ArmadilloaJava

Master test plan

Armadillo C++ like linear algebra library in pure Java

Sebastian Niemann
29.04.2014

Content

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

VecNormString	String	Vector norm
Sort	String	Sort direction
Search	String	Search direction
SinValSel	String	Singular vector selection
OutputStream	OutputStream	Output stream
InputStream	Input stream	Input stream
FileType	FileType	File type
DistrParam	DistrParam	Distribution interval
Fill	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
ElemInd	{0, 1, …, n - 1}, n = Number of elements	0, 9
ColInd	{0, 1, …, n - 1}, n = Number of columns	0, 1
ExtColInd	{0, 1, …, n}, n = Number of columns	0, 1, 2
RowInd	{0, 1, …, n - 1}, n = Number of rows	0, 1
ExtRowInd	{0, 1, …, n}, n = Number of rows	0, 1, 2
NumElems	Natural numbers	1, 10
NumCols	Natural numbers	1, 5
NumRows	Natural numbers	1, 5
Normal	{0, 1}	0, 1
Dim	{0, 1}	0, 1
Exp	Positive real numbers	0.5, 1, 2, 3
MatNormInt	{1, 2}	1, 2
VecNormInt	Natural numbers	1, 2, 3, 4
GenDouble	Real numbers	-inf, -2, 0, machine epsilon, 0.5, 1, euler number, pi, inf
SinValTol	Real numbers	0, 1
ElemIndRange	Interval [a, b], a >= 0, b < n, n = Number of elements	[0, 0], [0, 9], [1, 4]
ColIndRange	Interval [a, b], a >= 0, b < n, n = Number of columns	[0, 0], [0, 4], [1, 2]
RowIndRange	Interval [a, b], a >= 0, b < n, n = Number of rows	[0, 0], [0, 4], [1, 2]
MatSize	2-tuple of natural numbers	(1, 1), (1, 5), (2, 5), (5, 1), (5, 2), (5, 5)
GenMat	Any real valued matrix	Union of SquMat, matrices of ones, zero matrices and {{inf, -inf}, {-inf, inf}}
SquMat	Any real valued square matrix	Union of InvMat and SymMat
InvMat	Any real valued invertible matrix	Identity matrices and Kac-Murdock-Szegö matrices
SymMat	Any real valued symmetric matrix	Union of SymPDMat, zero matrices and matrices of ones
SymPDMat	Any real valued symmetric and positive-definite matrix	Identity matrices and Hilbert matrices
LogicMat	Any real valued matrix	Zero matrices, matrices of ones and identity matrices
OOMat	Any real valued matrix with just one element	Filled by GenDouble
GenColVec	Any real valued column vector	Transposition of GenRowVec

MonColVec	Any real valued column vector with strict monotone increasing values	Transposition of MonRowVec
LogicColVec	Any real valued column vector	Transposition of LogicRowVec
OOColVec	Any real valued column vector with just one element	Filled by GenDouble
GenRowVec	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}
MonRowVec	Any real valued row vector with strict monotone increasing values	(-10, -5, 0.5, 10), (-inf, 0, inf), (0), (-inf), (inf)
LogicRowVec	Any real valued row vector	Zero vectors, vectors of ones and the first row of identity matrices
OORowVec	Any real valued row vector with just one element	Filled by GenDouble
ElemInds	Subset of {0, 1, …, n - 1}, n = Number of elements	(0), (9), (9, 0, 1), (1, 1, 1)
ColInds	Subset of {0, 1, …, n - 1}, n = Number of columns	Same as RowInds
RowInds	Subset of {0, 1, …, n - 1}, n = Number of rows	(0), (4), (4, 0, 1), (1, 1, 1)
Text	Any UTF-8 string	"Test string"
FilePath	Any UTF-8 file path with existing folder structure	"./test.mat"
MatNormString	{"inf", "fro"}	"inf", "fro"
VecNormString	{"inf", "-inf", "fro"}	"inf", "-inf", "fro"
Sort	{"ascend", "descend"}	"ascend", "descend"
Search	{"first", "last"}	"first", "last"
SinValSel	{"left", "right", "both"}	"left", "right", "both"
OutputStream	Any stream extending OutputStream	System.out, System.err, FileOutputStream("./test.mat", false);
InputStream	Any stream extending InputStream	FileInputStream("./test.mat")
FileType	{RAW_ASCII}	RAW_ASCII
DistrParam	2-tupel of integers	{0, 10}, {1, 1}, {-5, 6}
Fill	{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. Szegö, On the eigenvalues of certain Hermitian forms, Journal of Rational Mechanics and Analysis, Volume 2 (1953), 787-800

Glossary

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