

ArmadilloJava

# Master test plan

Armadillo C++ like linear algebra library in pure Java

Sebastian Niemann  
29.04.2014

## Content

Introduction.....	2
Features to be tested .....	3
Features not to be tested .....	4
Approach .....	5
Input types.....	5
Input instances .....	7
Bug-report based input values .....	7
Independent methods.....	10
Expected results .....	10
Static methods.....	10
In-place methods.....	10
Random methods .....	10
Mutual influencing methods .....	11
Item pass/fail criteria .....	12
Environmental needs .....	13
References.....	14
Glossary .....	15

# Introduction

---

ArmadilloJava provides Java based interfaces similar to the Armadillo C++ algebra library (called Armadillo C++ further on) by Conrad Sanderson et al., NICTA, Australia.

The ArmadilloJava library is mainly based on three kinds of functional, public classes:

- Matrix/Vector classes called *Mat*, *Col* and *Row*, which hold the logic structure of general matrices, column vectors and row vectors, as well as member methods that perform some linear algebra or statistic calculations on these.
- A single class called *Arma* consisting of static functions that performs some linear algebra or statistic calculations on provided Matrix/Vector classes.
- Miscellaneous functions that provided pre-defined constants (Called *Datum*), online statistic calculations (Called *RunningStat* for double entries and *RunningStatVec* for vector inputs) and CPU time measurements (called *WallClock*)

Both, the methods within the matrix\vector classes as well as the methods within *Arma* are mainly independent of other functions within this library.

**The main purpose of these tests is to verify the mathematical correctness of the implemented methods and the API compatibility towards Armadillo C++.**

# Features to be tested

---

**Any** implemented public method and attribute of the classes ...

- *Mat*
- *Col*
- *Row*
- *Arma*
- *Datum*
- *RunningStat*
- *RunningStatVec*
- *WallClock*

... are to be tested regarding their mathematical correctness of the implemented method and the API compatibility towards Armadillo C++.

The mathematical correctness must guarantee that each result is correct within a relative error of  $10^{-12}$  due to loss of information because of the finite representation of real valued number. To be more precise, a number  $a$  is classified as the correct result compared to a pre-calculated result  $b$ , if  $|a - b| < |b| \cdot 10^{-12}$ .

The tests for compatibility needs to verify that each method is present in both libraries and requires the same number and order of input types - ignoring differences that are purely based on technical differences between Java and C++. It must further be ensured, that invalid input values for Armadillo C++ are also handled in the same way by ArmadilloJava.

**In addition, all of these are to be verified for recent versions of the Oracle Java JDK and OpenJDK.**

# Features not to be tested

---

- It is not tested whether the computation time is similar between ArmadilloJava and Armadillo C++ (As we always expect the C++ implementation to be faster) ...
- ... neither will be tested if the used pseudo random number generators will provided the identical sequences for the same random seed ...
- ... nor is verified if both libraries can make use of the same precompiled LAPACK/BLAS libraries.

# Approach

---

The tests are based on unit tests, build with a recent version of [JUnit](#). Automated integration tests are handled by [Travis CI](#).

The tests are organised by the different types of methods to be tested. The functions within ArmadilloJava are firstly split into methods that work independent of each other and methods that influence each other.

The first type of methods is further divided into static types that will always return the same result for the same input and methods that are influenced by any kind of pseudo random behaviour.

The computational static methods are then finally split into in-place and out-of-place.

Summed up, the methods within ArmadilloJava are classified as:

- Independent methods (As present in [Mat](#), [Col](#), [Row](#), [Arma](#) and [Datum](#))
  - Random methods (Any method beginning with *rand*)
  - Static methods (All other methods)
    - In-place operations ([inplace\\_trans](#) and operations on sub-views)
    - Out-of-place operations (All other methods)
- Mutual influencing methods (As present in [RunningStat](#), [RunningStatVec](#) and [WallClock](#))

## Input types

The following base types of input values are used within ArmadilloJava:

- `int` Integer values (*primitive Java type*)
- `double` Real valued double precision values (*primitive Java type*)
- `Span` Class representing an integer interval
- `Size` Class representing the size of a matrix
- `AbstractMat` Class for general real valued dense matrices/vectors
- `Mat` Class for general real valued dense matrices (excluding vectors)
- `AbstractVector` Class for general real valued dense vectors
- `Col` Class for general real valued dense column vectors
- `Row` Class for general real valued dense row vectors
- `String` Strings
- `Op` Enum to specify unary, binary and relational operations
- `OutputStream` Output stream
- `InputStream` Input stream
- `FileType` Enum of supported file types for I/O-operations
- `DistParam` Class containing two integer values
- `Fill` Enum of supported fill strategy for newly generated matrices/vectors

These are partitioned in equivalent classes, based on their varying usage/meaning within the library. The following table also includes abbreviations for each equivalent class to ease the reference later on. The abbreviation is also used in the naming of the test classes.

Abbreviation	Type	Description
<b>ElemInd</b>	int	Element index
<b>ColInd</b>	int	Column index
<b>ExtColInd</b>	int	Extended column index
<b>RowInd</b>	int	Row index
<b>ExtRowInd</b>	int	Extended row index
<b>NumElems</b>	int	Number of elements
<b>NumCols</b>	int	Number of columns
<b>NumRows</b>	int	Number of rows
<b>Normal</b>	int	Normalisation type
<b>Dim</b>	int	Dimension
<b>MatNormInt</b>	int	Matrix norm
<b>VecNormInt</b>	int	Vector norm
<b>GenDouble</b>	double	General real valued input
<b>SinValTol</b>	double	Singular value tolerance (Focused around the largest and smallest singular value of a matrix)
<b>ElemIndRange</b>	Span	Element index range
<b>ColIndRange</b>	Span	Column index range
<b>RowIndRange</b>	Span	Row index range
<b>MatSize</b>	Size	Matrix size
<b>GenMat</b>	Mat	General matrix input
<b>SquMat</b>	Mat	Square matrix input
<b>InvMat</b>	Mat	Invertable matrix input
<b>SymMat</b>	Mat	Symmetric matrix input
<b>SymPDMat</b>	Mat	Symmetric and positive-definite matrix input
<b>LogicMat</b>	Mat	Logical matrix
<b>OOMat</b>	Mat	(1, 1)-matrix
<b>GenColVec</b>	Col	General column vector input
<b>MonColVec</b>	Col	Monotone column vector input
<b>LogicColVec</b>	Col	Logical column vector input
<b>OOColVec</b>	Col	(1, 1)-column vector input
<b>GenRowVec</b>	Row	General row vector input
<b>MonRowVec</b>	Row	Monotone row vector input
<b>LogicRowVec</b>	Row	Logical row vector input
<b>OORowVec</b>	Row	(1, 1)-row vector input
<b>ElemIndsAsColVec</b>	Col	Element indices as column vectors
<b>ElemIndsAsRowVec</b>	Row	Element indices as row vectors
<b>ColIndsAsColVec</b>	Col	Column indices as column vectors
<b>ColIndsAsRowVec</b>	Row	Column indices as row vectors
<b>RowIndsAsColVec</b>	Col	Row indices as column vectors
<b>RowIndsAsRowVec</b>	Row	Row indices as row vectors
<b>Text</b>	String	Text
<b>FilePath</b>	String	File path
<b>MatNormString</b>	String	Matrix norm

<b>VecNormString</b>	String	Vector norm
<b>Sort</b>	String	Sort direction
<b>Search</b>	String	Search direction
<b>SinValSel</b>	String	Singular vector selection
<b>OutputStream</b>	OutputStream	Output stream
<b>InputStream</b>	Input stream	Input stream
<b>FileType</b>	FileType	File type
<b>DistrParam</b>	DistrParam	Distribution interval
<b>Fill</b>	Fill	Fill type

Table 1: Types and functionality of input parameters

## Input instances

The next table presents the valid domain range and the instances to be tested for each equivalent class.

The input parameters to be tested are included based on three different aspects:

1. Is the input a commonly used value?
2. Covers the input a special kind or combination of values, which may be treated differently by the methods to be tested?
3. Was at any time a reproducible bug reported for this input value?

Any value that fulfils at least one of these aspects will be tested.

## Bug-report based input values

*None currently*



Abbreviation	Domain range	Test instances
<b>ElemInd</b>	{0, 1, ..., n - 1}, n = Number of elements	0, 9
<b>ColInd</b>	{0, 1, ..., n - 1}, n = Number of columns	0, 1
<b>ExtColInd</b>	{0, 1, ..., n}, n = Number of columns	0, 1, 2
<b>RowInd</b>	{0, 1, ..., n - 1}, n = Number of rows	0, 1
<b>ExtRowInd</b>	{0, 1, ..., n}, n = Number of rows	0, 1, 2
<b>NumElems</b>	Natural numbers	1, 10
<b>NumCols</b>	Natural numbers	1, 5
<b>NumRows</b>	Natural numbers	1, 5
<b>Normal</b>	{0, 1}	0, 1
<b>Dim</b>	{0, 1}	0, 1
<b>Exp</b>	Positive real numbers	0.5, 1, 2, 3
<b>MatNormInt</b>	{1, 2}	1, 2
<b>VecNormInt</b>	Natural numbers	1, 2, 3, 4
<b>GenDouble</b>	Real numbers	-inf, -2, 0, machine epsilon, 0.5, 1, euler number, pi, inf
<b>SinValTol</b>	Real numbers	0, 1
<b>ElemIndRange</b>	Interval [a, b], a >= 0, b < n, n = Number of elements	[0, 0], [0, 9], [1, 4]
<b>ColIndRange</b>	Interval [a, b], a >= 0, b < n, n = Number of columns	Span.all(), [0, 0], [0, 4], [1, 2]
<b>RowIndRange</b>	Interval [a, b], a >= 0, b < n, n = Number of rows	Span.all(), [0, 0], [0, 4], [1, 2]
<b>MatSize</b>	2-tuple of natural numbers	(1, 1), (1, 5), (2, 5), (5, 1), (5, 2), (5, 5)
<b>GenMat</b>	Any real valued matrix	Union of SquMat, matrices of ones, zero matrices and {{inf, -inf}, {-inf, inf}}
<b>SquMat</b>	Any real valued square matrix	Union of InvMat and SymMat
<b>InvMat</b>	Any real valued invertible matrix	Identity matrices and Kac-Murdock-Szegő matrices
<b>SymMat</b>	Any real valued symmetric matrix	Union of SymPDMat, zero matrices and matrices of ones
<b>SymPDMat</b>	Any real valued symmetric and positive-definite matrix	Identity matrices and Hilbert matrices
<b>LogicMat</b>	Any real valued matrix	Zero matrices, matrices of ones and identity matrices
<b>OOMat</b>	Any real valued matrix with just one element	Filled by GenDouble
<b>GenColVec</b>	Any real valued column vector	Transposition of GenRowVec

<b>MonColVec</b>	Any real valued column vector with strict monotone increasing values	Transposition of MonRowVec
<b>LogicColVec</b>	Any real valued column vector	Transposition of LogicRowVec
<b>OOColVec</b>	Any real valued column vector with just one element	Filled by GenDouble
<b>GenRowVec</b>	Any real valued row vector	Zero vectors, vectors of ones as well as the first row of identity, Hilbert, Kac-Murdock-Szegö matrices and {inf, -inf}
<b>MonRowVec</b>	Any real valued row vector with strict monotone increasing values	(-10, -5, 0.5, 10), (-inf, 0, inf), (0), (-inf), (inf)
<b>LogicRowVec</b>	Any real valued row vector	Zero vectors, vectors of ones and the first row of identity matrices
<b>OORowVec</b>	Any real valued row vector with just one element	Filled by GenDouble
<b>ElemInds</b>	Subset of {0, 1, ..., n - 1}, n = Number of elements	(0), (9), (9, 0, 1), (1, 1, 1)
<b>ColInds</b>	Subset of {0, 1, ..., n - 1}, n = Number of columns	Same as RowInds
<b>RowInds</b>	Subset of {0, 1, ..., n - 1}, n = Number of rows	(0), (4), (4, 0, 1), (1, 1, 1)
<b>Text</b>	Any UTF-8 string	"Test string"
<b>FilePath</b>	Any UTF-8 file path with existing folder structure	"../test.mat"
<b>MatNormString</b>	{"inf", "fro"}	"inf", "fro"
<b>VecNormString</b>	{"inf", "-inf", "fro"}	"inf", "-inf", "fro"
<b>Sort</b>	{"ascend", "descend"}	"ascend", "descend"
<b>Search</b>	{"first", "last"}	"first", "last"
<b>SinValSel</b>	{"left", "right", "both"}	"left", "right", "both"
<b>OutputStream</b>	Any stream extending OutputStream	System.out, System.err, FileOutputStream("./test.mat", false);
<b>InputStream</b>	Any stream extending InputStream	FileInputStream("./test.mat")
<b>FileType</b>	{RAW_ASCII}	RAW_ASCII
<b>DistrParam</b>	2-tupel of integers	{0, 10}, {1, 1}, {-5, 6}
<b>Fill</b>	{NONE, ZEROS, ONES, EYE, RANDU, RANDN}	NONE, ZEROS, ONES, EYE, RANDU, RANDN

Table 2: Domain range and test instances of input parameters

## Independent methods

All independent methods are grouped into parameterised test classes, based on their sub-classification (static/random and in-place/out-of-place) and the equivalent class of input values they are depending on.

These test classes are named as ...

`Test(InPlace|)(Random|)(Equivalent classes).java`

... where *[Equivalent classes]* stands for a natural ascending ordered concatenation of abbreviation of the equivalent class required by all methods within this test class.

For example, a test class named *TestStaticOutOfPlaceColIndRowInd.java* will contains all independent methods that are randomless, operate out-of-place and requires both column index as well as row index values.

## Expected results

The expected values to be compared against are generated by Armadillo C++ and save in their ASCII form within the folder

`./data/expected/[Name of the test class, without '.java'].mat`

## Static methods

### *Out-of-place methods*

Before each test, a deep copy of the input values is stored and after each execution compared against the input values provided to the test method to ensure that the input values were at least not noticeable modified.

### *In-place methods*

*Basic unit test.*

## Random methods

Each test method encapsulates the actual test within a loop that is iterated 100 times and aggregates all generated results within a single vector, as we expect that the random distribution should be detectable after this.

The vector is then used to create a histogram that divides the range of values within the vector in 10 evenly wide bins.

Methods within ArmadilloJava that are based on a uniform distribution shall have the same amount of entries within each bin, within a maximal acceptable error of 10.

Methods that are based on a normal distribution shall have an amount of entries in each bin that correlates with the expected value for the probability dense function, also within a maximal acceptable error of 10.

## Mutual influencing methods

All of these methods are tested within a single test class grouped by their parent class in `ArmadilloJava`, which should result in:

- `TestRunningStat.java`
- `TestRunningtatVec.java`
- `WallCock.java`

The actual test is than individually implemented.

For *RunningStat* and *RunningStatVec*, the actual results are to be compared against pre-calculated expectations, while *WallClock* is measured by the expected difference in time between two CPU-time measurements.

# Item pass/fail criteria

---

An item fails if a unit test is erroneous or fails. The whole test fails, if any item fails.

The test passes, if **everything** is tested within *Features to be tested* (According to *Approach*) and not a single item fails.

# Environmental needs

---

All tests should be executable out of the box after including the whole maven based project.

# References

---

M. Dow, Explicit inverses of Toeplitz and associated matrices, Australian & New Zealand Industrial and Applied Mathematics Journal 44 (2003), Edition E , 185-215.

M. Kac, W. L. Murdock, and G. Szegő, On the eigenvalues of certain Hermitian forms, Journal of Rational Mechanics and Analysis, Volume 2 (1953), 787-800

# Glossary

---

<b>Mat</b>	Java class for general matrices
<b>Col</b>	Java class for column vectors
<b>Vec</b>	Java class for row vectors
<b>Arma</b>	Java class for linear algebra functions
<b>Datum</b>	Java class for pre-defined constants
<b>RunningStat</b>	Java class for online statistic based on single double values
<b>RunningStatVec</b>	Java class for online statistic based on vectors
<b>WallClock</b>	Java class for C PU time measurements
<b>KMS</b>	Kac-Murdock-Szegö matrix, a special instance of a toeplitz matrix