RcppCNPy: Reading and writing **NumPy** binary files

Dirk Eddelbuettel

RcppCNPy version 0.0.2 as of July 7, 2012

Abstract

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

RcppCNPy is based on **cnpy**, a C++ library written by Carl Rogers.

1 Motivation

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

NumPy can be used for N-dimenional 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.

R has (as of mid-2012) no dedicated reading or writing functionality for these files. However, Carl Rogers has provided a small C++ library called **cnpy**[3] which is released under the MIT license. Using the 'Rcpp modules' feature in **Rcpp**[4][5], we provide (some) features of this library to R.

2 Examples

2.1 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
>>> mat
array([[  0. ,   1.1,   2.2,   3.3],
       [  4.4,   5.5,   6.6,   7.7],
       [  8.8,   9.9,  11. ,  12.1]])
>>> np.save("fmat.npy", mat)
>>>
>>> vec = np.arange(5) * 1.1
>>> vec
array([ 0. ,  1.1,  2.2,  3.3,  4.4])
>>> np.save("fvec.npy", vec)
>>>
```

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 tranpose these matrices (which represented internally as two-dimensional arrays).

2.2 Data reading in R

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

```
R> library(RcppCNPy)
Loading required package: Rcpp
R>
R> mat <- npyLoad("fmat.npy")
R> 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
R>
R> vec <- npyLoad("fvec.npy")
R> vec
[1] 0.0 1.1 2.2 3.3 4.4
R>
```

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

[1] http://www.python.org
[2] http://numpy.scipy.org/
[3] https://github.com/rogersce/cnpy
[4] Eddelbuettel and Françcois, 2011, JSS, 40(8), http://www.jstatsoft.org/v40/i08/
[5] http://CRAN.R-Project.org/package=Rcpp

2.3 Reading compressed data in R

A useful extension to the **cnpy** is the support of **gzip**-compressed data.

```
R> mat2 <- npyLoad("fmat.npy.gz")
R> mat2
     [,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
R>
```

Support for compressed file is currently limited to reading, but could be implemented for writing as well.

2.4 Data writing in R

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

```
R> set.seed(42)
R> m <- matrix(sort(rnorm(6)), 3, 2)
R> m
          [,1]     [,2]
[1,] -0.564698 0.404268
[2,] -0.106125 0.632863
[3,]  0.363128 1.370958
R> npySave("randmat.npy", m)
R>
R> v <- seq(10, 12)
R> v
[1] 10 11 12
R> npySave("simplevec.npy", v)
R>
```

2.5 Data reading in Python

```
>>> m = np.load("randmat.npy")
>>> m
array([[-0.56469817,  0.40426832],
       [-0.10612452,  0.6328626 ],
       [ 0.36312841,  1.37095845]])
>>>
>>> v = np.load("simpleve.npy")
>>> v
array([ 10.,  11.,  12.])
>>>
```

3 Limitations

3.1 Integer support

Support for integer data types is available, but conditional on use of the `-std=c++11` compiler extension. Only the newer standard supports the `long long int` type needed to represent `int64` data on a 32-bit OS. So until R switches to allowing `-std=c++11` on CRAN packages, users will need to rebuild both **Rcpp** and **RcppCNPy** with the switch enabled.

As shown in the previous examples, integers also transparently convert to float types.

3.2 Higher-dimensional arrays

Rcpp supports three-dimensional arrays, this could be support in **RcppCNPy** as well.

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.