzip**-compressed data.

```
mat2 <- npyLoad("fmat.npy.gz")
```

Support for writing compressed files has been added in version 0.2.0.

Data writing in R . Matrices and vectors can be written to files using the `npySave()` function.

```
set.seed(42)
m <- matrix(sort(rnorm(6)), 3, 2)
m
```

```
#             [,1]      [,2]
#  [1,] -0.5646982 0.4042683
#  [2,] -0.1061245 0.6328626
#  [3,]  0.3631284 1.3709584
```

```
npySave("randmat.npy", m)
v <- seq(10, 12)
v
```

[1]http://www.python.org
[2]http://numpy.scipy.org/
[3]https://github.com/rogersce/cnpy

Access method	Time in sec.	Relative to best
npyLoad(pyfile)	0.074	1.000
npyLoad(pygzfile)	0.190	2.568
read.table(txtfile)	4.189	56.608

Table 1. Performance comparison of data reads using a matrix of size $10^5 \times 50$. File size are 39.7mb for ascii, 40.0mb for npy and 10.8mb for npy.gz. Ten replications were performed, and total times are shown. R 3.3.1 was used on a laptop with an SSD disk.

```
#  [1] 10 11 12
```

```
npySave("simplevec.npy", v)
```

Data reading in Python. Reading the data back in Python is also straightforward as shown in the following example:

```
import numpy as np
m = np.load("randmat.npy")
print m
v = np.load("simplevec.npy")
print v
```

```
#  [[-0.56469817  0.40426832]
#   [-0.10612452  0.6328626 ]
#   [ 0.36312841  1.37095845]]
#  [10 11 12]
```

Integer support. Support for integer data types has been conditional on use of either the -std=c++0x or the -std=c++11 compiler extensions. Only these standards support the long long int type needed to represent int64 data on a 32-bit OS. Following the release of R 3.1.0, it has been enabled by default in **RcppCNPy** (whereas it previously required a manual rebuild), and following the release of R 3.3.0 with its updated Windows toolchain, C++11 is now available on all common R platforms. Consequently, support for large integers in **RcppCNPy** is no longer just a compile-time option for some platforms, but generally available on all (current) R installations.

Performance. The R script timing in the demo/ directory of the package **RcppCNPy** provides a simple benchmark. Given two values n and k, a matrix of size $n \times k$ is created with n rows and k columns. It is written to temporary files in i) ascii format using write.table(); ii) NumPy format using npySave(); and iii) NumPy format using npySave() with compression via the zlib library (used also by gzip).

Table~1 shows some timing comparisons for a matrix with five million elements. Reading the npy data is clearly fastest as it required only parsing of the header, followed by a single large binary read (and the transpose required to translate the representation used by R). The compressed file requires only one-fourth of the disk space, but takes approximately 2.5 times as long to read as the binary stream has be transformed. Lastly, the default ascii reading mode is clearly by far the slowest.

3. Limitations

Higher-dimensional arrays. **Rcpp** supports three-dimensional arrays, this could be support in **RcppCNPy** as well.
npz files. The **cnpy** library supports reading and writing of sets of arrays; this feature could also be exported.

4. Summary

The **RcppCNPy** package provides simple reading and writing of **NumPy** files, using the **cnpy** library. Reading of compressed files is also supported as an extension, offering more compact storage at the cost of slightly longer read times.

References

Eddelbuettel, D. (2013). *Seamless R and C++ Integration with Rcpp.* Use R! New York: Springer.

Eddelbuettel, D. and R. François (2011). Rcpp: Seamless R and C++ integration. *Journal of Statistical Software 40*(8), 1–18.

Eddelbuettel, D., R. François, J. Allaire, K. Ushey, Q. Kou, N. Russel, J. Chambers, and D. Bates (2017). *Rcpp: Seamless R and C++ Integration.* R package version 0.12.12.