**Announcement**

MDArray version 0.5.5.1 has been released. MDArray is a multi dimensional array implemented for JRuby inspired by NumPy ([www.numpy.org](http://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 linear algebra, mathematical, trigonometric and descriptive statistics methods.

NetCDF-Java Library is a Java interface to [NetCDF files](http://www.unidata.ucar.edu/software/netcdf/index.html), 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](http://en.wikipedia.org/wiki/Thread_%28computer_science%29) version of [Colt](http://dsd.lbl.gov/~hoschek/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.

**What´s new:**

This version is a minor version and has fixes for some small bugs, configuration files for using during development. It isn´t a critical upgrade.

**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;
* Support for many linear algebra methods (see bellow);
* 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).

**Supported linear algebra methods:**

* Backward\_solve: Solves the upper triangular system U\*x=b;
* chol: Constructs and returns the cholesky-decomposition of the given matrix.
* cond: Returns the condition of matrix A, which is the ratio of largest to smallest singular value.
* det: Returns the determinant of matrix A.
* eig: Constructs and returns the Eigenvalue-decomposition of the given matrix.
* Forward\_solve: Solves the lower triangular system L\*x=b;
* inverse: Returns the inverse or pseudo-inverse of matrix A.
* kron: Computes the Kronecker product of two real matrices.
* lu: Constructs and returns the LU-decomposition of the given matrix.
* mult: Inner product of two vectors; Sum(x[i] \* y[i]).
* mult: Linear algebraic matrix-vector multiplication; z = A \* y.
* mult: Linear algebraic matrix-matrix multiplication; C = A x B.
* mult\_outer: Outer product of two vectors; Sets A[i,j] = x[i] \* y[j].
* norm1: Returns the one-norm of vector x, which is Sum(abs(x[i])).
* norm1: Returns the one-norm of matrix A, which is the maximum absolute column sum.
* norm2: Returns the two-norm (aka euclidean norm) of vector x; equivalent to Sqrt(mult(x,x)).
* norm2: Returns the two-norm of matrix A, which is the maximum singular value; obtained from SVD.
* normF: Returns the Frobenius norm of matrix A, which is Sqrt(Sum(A[i]2)).
* normF: Returns the Frobenius norm of matrix A, which is Sqrt(Sum(A[i,j]2)).
* norm\_nfinity: Returns the infinity norm of vector x, which is Max(abs(x[i])).
* norm\_infinity: Returns the infinity norm of matrix A, which is the maximum absolute row sum.
* pow: Linear algebraic matrix power; B = Ak <==> B = A\*A\*...\*A.
* qr: Constructs and returns the QR-decomposition of the given matrix.
* rank: Returns the effective numerical rank of matrix A, obtained from Singular Value Decomposition.
* solve: Solves A\*x = b.
* solve: Solves A\*X = B.
* solve\_transpose: Solves X\*A = B, which is also A'\*X' = B'.
* svd: Constructs and returns the SingularValue-decomposition of the given matrix.
* trace: Returns the sum of the diagonal elements of matrix A; Sum(A[i,i]).
* trapezoidal\_lower: Modifies the matrix to be a lower trapezoidal matrix.
* vector\_norm2: Returns the two-norm (aka euclidean norm) of vector X.vectorize();
* xmult\_outer: Outer product of two vectors; Returns a matrix with A[i,j] = x[i] \* y[j].
* xpow\_slow: Linear algebraic matrix power; B = Ak <==> B = A\*A\*...\*A.

**Properties´ methods tested on matrices:**

* density: Returns the matrix's fraction of non-zero cells; A.cardinality() / A.size().
* generate\_non\_singular: Modifies the given square matrix A such that it is diagonally dominant by row and column, hence non-singular, hence invertible.
* diagonal?: A matrix A is diagonal if A[i,j] == 0 whenever i != j.
* diagonally\_dominant\_by\_column?: A matrix A is diagonally dominant by column if the absolute value of each diagonal element is larger than the sum of the absolute values of the off-diagonal elements in the corresponding column.
* diagonally\_dominant\_by\_row?: A matrix A is diagonally dominant by row if the absolute value of each diagonal element is larger than the sum of the absolute values of the off-diagonal elements in the corresponding row.
* identity?: A matrix A is an identity matrix if A[i,i] == 1 and all other cells are zero.
* lower\_bidiagonal?: A matrix A is lower bidiagonal if A[i,j]==0 unless i==j || i==j+1.
* lower\_triangular?: A matrix A is lower triangular if A[i,j]==0 whenever i < j.
* nonnegative?: A matrix A is non-negative if A[i,j] >= 0 holds for all cells.
* orthogonal?: A square matrix A is orthogonal if A\*transpose(A) = I.
* positive?: A matrix A is positive if A[i,j] > 0 holds for all cells.
* singular?: A matrix A is singular if it has no inverse, that is, iff det(A)==0.
* skew\_symmetric?: A square matrix A is skew-symmetric if A = -transpose(A), that is A[i,j] == -A[j,i].
* square?: A matrix A is square if it has the same number of rows and columns.
* strictly\_lower\_triangular?: A matrix A is strictly lower triangular if A[i,j]==0 whenever i <= j.
* strictly\_triangular?: A matrix A is strictly triangular if it is triangular and its diagonal elements all equal 0.
* strictly\_upper\_triangular?: A matrix A is strictly upper triangular if A[i,j]==0 whenever i >= j.
* symmetric?: A matrix A is symmetric if A = tranpose(A), that is A[i,j] == A[j,i].
* triangular?: A matrix A is triangular iff it is either upper or lower triangular.
* tridiagonal?: A matrix A is tridiagonal if A[i,j]==0 whenever Math.abs(i-j) > 1.
* unit\_triangular?: A matrix A is unit triangular if it is triangular and its diagonal elements all equal 1.
* upper\_bidiagonal?: A matrix A is upper bidiagonal if A[i,j]==0 unless i==j || i==j-1.
* upper\_triangular?: A matrix A is upper triangular if A[i,j]==0 whenever i > j.
* zero?: A matrix A is zero if all its cells are zero.
* lower\_bandwidth: The lower bandwidth of a square matrix A is the maximum i-j for which A[i,j] is nonzero and i > j.
* semi\_bandwidth: Returns the semi-bandwidth of the given square matrix A.
* upper\_bandwidth: The upper bandwidth of a square matrix A is the maximum j-i for which A[i,j] is nonzero and j > i.

**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:**

* 28/10/2013: Version 0.5.5 – class MDMatrix and Support for linear algebra;
* 07/08/2013: Version 0.5.4 – Support for NetCDF files reading and writing;
* 24/06/2013: Version 0.5.3 – 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.