**Announcement**

MDArray version 0.5.4 has Just been released. MDArray is a multi dimensional array implemented for JRuby inspired by NumPy (www.numpy.org) and Masahiro Tanaka´s Narray (narray.rubyforge.org). MDArray stands on the shoulders of Java-NetCDF and Parallel Colt. At this point MDArray has libraries for mathematical, trigonometric and descriptive statistics methods.

NetCDF-Java Library is a Java interface to NetCDF files, as well as to many other types of scientific data formats. It is developed and distributed by Unidata (http://www.unidata.ucar.edu).

Parallel Colt http://grepcode.com/snapshot/repo1.maven.org/maven2/net.sourceforge.parallelcolt/

parallelcolt/0.10.0/) is a multithreaded version of Colt (http://acs.lbl.gov/software/colt/). Colt provides a set of Open Source Libraries for High Performance Scientific and Technical Computing in Java. Scientific and technical computing is characterized by demanding problem sizes and a need for high performance at reasonably small memory footprint.

For more information and (some) documentation please go to: https://github.com/rbotafogo/mdarray/wiki

**What´s new:**

**NetCDF-3 File support**

From Wikipedia, the free encyclopedia:

"NetCDF (Network Common Data Form) is a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data. The project homepage is hosted by the Unidata program at the University Corporation for Atmospheric Research (UCAR). They are also the chief source of netCDF software, standards development, updates, etc. The format is an open standard. NetCDF Classic and 64-bit Offset Format are an international standard of the Open Geospatial Consortium.

The project is actively supported by UCAR. Version 4.0 (released in 2008) allows the use of the HDF5 data file format. Version 4.1 (2010) adds support for C and Fortran client access to specified subsets of remote data via OPeNDAP.

The format was originally based on the conceptual model of the Common Data Format developed by NASA, but has since diverged and is not compatible with it. "

This version of MDArray implements NetCDF-3 file support only. NetCDF-4 is not yet supported. At the end of this announcement we show the MDArray implementation of the NetCDF-3 file writing from the tutorial at:

http://www.unidata.ucar.edu/software/netcdf-java/tutorial/NetcdfWriting.html

**MDArray and SciRuby:**

MDArray subscribes fully to the SciRuby Manifesto (<http://sciruby.com/>).

*“*[*Ruby*](http://www.ruby-lang.org/)*has for some time had no equivalent to the beautifully constructed [NumPy](http://numpy.scipy.org/), [SciPy](http://www.scipy.org/), and [matplotlib](http://matplotlib.sourceforge.net/) libraries for*[*Python*](http://www.python.org/)*.*

*We believe that the time for a Ruby science and visualization package has come. Sometimes when a solution of sugar and water becomes super-saturated, from it precipitates a pure, delicious, and diabetes-inducing crystal of sweetness, induced by no more than the tap of a finger. So is occurring now, we believe, with numeric and visualization libraries for Ruby.”*

**MDArray main properties are:**

* Homogeneous multidimensional array, a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers;
* Easy calculation for large numerical multi dimensional arrays;
* Basic types are: boolean, byte, short, int, long, float, double, string, structure;
* Based on JRuby, which allows importing Java libraries;
* Operator: +,-,\*,/,%,\*\*, >, >=, etc.;
* Functions: abs, ceil, floor, truncate, is\_zero, square, cube, fourth;
* Binary Operators: &, |, ^, ~ (binary\_ones\_complement), <<, >>;
* Ruby Math functions: acos, acosh, asin, asinh, atan, atan2, atanh, cbrt, cos, erf, exp, gamma, hypot, ldexp, log, log10, log2, sin, sinh, sqrt, tan, tanh, neg;
* Boolean operations on boolean arrays: and, or, not;
* Fast descriptive statistics from Parallel Colt (complete list found bellow);
* Easy manipulation of arrays: reshape, reduce dimension, permute, section, slice, etc.;
* Reading of two dimensional arrays from CSV files (mainly for debugging and simple testing purposes);
* StatList: a list that can grow/shrink and that can compute Parallel Colt descriptive statistics;
* Experimental lazy evaluation (still slower than eager evaluation).

**Descriptive statistics methods imported from Parallel Colt:**

auto\_correlation, correlation, covariance, durbin\_watson, frequencies, geometric\_mean, harmonic\_mean, kurtosis, lag1, max, mean, mean\_deviation, median, min, moment, moment3, moment4, pooled\_mean, pooled\_variance, product, quantile, quantile\_inverse, rank\_interpolated, rms, sample\_covariance, sample\_kurtosis, sample\_kurtosis\_standard\_error, sample\_skew, sample\_skew\_standard\_error, sample\_standard\_deviation, sample\_variance, sample\_weighted\_variance, skew, split, standard\_deviation, standard\_error, sum, sum\_of\_inversions, sum\_of\_logarithms, sum\_of\_powers, sum\_of\_power\_deviations, sum\_of\_squares, sum\_of\_squared\_deviations, trimmed\_mean, variance, weighted\_mean, weighted\_rms, weighted\_sums, winsorized\_mean.

**Double and Float methods from Parallel Colt**:

acos, asin, atan, atan2, ceil, cos, exp, floor, greater, IEEEremainder, inv, less, lg, log, log2, rint, sin, sqrt, tan.

**Double, Float, Long and Int methods from Parallel Colt**:

abs, compare, div, divNeg, equals, isEqual (is\_equal), isGreater (is\_greater), isles (is\_less), max, min, minus, mod, mult, multNeg (mult\_neg), multSquare (mult\_square), neg, plus (add), plusAbs (plus\_abs), pow (power), sign, square.

**Long and Int methods from Parallel Colt**

and, dec, factorial, inc, not, or, shiftLeft (shift\_left), shiftRightSigned (shift\_right\_signed), shiftRightUnsigned (shift\_right\_unsigned), xor.

**MDArray installation and download:**

* Install Jruby
* jruby –S gem install mdarray

**MDArray Homepages:**

* <http://rubygems.org/gems/mdarray>
* <https://github.com/rbotafogo/mdarray/wiki>

**Contributors:**

Contributors are welcome.

**MDArray History:**

* 24/06/2013: Version 0.5.0 – Over 90% Performance improvements for methods imported from Parallel Colt and over 40% performance improvements for all other methods (implemented in Ruby);
* 16/05/2013: Version 0.5.0 - All loops transferred to Java with over 50% performance improvements. Descriptive statistics from Parallel Colt;
* 19/04/2013: Version 0.4.3 - Fixes a simple, but fatal bug in 0.4.2. No new features;
* 17/04/2013: Version 0.4.2 - Adds simple statistics and boolean operators;
* 05/04/2013: Version 0.4.0 – Initial release.

**NetCDF-3 Writing with MDArray API**

=================================

require 'mdarray'

class NetCDF

attr\_reader :dir, :filename, :max\_strlen

#---------------------------------------------------------------------------------------

#

#---------------------------------------------------------------------------------------

def initialize

@dir = "~/tmp"

@filename1 = "testWriter"

@filename2 = "testWriteRecord2"

@max\_strlen = 80

end

#---------------------------------------------------------------------------------------

# Define the NetCDF-3 file

#---------------------------------------------------------------------------------------

def define\_file

# We pass the directory, filename, filetype and optionaly the outside\_scope.

#

# I'm implementing in cygwin, so the need for method cygpath that converts the

# directory name to a Windows name. In another environment, just pass the directory

# name.

#

# Inside a block we have another scope, so the block cannot access any variables, etc.

# from the ouside scope. If we pass the outside scope, in this case we are passing self,

# we can access variables in the outside scope by using @outside\_scope.<variable>.

NetCDF.define(cygpath(@dir), @filename1, "netcdf3", self) do

# add dimensions

dimension "lat", 64

dimension "lon", 128

# add variables and attributes

# add Variable double temperature(lat, lon)

variable "temperature", "double", [@dim\_lat, @dim\_lon]

variable\_att @var\_temperature, "units", "K"

variable\_att @var\_temperature, "scale", [1, 2, 3]

# add a string-value variable: char svar(80)

# note that this is created as a scalar variable although in NetCDF-3 there is no

# string type and the string has to be represented as a char type.

variable "svar", "string", [], {:max\_strlen => @outside\_scope.max\_strlen}

# add a 2D string-valued variable: char names(names, 80)

dimension "names", 3

variable "names", "string", [@dim\_names], {:max\_strlen => @outside\_scope.max\_strlen}

# add a scalar variable

variable "scalar", "double", []

# add global attributes

global\_att "yo", "face"

global\_att "versionD", 1.2, "double"

global\_att "versionF", 1.2, "float"

global\_att "versionI", 1, "int"

global\_att "versionS", 2, "short"

global\_att "versionB", 3, "byte"

end

end

#---------------------------------------------------------------------------------------

# write data on the above define file

#---------------------------------------------------------------------------------------

def write\_file

NetCDF.write(cygpath(@dir), @filename1, self) do

temperature = find\_variable("temperature")

shape = temperature.shape

data = MDArray.fromfunction("double", shape) do |i, j|

i \* 1\_000\_000 + j \* 1\_000

end

write(temperature, data)

svar = find\_variable("svar")

write\_string(svar, "Two pairs of ladies stockings!")

names = find\_variable("names")

# careful here with the shape of a string variable. A string variable has one

# more dimension than it should as there is no string type in NetCDF-3. As such,

# if we look as names' shape it has 2 dimensions, be we need to create a one

# dimension string array.

data = MDArray.string([3], ["No pairs of ladies stockings!",

"One pair of ladies stockings!",

"Two pairs of ladies stockings!"])

write\_string(names, data)

# write scalar data

scalar = find\_variable("scalar")

write(scalar, 222.333 )

end

end

#---------------------------------------------------------------------------------------

# Define a file for writing one record at a time

#---------------------------------------------------------------------------------------

def define\_one\_at\_time

NetCDF.define(cygpath(@dir), @filename2, "netcdf3", self) do

dimension "lat", 3

dimension "lon", 4

# zero sized dimension is an unlimited dimension

dimension "time", 0

variable "lat", "float", [@dim\_lat]

variable\_att @var\_lat, "units", "degree\_north"

variable "lon", "float", [@dim\_lon]

variable\_att @var\_lon, "units", "degree\_east"

variable "rh", "int", [@dim\_time, @dim\_lat, @dim\_lon]

variable\_att @var\_rh, "long\_name", "relative humidity"

variable\_att @var\_rh, "units", "percent"

variable "T", "double", [@dim\_time, @dim\_lat, @dim\_lon]

variable\_att @var\_t, "long\_name", "surface temperature"

variable\_att @var\_t, "units", "degC"

variable "time", "int", [@dim\_time]

variable\_att @var\_time, "units", "hours since 1990-01-01"

end

end

#---------------------------------------------------------------------------------------

# Define a file for writing one record at a time

#---------------------------------------------------------------------------------------

def write\_one\_at\_time

NetCDF.write(cygpath(@dir), @filename2, self) do

lat = find\_variable("lat")

lon = find\_variable("lon")

# write non recored data to the variables

write(lat, MDArray.float([3], [41, 40, 39]))

write(lon, MDArray.float([4], [-109, -107, -105, -103]))

# get record variables from file

rh = find\_variable("rh")

time = find\_variable("time")

t = find\_variable("T")

# there is no method find\_dimension for NetcdfFileWriter, so we need to get the

# dimension from a variable.

rh\_shape = rh.shape

dim\_lat = rh\_shape[1]

dim\_lon = rh\_shape[2]

(0...10).each do |time\_idx|

# fill rh\_data array

rh\_data = MDArray.fromfunction("int", [dim\_lat, dim\_lon]) do |lat, lon|

time\_idx \* lat \* lon

end

# reshape rh\_data so that it has the same shape as rh variable

# Method reshape! reshapes the array in-place without data copying.

rh\_data.reshape!([1, dim\_lat, dim\_lon])

# fill temp\_data array

temp\_data = MDArray.fromfunction("double", [dim\_lat, dim\_lon]) do |lat, lon|

time\_idx \* lat \* lon / 3.14159

end

# reshape temp\_data array so that it has the same shape as temp variable.

temp\_data.reshape!([1, dim\_lat, dim\_lon])

# write the variables

write(time, MDArray.int([1], [time\_idx \* 12]), [time\_idx])

write(rh, rh\_data, [time\_idx, 0, 0])

write(t, temp\_data, [time\_idx, 0, 0])

end # End time\_idx loop

end

end

end

netcdf = NetCDF.new

netcdf.define\_file

netcdf.write\_file

netcdf.define\_one\_at\_time

netcdf.write\_one\_at\_time