System Test Plan
Input Space Partitioning

By: Karl Dylan Baes

Table of Contents

1. System Under Test

The Apache Commons Numbers “provides implementations of number types and utilities” for those numbers. Some features include and are not limited to prime numbers, fractions, complex numbers, and gamma functions.

The methods that will be used to be tested are SortInPlace, CosAngle, and pow. SortInPlace, as stated in the official documentation, sorts the array given in the first argument, and performs the same reordering of entries in the other given arrays. In CosAngle, given two vectors, it calculates the cosine. In Pow, given two integers k and e, the method calculates k^e.

1.1. Details

Homepage: https://commons.apache.org/proper/commons-numbers/developers.html
GitHub: https://github.com/apache/commons-numbers

The project source code includes 13 packages and their corresponding class files:

- Commons Numbers Angles
  - Angle; LOC 311
  - CosAngle; LOC 44
  - Reduce; LOC 49
- Commons Numbers Arrays
  - MultidimensionalCounter; LOC 214
  - Enum File SortInPlace; LOC 138
- Commons Numbers Combinatorics
  - BinomialCoefficient; LOC 147
  - BinomialCoefficientDouble; LOC 125
  - Combinations; LOC 376
  - CombinatoricsException; LOC 46
  - Factorial; LOC 279
  - FactorialDouble; LOC 71
  - LogBinomialCoefficient; LOC 85
  - LogFactorial; LOC 119
- Commons Numbers Complex Streams
  - ComplexUtils; LOC 1624
- Commons Numbers Complex

    - Complex; LOC 3697
- Commons Numbers Core
    - Interface Addition; LOC 47
    - ArithmeticUtils; LOC 606
    - ExtendedPrecision; LOC 307
    - Interface Multiplication; LOC 47
    - Interface NativeOperators; LOC 59
    - Enum Norm; LOC 571
    - Sum; LOC 255
- Commons Numbers Examples
    - ComplexPerformance; LOC 655
    - SinCosPerformance; LOC 331
    - DoublePrecision; LOC 705
    - DoubleSplitPerformance; LOC 615
    - DoubleUtils; LOC 72
    - EuclideanNormAlgorithmPerformance; LOC 224
    - EuclideanNormAlgorithms; LOC 751
- Commons Numbers Field
    - AbstractField; LOC 69
    - BigFractionField; LOC 47
    - FP64; LOC 172
    - FP64Field; LOC 49
    - Interface Field; LOC 83
    - FractionField; LOC 47
- Commons Numbers Fraction
    - BigFraction; LOC 1292
    - ContinuedFraction; LOC 158
    - Fraction; LOC 910
    - FractionExceotion; LOC 55
    - GeneralizedContinuedFraction; LOC 460
- Commons Numbers Primes
    - Primes; LOC 115
    - SmallPrimes; LOC 289

Total LOC: 14,684

1.2. Software Architecture

The source code is structured generically in a way that each library is nested in its own folder. Fortunately, this software has exceptional documentation in which most of the functionalities can easily be accessed and read via the Javadoc that is provided in each library. When looking at each file and their documentation, the methods are arranged by their

categories: static, instance, or concrete methods. There are also not much global variables used in these methods. Most of the variables are local to their class and are mostly input variables, if not, variables extracted from Java libraries which is then used to develop their methods.

2. Test Environment

Project under test is deployed and hosted on GitHub as linked ***Details*** section. The operating system used to run tests is Windows 10, on IntelliJ IDE 2022.2.2, Maven, OpenJDK version 11, JUnit 5.8.1.

3. Input Space Partitioning

3.1. The Input Domain

Imports Needed for Tests:

```
import org.apache.commons.numbers.*;
```

SPECIFIC Imports If Needed:

```
import org.apache.commons.numbers.primes.*;
import org.apache.commons.numbers.angle.*;
import org.apache.commons.numbers.combinatorics.*;
```

Methods To Be Tested:

Numbers Arrays → SortInPlace:

- Parameters: user input (double[] x, …, double[] z)
- Number of user input parameters can be more than two
- Can test to ascend or descend the array… for this project we will use ascend

Numbers Angle → CosAngle:

- Parameters: user input (double[] v1, double[] v2)

Numbers Core → pow:

- Parameters: user input (int k, int e)

3.2. Characteristics and Blocks

Characteristic	B1	B2	B3
C1 = The order of the initial array	Ascending order	Descending order	No order
C2 = One or more of the arrays are different sizes	true	false	
C3 = The integrity of distinction of the input arrays	the same array is used as a parameter twice (x[], x[], y[])	all arrays are distinct (x[], y[], z[])	the same array is used for all parameters (x[], x[], x[])
C4 = v1 values = 0	true	false	
C5 = relation of array sizes	|v1| > |v2|	|v1| = |v2|	|v1| < |v2|
C6 = The integrity of the difference of each vector (for the available and corresponding entries)	v1 = v2	v1 ≠ v2	
C7 = k's relation to 0	K > 0	K = 0	K < 0
C8 = e's relation to 0	E > 0	E = 0	E < 0
C9 = the result of k^e would surpass 10 digits	true	false	

For the SortInPlace method, characteristics 1 and 2 are interface-based characteristics in which they are characteristics that can simply be distinguished by true or false. Characteristic 3 on the other hand is a functionality-based characteristic in which it focuses on how the method reacts when different sets of parameters are used as input. Characteristics 1 and 2 are disjoint as an array cannot be ascending and descending at the same time. Characteristic 3 is also disjoint in it of itself as not only does it include all 3 arrays but it also would not include nor does it base its output off of the other characteristic.

For the CosAngle method, characteristics 4 and 6 are interface-based characteristics as they are more concerned with the values of their corresponding variable or variables. Characteristic 5 on the other hand, looks at the functionality of how the cosine of the two vectors are calculated, therefore, looking more deeply at the sizes of the array, in this case the dimensions, of the vectors. Each of these characteristics satisfy the disjoint property that comes with partitioning as when looking at each characteristic, if one partition is true, then the other partitions cannot be true as well, which leads to also satisfying the completeness as each characteristic covers the entire domain for their respective aspect of the method.

For the power method, characteristic 7 and 8 are considered interface-based as they are both looking at the variables k and e respectively, and changing around the value of the variables. Characteristic 9 on the other hand would be considered functionality-based as it looks more at the result of going through the method rather than the actual variables themselves. Each of these partitions are disjoint in that they can never be in the same partition and be of any relation to each other, therefore also satisfying the completeness of the partitioning.

4. Test Requirements

4.1. Pair-Wise Coverage Criteria

Method under test = SortInPlace:

Partitions for characteristic 1 = characteristic A = {A, B, C}

Partitions for characteristic 2 = characteristic B = {1, 2}

Partitions for characteristic 3 = characteristic C = {X, Y, Z}

Method under test = from:

Partitions for characteristic 4 = characteristic D = {A, B}

Partitions for characteristic 5 = characteristic E = {X, Y, Z}

Partitions for characteristic 6 = characteristic F = {1, 2}

Method under test = pow:

Partitions for characteristic 7 = characteristic X = {A, B, C}

Partitions for characteristic 8 = characteristic Y = {1, 2, 3}

Partitions for characteristic 9 = characteristic Z = {T, F}

Method Under Test: SortInPlace

TR	A	B	C	Feasible?
1	A	1	X	Y
2	A	2	Y	Y
3	A	1	Z	N, cannot have one array that is different size from one another
4	B	2	X	Y
5	B	1	Y	Y
6	B	2	Z	Y
7	C	1	X	Y
8	C	2	Y	Y
9	C	1	Z	N, cannot have one array that is different size from one another

Module Under Test: CosAngle

TR	D	E	F	Feasible?
10	A	X	1	Y
11	B	X	2	Y
12	A	X	2	Y
13	B	Y	1	Y
14	A	Y	1	Y
15	B	Y	2	Y
16	A	Z	2	Y
17	B	Z	1	Y
18	A	Z	1	Y

Method Under Test: pow

TR	X	Y	Z	Feasible?
19	A	1	T	Y
20	A	2	F	Y
21	A	3	T	Y
22	B	1	F	Y
23	B	2	T	N, 0^0 will never exceed 10 digits
24	B	3	F	Y
25	C	1	T	Y
26	C	2	F	Y
27	C	3	F	Y

Test Results with Traceability

Please note that all input can be seen in the MainTest.java file. I just put the input variable names into the table to make it more readable as all of the inputs are composed of arrays of doubles which will make the tables very unreadable.

Test ID	Targeted TR	MUT	Input	Observed Output	Result
1	1	SortInPlace	(a1_ascending, a1_ascending, a2_size4)	IllegalArgumentException	Fail
2	2	SortInPlace	(a1_ascending, a2, a3)	All arrays are the same	Pass
3	4	SortInPlace	(a1_descending, a1_descending, a2)	a1 is ordered the same as it initially was, a2 is sorted differently than it initially was	Pass
4	5	SortInPlace	(a1_descending_5d, a2_2d, a3_2d)	IllegalArgumentException Size mismatch 2!=5	Fail
5	6	SortInPlace	(a1_descending_5d, a1_descending_5d, a1_descending_5d)	a1 is ordered in ascending order	Pass
6	7	SortInPlace	(a1_no_order_5d, a1_no_order_5d, a2_3d)	IllegalArgumentException Size Mismatch 3!=6	Fail
7	8	SortInPlace	(a1, a2, a3)	a1 is sorted in ascending order, a2 and a3 are sorted according to a1	Pass
8	10	CosAngle	(v1_2d, v2_3d)	IllegalArgumentException Dimension mismatch	Fail
9	11	CosAngle	(v1_2d, v2_3d)	IllegalArgumentException Dimension mismatch	Fail
10	12	CosAngle	(v1_2d, v2_3d)	IllegalArgumentException Dimension mismatch	Fail
11	13	CosAngle	(v1, v2)	1	Pass
12	14	CosAngle	(v1, v2)	NaN	Pass
13	15	CosAngle	(v1, v2)	0.3922578230875513	Pass
14	16	CosAngle	(v1_3d, v2_2d)	IllegalArgumentException Dimension mismatch	Fail
15	17	CosAngle	(v1_3d, v2_2d)	IllegalArgumentException Dimension mismatch	Fail

16	18	CosAngle	(v1_3d, v2_2d)	IllegalArgumentException Dimension mismatch	Fail
17	19	pow	(k=22, e=8)	ArithmeticException Integer Overflow	Fail
18	20	pow	(k=100, e=0)	1	Pass
19	21	pow	(k=20, e=-29)	IllegalArgumentException negative exponent	Fail
20	22	pow	(k=0, e=241)	0	Pass
21	24	pow	(k=0, e=-29)	IllegalArgumentException negative exponent	Fail
22	25	pow	(k=-29, e=14)	ArithmeticException Integer Overflow	Fail
23	26	pow	(k = -512, e = 0)	1	Pass
24	27	pow	(k=-2, e=21)	-2097152	Pass