Assignment 2 - Math.js

Daniel Carlsson - dc222bz@student.lnu.se

Jacob Andersson – ja224iy@student.lnu.se

A short description of the project.

What is its purpose?

- Math.js is a JavaScript library for performing mathematical operations.
- It provides a wide range of functions for algebra, analysis, combinatorics, geometry, and more.
- It is designed to be flexible and easy to use, allowing users to incorporate mathematical calculations into their JavaScript programs.
- Potential uses for math.js include scientific computing, da5ta analysis, and educational software.

What does it aim to accomplish?

- Provide a comprehensive set of mathematical functions and operations that can be easily used in JavaScript programs.
- Be flexible and intuitive, allowing developers to incorporate mathematical calculations into their programs with minimal effort.
- Make it easy for developers to perform complex mathematical calculations and operations in their programs.
- Provide a wide range of functions and operations to support a variety of applications and use cases.

Provide an overview of the requirements and specifications.

Requirements

Mathematical operations, arithmetic, algebra, trigonometry, calculus. Compatibility with different JavaScript environments (browser, NodeJS), easy to use, documentation explaining the use etc.

Stakeholders

Developers, users - The developers use the library in their development and the users rely on the fact that the library works when they are using the applications built by the developers.

Risks

Dependency risk:

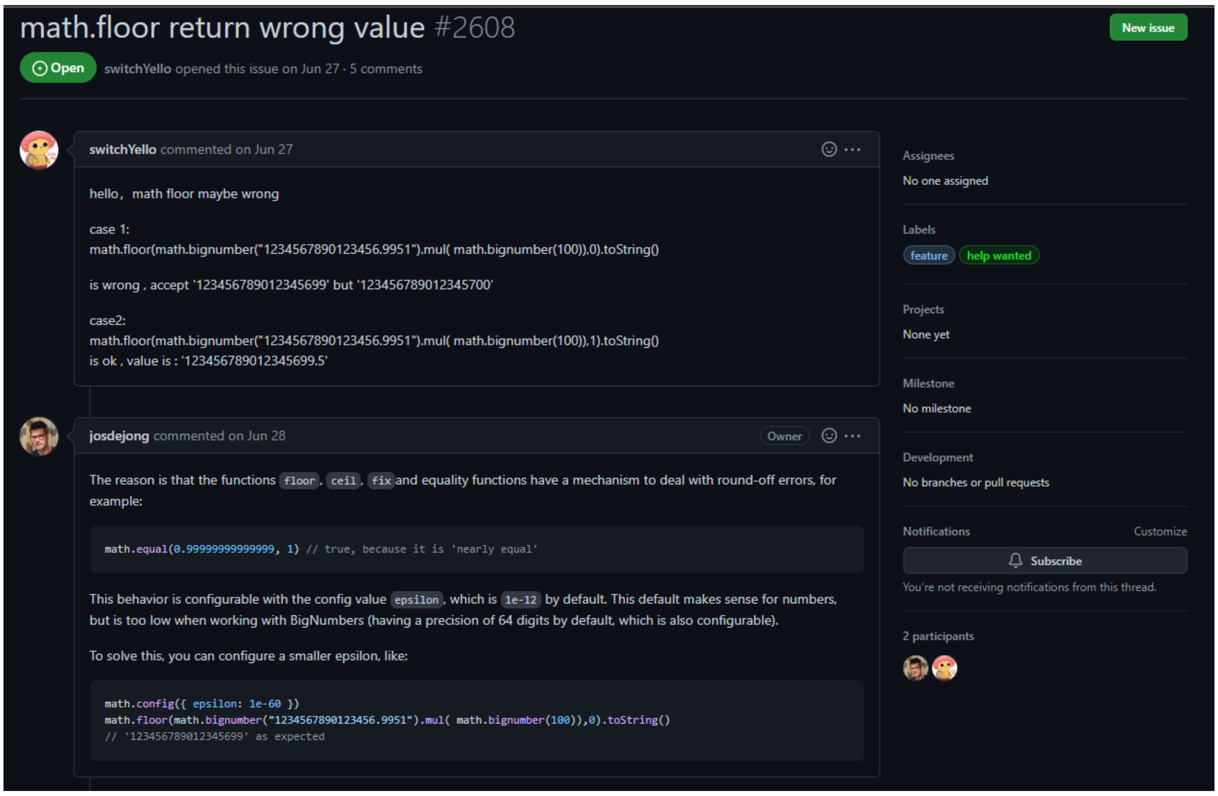
Math.js is a third-party library that you are relying on to perform certain calculations. If there is an issue with the library (e.g., a bug or security vulnerability), it could affect the reliability and accuracy of your calculations. To mitigate this risk, you should ensure that you are using a stable and wellmaintained version of Math.js and stay up to date with any updates or security patches that are released.

Performance risk:

Math.js can be computationally intensive, especially for large or complex calculations. This could impact the performance of your application, especially if you are running the calculations on a server with limited resources. To mitigate this risk, you should optimize your code and be mindful of the performance implications of the calculations you are performing.

Accuracy risk:

As with any software, there is always the possibility of errors or inaccuracies in the calculations performed by Math.js as shown in the bug report in figure 1 if not configured in the right way according to the situation.



math.floor return wrong value #2608

Open switchYello opened this issue on Jun 27 · 5 comments

switchYello commented on Jun 27

hello， math floor maybe wrong

case 1:
math.floor(math.bignumber("1234567890123456.9951").mul(math.bignumber(100)),0).toString()

is wrong , accept '123456789012345699' but '123456789012345700'

case2:
math.floor(math.bignumber("1234567890123456.9951").mul(math.bignumber(100)),1).toString()
is ok , value is : '123456789012345699.5'

josdejong commented on Jun 28 · Owner

The reason is that the functions `floor`, `ceil`, `fix` and equality functions have a mechanism to deal with round-off errors, for example:

```
math.equal(0.99999999999999, 1) // true, because it is 'nearly equal'
```

This behavior is configurable with the config value `epsilon`, which is `1e-12` by default. This default makes sense for numbers, but is too low when working with BigNumbers (having a precision of 64 digits by default, which is also configurable).

To solve this, you can configure a smaller epsilon, like:

```
math.config({ epsilon: 1e-60 })
math.floor(math.bignumber("1234567890123456.9951").mul( math.bignumber(100)),0).toString()
// '123456789012345699' as expected
```

Figure 1. Accuracy risk.

To mitigate this risk, you should carefully test your code and verify the accuracy of your results. You should also be aware of the limitations of the library and any potential sources of error that may affect your calculations.

Overall, while there are some risks associated with using Math.js, these risks can be mitigated by being aware of them and taking appropriate precautions. If you use the library responsibly and test your code thoroughly, Math.js can be a valuable tool for performing complex mathematical calculations in your projects.

Evaluation

Math.js is a widely used and well-respected library for performing mathematical calculations in JavaScript. It provides many functions and operations that are useful for a wide range of applications, and it is easy to use and flexible. Many developers find it to be a valuable tool for working with mathematics in JavaScript.

One of the strengths of Math.js is its wide range of functions and operations, which cover a wide range of mathematical concepts and allow developers to perform a variety of complex calculations. It also has support for complex numbers, matrices, and units, which can be useful in certain contexts.

Another advantage of Math.js is its ease of use. It has a simple, intuitive API that makes it easy for developers to get started with the library, and it is well-documented, with clear examples and explanations of how to use its various functions and operations.

Overall, Math.js is a powerful and useful tool for working with mathematics in JavaScript. It is widely used and well-regarded by developers, and it is a valuable resource for anyone looking to perform complex mathematical calculations in their projects.

The past, present, and future development of the project.

The past:

The development of the library started out in 2013 by Jos De Jong and has been actively worked on since then with several updates in the form of versions from 0.0.1 to 11.5.0 which indicates a constant improvement of the library. Most of the core functionality of the library was implemented in the early years but new features and functionality has been added throughout the years. A lot of features and functionality that has been implemented after the initial stage of the project has been implemented by the help of the open-source community. After 2015 the development of the project mainly consisted of bug fixes and improvements of already existing functionality, however new functionality is still added but not as frequently which is to be expected. As the project grows bigger there are also a lot of new developers contributing to the development as well as the variety of developers increases.

Current:

The current development is mostly done by the open-source community and consists of a lot of fixes and improvements of earlier implementations, there are some features that are planned to be developed. There are currently discussions about design choices of certain parts of the project which could change certain parts codebase. Documentation is also something that is constantly being updated, fixed, and added to the project to further improve the overall understanding of the project.

Future:

The future development of the math.js library will consist of more features and improvements to existing features and functionality such as support for more advanced mathematics. There are also possibilities of improving performance of certain parts of the library which is something that is currently discussed.

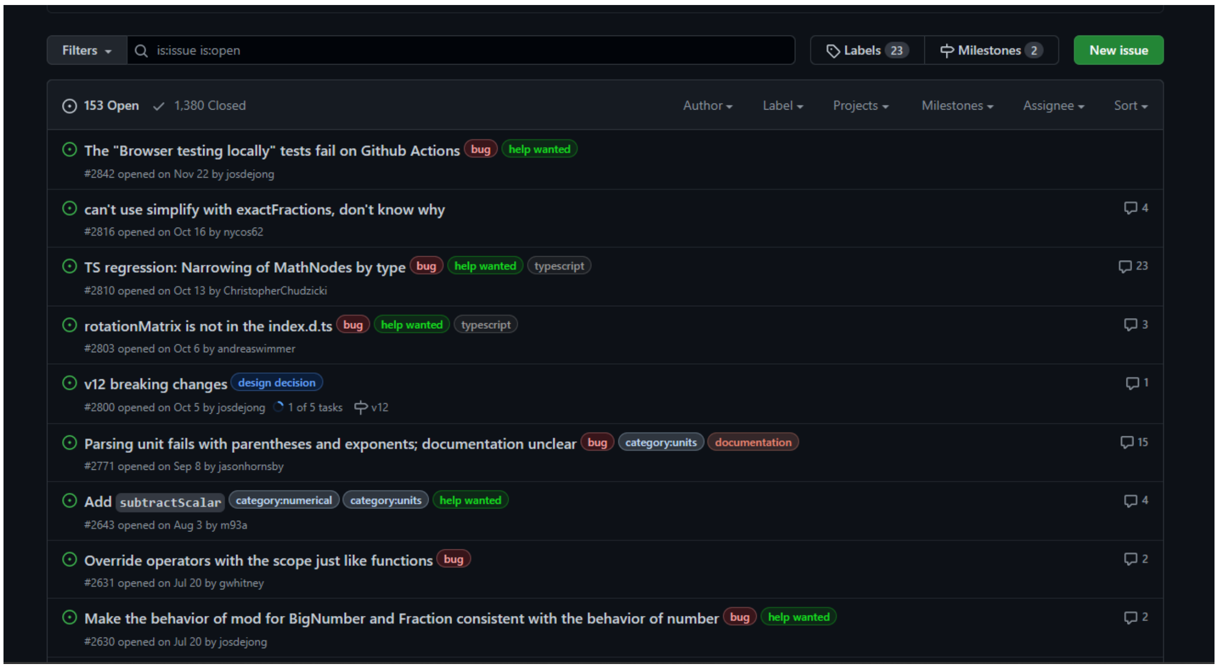
The current testing strategy, the kinds of tests being performed, and how the testing is reported on.

What tools are used?

The current testing strategy involves unit testing to test the individual functions as well as continuous integration to automate the testing of the codebase. The tool used for testing is Mocha, a JavaScript test library for Nodejs.

How do they handle bugs?

The developers of the Math.js library handle bugs by creating issues on GitHub to allow the community to fix these as presented in figure 2.



The "Browser testing locally" tests fail on Github Actions bug help wanted
#2842 opened on Nov 22 by josdejong

can't use simplify with exactFractions, don't know why — 4
#2816 opened on Oct 16 by nycos62

TS regression: Narrowing of MathNodes by type bug help wanted typescript — 23
#2810 opened on Oct 13 by ChristopherChudzicki

rotationMatrix is not in the index.d.ts bug help wanted typescript — 3
#2803 opened on Oct 6 by andreaswimmer

v12 breaking changes design decision — 1
#2800 opened on Oct 5 by josdejong 1 of 5 tasks v12

Parsing unit fails with parentheses and exponents; documentation unclear bug category:units documentation — 15
#2771 opened on Sep 8 by jasonhornsby

Add `subtractScalar` category:numerical category:units help wanted — 4
#2643 opened on Aug 3 by m93a

Override operators with the scope just like functions bug — 2
#2631 opened on Jul 20 by gwhitney

Make the behavior of mod for BigNumber and Fraction consistent with the behavior of number bug help wanted — 2
#2630 opened on Jul 20 by josdejong

Figure 2. Bug handling.

When a bug is found an issue is created and clarifying labels are added to the issues. These can include labels that indicate what kind of bug, what category and if help is wanted to solve the problem. This way of handling bugs makes the process much more efficient since the whole community can find bugs and create an issue as well as be part of the solution to the bug.

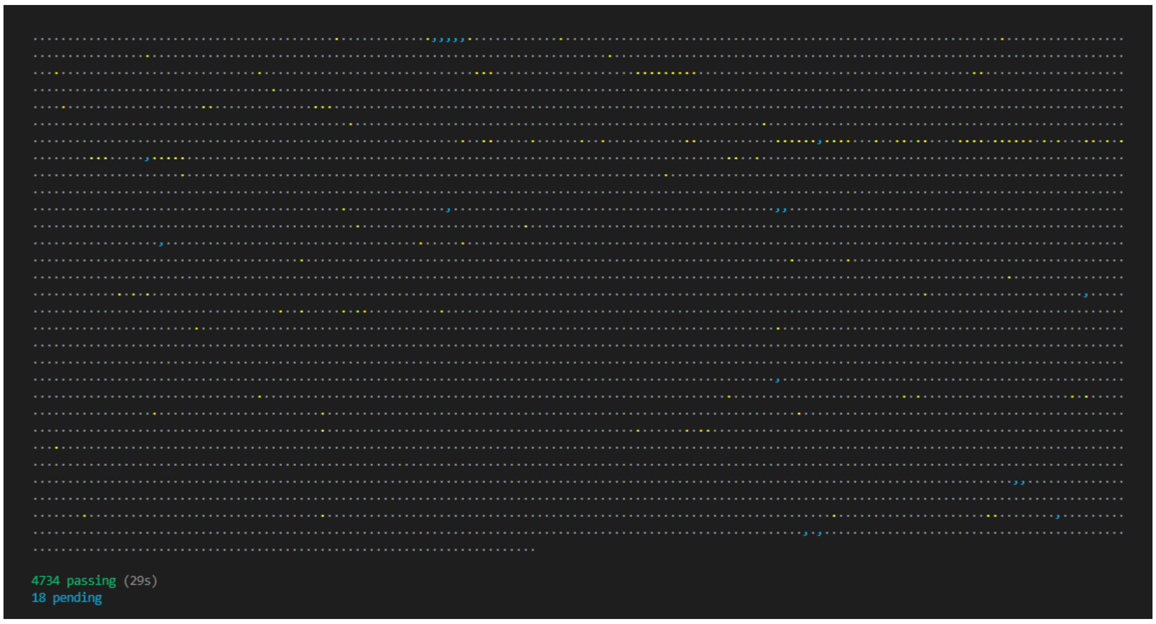
Document your testing performed, the pre-existing tests and their results.

Exploratory testing is optional, what kind of structured testing will you do?

We planned to run the unit tests and evaluate the result and run the coverage test and see what sort of result that will give us.

What did you do and what did you find?

The result from this was that the unit tests was that they had 4734 passing tests and 18 pending as seen in figure 3.



```
4734 passing (29s)
18 pending
```

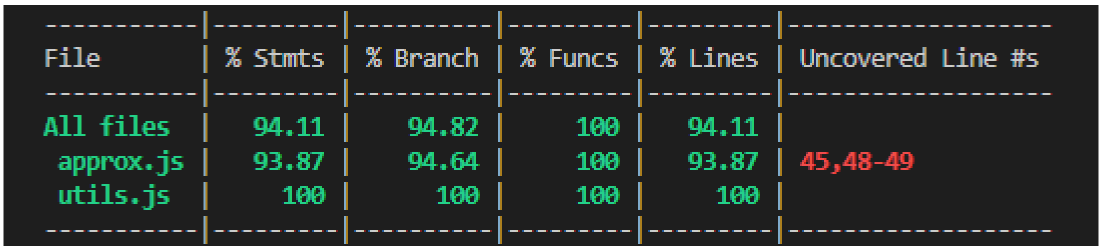
Figure 3. Unit Tests.

There are quite a few “yellow” tests which are shown in the picture of the test summary, yellow indicates that a test is slow which as mentioned earlier in the report shows the risk for performance issues of the operations that require more (memory)?

The “blue” tests are pending tests which there are 18 of currently, a pending test in mocha means that there are test cases without a call-back, these are not considered a failed test but a test that should be written or is being worked on.

From this data we concluded that they must have tested all functions in the library.

But when we began to run the coverage test, we ran in to a problem. The testing library that they used was not updated to handle ESM and didn’t work as intended (See figure 4) only two files was of the old format and could be tested.



```
-----------|---------|----------|---------|---------|-------------------
File       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------|---------|----------|---------|---------|-------------------
All files  |   94.11 |    94.82 |     100 |   94.11 |
 approx.js |   93.87 |    94.64 |     100 |   93.87 | 45,48-49
 utils.js  |     100 |      100 |     100 |     100 |
-----------|---------|----------|---------|---------|-------------------
```

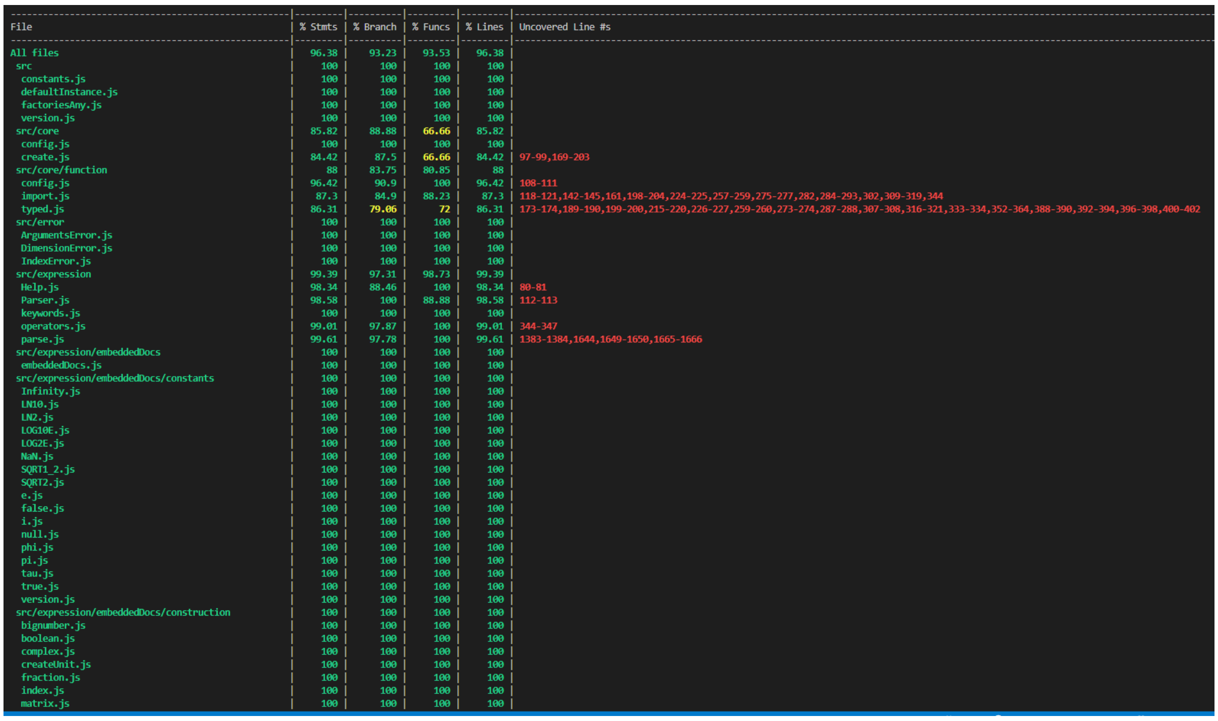
Figure 4. Coverage tests not working.

We emailed the owner of the library and got the following response:

Yeah, code coverage is broken, the nyc library doesn't work with ESM modules out of the box, so after refactoring mathjs from CommonJS to ESM, coverage didn't work anymore (except for two files that are still CommonJS). I don't really use coverage that much, so it hasn't been a prio to fix this yet. Help getting the coverage working again would be welcome (either implementing a workaround for nyc or replacing that with a different coverage library).

Have a nice day,
Jos

The solution to this problem was that we installed a testing library called c8 that specialises on coverage for ESM and now we could see all the coverage information over all the files, some of them shown in figure 5.

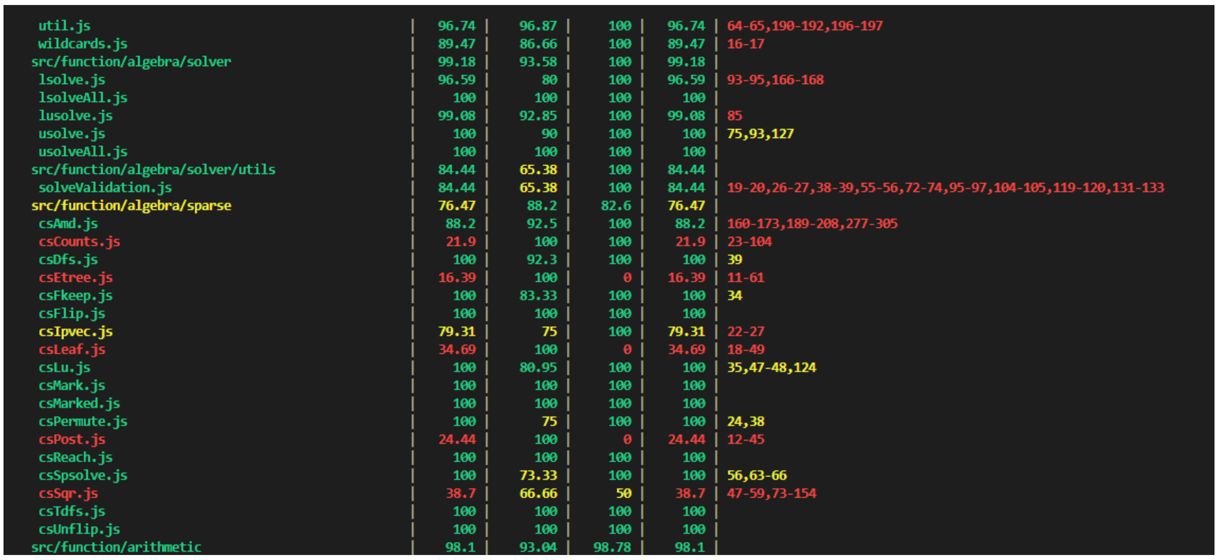


```
----------------------------------------------------|---------|----------|---------|---------|-------------------------------------------------------------------------------------------------------------------
File                                                | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------------------------------------|---------|----------|---------|---------|-------------------------------------------------------------------------------------------------------------------
All files                                           |   96.38 |    93.23 |   93.53 |   96.38 |
 src                                                |     100 |      100 |     100 |     100 |
  constants.js                                      |     100 |      100 |     100 |     100 |
  defaultInstance.js                                |     100 |      100 |     100 |     100 |
  factoriesAny.js                                   |     100 |      100 |     100 |     100 |
  version.js                                        |     100 |      100 |     100 |     100 |
 src/core                                           |   85.82 |    88.88 |   66.66 |   85.82 |
  config.js                                         |     100 |      100 |     100 |     100 |
  create.js                                         |   84.42 |     87.5 |   66.66 |   84.42 | 97-99,169-203
 src/core/function                                  |      88 |    83.75 |   80.85 |      88 |
  config.js                                         |   96.42 |     90.9 |     100 |   96.42 | 108-111
  import.js                                         |    87.3 |     84.9 |   88.23 |    87.3 | 118-121,142-145,161,198-204,224-225,257-259,275-277,282,284-293,302,309-319,344
  typed.js                                          |   86.31 |    79.06 |      72 |   86.31 | 173-174,189-190,199-200,215-220,226-227,259-260,273-274,287-288,307-308,316-321,333-334,352-364,388-390,392-394,396-398,400-402
 src/error                                          |     100 |      100 |     100 |     100 |
  ArgumentsError.js                                 |     100 |      100 |     100 |     100 |
  DimensionError.js                                 |     100 |      100 |     100 |     100 |
  IndexError.js                                     |     100 |      100 |     100 |     100 |
 src/expression                                     |   99.39 |    97.31 |   98.73 |   99.39 |
  Help.js                                           |   98.34 |    88.46 |     100 |   98.34 | 80-81
  Parser.js                                         |   98.58 |      100 |   88.88 |   98.58 | 112-113
  keywords.js                                       |     100 |      100 |     100 |     100 |
  operators.js                                      |   99.01 |    97.87 |     100 |   99.01 | 344-347
  parse.js                                          |   99.61 |    97.78 |     100 |   99.61 | 1383-1384,1644,1649-1650,1665-1666
 src/expression/embeddedDocs                        |     100 |      100 |     100 |     100 |
  embeddedDocs.js                                   |     100 |      100 |     100 |     100 |
 src/expression/embeddedDocs/constants              |     100 |      100 |     100 |     100 |
  Infinity.js                                       |     100 |      100 |     100 |     100 |
  LN10.js                                           |     100 |      100 |     100 |     100 |
  LN2.js                                            |     100 |      100 |     100 |     100 |
  LOG10E.js                                         |     100 |      100 |     100 |     100 |
  LOG2E.js                                          |     100 |      100 |     100 |     100 |
  NaN.js                                            |     100 |      100 |     100 |     100 |
  SQRT1_2.js                                        |     100 |      100 |     100 |     100 |
  SQRT2.js                                          |     100 |      100 |     100 |     100 |
  e.js                                              |     100 |      100 |     100 |     100 |
  false.js                                          |     100 |      100 |     100 |     100 |
  i.js                                              |     100 |      100 |     100 |     100 |
  null.js                                           |     100 |      100 |     100 |     100 |
  phi.js                                            |     100 |      100 |     100 |     100 |
  pi.js                                             |     100 |      100 |     100 |     100 |
  tau.js                                            |     100 |      100 |     100 |     100 |
  true.js                                           |     100 |      100 |     100 |     100 |
  version.js                                        |     100 |      100 |     100 |     100 |
 src/expression/embeddedDocs/construction           |     100 |      100 |     100 |     100 |
  bignumber.js                                      |     100 |      100 |     100 |     100 |
  boolean.js                                        |     100 |      100 |     100 |     100 |
  complex.js                                        |     100 |      100 |     100 |     100 |
  createUnit.js                                     |     100 |      100 |     100 |     100 |
  fraction.js                                       |     100 |      100 |     100 |     100 |
  index.js                                          |     100 |      100 |     100 |     100 |
  matrix.js                                         |     100 |      100 |     100 |     100 |
```

Figure 5. Coverage test working.

From this coverage report we can see which lines of the files tested in the unit tests that are not run during the test, but it can not say if the tests are good or not just that they are covered (gold sprayed turd).

As we understand coverage can best be used for finding test cases that need to be implemented as seen in figure 6, csEtree.js, csLeaf.js, csPost.js have zero functions invoked by the unit tests but some lines in theses are ran inside other test cases which gives them some coverage.



```
   util.js                                  |   96.74 |   96.87 |     100 |   96.74 | 64-65,190-192,196-197
   wildcards.js                             |   89.47 |   86.66 |     100 |   89.47 | 16-17
  src/function/algebra/solver               |   99.18 |   93.58 |     100 |   99.18 |
   lsolve.js                                |   96.59 |      80 |     100 |   96.59 | 93-95,166-168
   lsolveAll.js                             |     100 |     100 |     100 |     100 |
   lusolve.js                               |   99.08 |   92.85 |     100 |   99.08 | 85
   usolve.js                                |     100 |      90 |     100 |     100 | 75,93,127
   usolveAll.js                             |     100 |     100 |     100 |     100 |
  src/function/algebra/solver/utils         |   84.44 |   65.38 |     100 |   84.44 |
   solveValidation.js                       |   84.44 |   65.38 |     100 |   84.44 | 19-20,26-27,38-39,55-56,72-74,95-97,104-105,119-120,131-133
  src/function/algebra/sparse               |   76.47 |    88.2 |    82.6 |   76.47 |
   csAmd.js                                 |    88.2 |    92.5 |     100 |    88.2 | 160-173,189-208,277-305
   csCounts.js                              |    21.9 |     100 |     100 |    21.9 | 23-104
   csDfs.js                                 |     100 |    92.3 |     100 |     100 | 39
   csEtree.js                               |   16.39 |     100 |       0 |   16.39 | 11-61
   csFkeep.js                               |     100 |   83.33 |     100 |     100 | 34
   csFlip.js                                |     100 |     100 |     100 |     100 |
   csIpvec.js                               |   79.31 |      75 |     100 |   79.31 | 22-27
   csLeaf.js                                |   34.69 |     100 |       0 |   34.69 | 18-49
   csLu.js                                  |     100 |   80.95 |     100 |     100 | 35,47-48,124
   csMark.js                                |     100 |     100 |     100 |     100 |
   csMarked.js                              |     100 |     100 |     100 |     100 |
   csPermute.js                             |     100 |      75 |     100 |     100 | 24,38
   csPost.js                                |   24.44 |     100 |       0 |   24.44 | 12-45
   csReach.js                               |     100 |     100 |     100 |     100 |
   csSpsolve.js                             |     100 |   73.33 |     100 |     100 | 56,63-66
   csSqr.js                                 |    38.7 |   66.66 |      50 |    38.7 | 47-59,73-154
   csTdfs.js                                |     100 |     100 |     100 |     100 |
   csUnflip.js                              |     100 |     100 |     100 |     100 |
  src/function/arithmetic                   |    98.1 |   93.04 |   98.78 |    98.1 |
```

Figure 6. Coverage Result.

These numbers can be interested in future testing especially the last column total lines cover in the file. These red files have a lot of if/else statements which make them harder to test and can be why they have lower coverage than others.

To summarize what coverage can be used for is as an indicator of how much lines are reached by the test so the developer can manually check if the unreached code is unreachable code or not and maybe test them separately.