# Dynamics Lab Verification Audit

There are two types of testing in Unity's testing framework: Edit Mode testing and Play Mode testing. Edit Mode testing can be done inside the editor, whereas Unity must build and run the project in order to run Play Mode tests. In this way, software functionality at a high level must be tested using Play Mode tests that can run a single test over multiple update frames.

We have learned how to use Edit Mode automated testing, but due to either our misunderstanding of how to write Play Mode tests, or confusing documentation, or both, we believe we have been unsuccessful in getting Play Mode testing to work. This limits the range of features we can appropriately test to those which do not depend on frame update cycles. We are continuing to try to get Play Mode testing working.

There is no built-in coverage tool in Unity, and it is very difficult to get any third party tools working in a Unity environment, as apart from specifically supported tools, third party tools in general are unsupported by Unity. While it is apparently possible to hack Unity into working with some third party tools, this is something we've already tried and failed at multiple times in this project. We are still attempting to make any coverage tool work with Unity on any single computer, but as it stands we don't expect to be able to produce coverage information at all in this project.

Due to the nature of our project as a numerical solver for dynamical systems, we need and have implemented other types of testing besides unit testing. Most importantly, we implement numerical accuracy performance testing to ensure that our software produces accurate data.

In this document we recommend other performance testing, such as time performance testing and stress testing. In order to overcome the issues with Play Mode testing, we could also use non-automated functional testing with a given test procedure and expected outcome, but this would be a temporary solution that would drastically raise time requirements for testing and make regression testing impossible.

Because our software has a layered architecture, we use bottom-up integration testing, and so we have started with the lowest layer of our architecture, the math library.

## Verification Progress by Module

Legend

🟩 Well-tested feature

Minimally-tested feature
Partially tested feature
Untested feature

For each class, the table will specify the number of unit tests written and whether they cover the happy path with expected output given expected input and whether they cover any possible sad paths in handling unexpected input, where such behavior is defined.

## Math Library: B83.ExpressionParser

Partially tested
This module is not our code, and so in the interest of time, we have held off on testing it explicitly for now. However, other tests in the math library will cover its basic functionality. The lowest-level class which implicitly tests this namespace is FunctionVectorND.

- Basic expression evaluation - Implicitly tested by other tests
- Delegate evaluation - Implicitly tested by other tests
- Defined constants - Untested

## Math Library: DynamicsLab.Vector

### VectorND

Minimally tested
Unit tests are written for happy paths and some sad paths.

| **VectorND Unit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| VectorND(byte dim_in) | 1 | Yes | No |
| VectorND(params double[] list) | 2 | Yes | No |
| VectorND(VectorND tocopy) | 1 | Yes | No |
| Byte GetDim() | 1 | Yes | N/A |
| double this[byte i] | 1 | Yes | No |
| VectorND operator +(VectorND a, VectorND b) | 2 | Yes | Part |
| VectorND operator -(VectorND a, VectorND b) | 2 | Yes | Part |
| VectorND operator *(VectorND a, VectorND b) | 2 | Yes | Part |
| VectorND operator *(double k, VectorND b) | 1 | Yes | N/A |
| VectorND operator *(VectorND a, double k) | 1 | Yes | N/A |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|

| | | |
|---|---|---|
| Unit tests | 14 | 20 |

Consideration of other types of testing for VectorND
Unit testing should be sufficient for VectorND. While numeric accuracy is a concern, it is included by default in all of the existing tests, since it is needed for double precision comparison anyway. We use a benchmark of 1e-12 accuracy.

Notes on VectorND unit testing:

- The biggest sad path for most VectorND operations is trying to perform math on vectors of different dimensionalities. This is partly tested, but only in one direction -- the first argument being higher dimensionality than the second. Rigorous testing of all sad paths would require vice versa.
- Another sad path for constructors would be trying to set dimensionality 0 or, for the params double constructor, trying to set dimensionality above 255.

Currently failing tests

- Copy constructor test is failing because we forgot to implement it.

## FunctionVectorND

Minimally tested
Happy paths are written for all functions.

| **VectorND Unit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| FunctionVectorND(byte dim_in) | 0* | Yes | No |
| byte GetDim() | 1 | Yes | N/A |
| Void Eval(params double[] list) | 2 | Yes | No |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit testing | 3 | 6 |

Consideration of other types of testing for FunctionVectorND
One thing that might be of interest here is numerical accuracy stress testing. Certain expressions could stress the limits of evaluation accuracy, such as dividing by very small numbers. Devising a way to test the limits of this could improve the value of our testing regime.

Notes on FunctionVectorND unit testing:

- The constructor is tested against GetDim()
- The only sad path that needs to be considered in testing Eval specifically is calling Eval when not all dimensions are allocated with a function. Currently, this has undefined behavior and just results in a null reference exception.
- A sad path that might need tests for all functions is initializing to zero dimensionality.

Currently failing tests

- None

## Math Library: DynamicsLab.Data

### DataChunk

N/A
This class is treated as an internal class to DataSet. It is not used outside of DataSet and has no functions outside of the constructor. We consider its testing to be included in DataSet testing.

### DataSet

Minimally tested
This class is also an internal class to the assembly, but was temporarily made public for testing. This is a problem for regression testing, as once it is made internal again the tests will become compiler errors. We do not know of a good way to fix this problem. For this class, many unit tests are implemented as white box tests in order to have defined expectations to test against outside the context of its usage by the Solver classes.

Most sad paths have undefined behavior. For instance, ChunkIndex() is not designed such that a user of this class can pass in negative values. Since behavior is undefined, tests cannot be written for it. This signals that our code is not robust.

| **DataSet Unit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| DataSet(byte dim_in, double tIntercept_in, double tSlope_in) | 1 | Yes | No |
| VectorND this[double t] | 4 | Yes | Part |
| VectorND this[int i] | 2 | Yes | No |
| VectorND this[int chunk, int local] | 2 | Yes | No |
| Int Lookup(double t) | 4 | Yes | Part |
| int ChunkIndex(int i) | 4 | Yes | No |
| Int LocalIndex(int i) | 4 | Yes | No |
| double Tval(int i) | 2 | Yes | No |
| Void WriteState(int chunk, int local, VectorND newState) | 2 | Yes | No |
| Void WriteNext(VectorND val) | 3 | Yes | No |
| void NewChunk() | 2 | Yes | N/A |
| Void RemoveFirstChunk() | 6 | Yes | Part |
| void RemoveLastChunk() | 3 | Yes | Part |
| Void SaveData(StreamWriter fout) | 3 | Yes | No |

| **DataSet Unit Testing : Other** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| Tracks data lower bound for first chunk removal | 2 | Yes | Part |
| Tracks data upper bound for new data points added | 2 | Yes | Part |
| Tracks t intercept for first chunk removal | 2 | Yes | N/A |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit testing | 48 | 75 |

Consideration of other types of testing for DataSet

Unit testing is sufficient for DataSet, as all of its functions are defined as individual actions that have a defined behavior on the happy path.

Notes on DataSet unit testing:

- Testing SaveData should read values as doubles rather than comparing strings, for generalized testing that gets as close to black box as possible. The way the test is currently written depends heavily on the file format.

Currently failing tests

- WriteNext fails a test in which it is called before WriteState.
- SaveData fails an edge case test which tests saving the last value in a data set
- RemoveFirstChunk and RemoveLastChunk should not remove when there is only one chunk, but are failing these tests because this functionality is not implemented.
- RemoveFirstChunk is failing to update minIndex correctly

## Math Library: DynamicsLab.Solvers

### ODEIVPSolver

Minimally tested

Like DataSet, ODEIVPSolver is an internal class. However, unlike DataSet, it is a small class with only one function apart from its constructor. It is used as a helper class to ODEInitialValueProblem. As an internal class, it cannot be used outside of its assembly, but we have made it temporarily public in order to test it.

| **ODEIVPSolver Unit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| ODEIVPSolver(byte dim_in) | 0 | No | No |
| VectorND RungeKutta(double t, double h, VectorND v, functionVectorND f) | 1 | Yes | No |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|

| | | |
|---|---|---|
| Unit testing | 1 | 7 |
| Time performance testing | 0 | 4 |

Consideration of other types of testing for ODEIVPSolver

- Runge-Kutta method of order 4 isn't a trivial algorithm--for each new n-dimensional point of data generated, it must evaluate an arbitrary function 4*n times. For this reason, time peformance testing could be of interest.
- More informative accuracy performance testing is not needed. Runge Kutta method of order 4 is a known algorithm, so as long as it is working within reasonable limits (the unit test written specifies a threshold of 1e-12) there should be no need to test its numerical accuracy performance directly.

Notes on ODEIVPSolver unit testing:

- The happy path is when ODEIVPSolver, the passed FunctionVectorND, and the passed VectorND to RungeKutta all have the same dimensionality. Therefore the sad paths would be when each one is more or less than the others, which would require 6 additional unit tests. At present, none of these occurences have defined behavior.
- With the current design of the class, there is no way to test the constructor directly since there is nothing in its public interface with which it can be checked. We've decided to still classify this module as minimally tested because the only thing the constructor should do is set the dimensionality, and if the dimensionality is wrong then the RungeKutta test would likely have failed, so the constructor is tested implicitly.

Currently failing tests

- None

## ODEInitialValueProblem

Minimally Tested

This is our primary solver class, so in addition to unit testing, numerical accuracy performance testing is also vital, and so is included in our testing regime. Within unit testing, most functions test both happy and sad paths, and so altogether this class is closer to well-tested than most of our other classes. While the sad paths are tested, they are not implemented so most sad path tests are currently failing. It currently passes all happy path unit tests. It is failing one of the two numerical accuracy performance benchmark tests.

This class is mostly well-tested except for its saving data to file feature.

| **ODEInitialValueProblem Unit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| ODEInitialValueProblem(byte dim_in, double h, double tIntercept) | 5 | Yes | Yes |
| byte GetDim() | 0* | Yes | N/A |

| | | | |
|---|---|---|---|
| double GetH() | 0* | Yes | N/A |
| double GetDataLowerBound() | 3 | Yes | Yes |
| double GetDataUpperBound() | 1 | Yes | N/A |
| double SolutionData(byte i, double t) | 5 | Yes | Yes |
| void SetH(double h_in, double tIntercept) | 3 | Yes | Yes |
| bool InvalidateData(double currentTime) | 3 | Yes | Yes |
| void SolveTo(double time) | 2* | Yes | Yes |
| void SetState(VectorND newState, double currentTime) | 8 | Yes | Yes |
| VectorND GetState(double currentTime) | 5 | Yes | Yes |
| void SaveData(StreamWriter fout) | 3 | Yes | No |

Notes on ODEInitialValueProblem unit testing:

- Almost all of the tests have to use SolveTo, so SolveTo is implicitly tested throughout the testing regime. This table indicates 2 tests which test its functionality specifically.
- GetH is tested against SetH
- GetDim is tested against the constructor

| **Accuracy Performance Tests** | **Checked on** | **Total Error** | **Passes Bench** |
|---|---|---|---|
| 1D nonlinear nonhomogeneous autonomous Initial condition (-2) at t=0 | [1,20] every 1 (20 values) | 1.6e-9 | Yes |
| 3D linear nonhomogeneous non-autonomous Initial condition (3, -1, 1) at t=0 | [0.1,1] every 0.1 (30 values) | 0.002 | No |

Notes on ODEInitialValueProblem accuracy performance testing:

- Benchmark used was error less than 1e-6 times the number of values.
- This is the 3D problem that fails to pass benchmark:

$$u_1' = u_1 + 2u_2 - 2u_3 + e^{-t}, \quad u_1(0) = 3;$$
$$u_2' = u_2 + u_3 - 2e^{-t}, \quad u_2(0) = -1;$$
$$u_3' = u_1 + 2u_2 + e^{-t}, \quad u_3(0) = 1; \quad 0 \le t \le 1; \quad h = 0.1;$$
$$\text{actual solutions } u_1(t) = -3e^{-t} - 3\sin t + 6\cos t, \quad u_2(t) = \frac{3}{2}e^{-t} + \frac{3}{10}\sin t - \frac{21}{10}\cos t - \frac{2}{5}e^{2t},$$
$$\text{and } u_3(t) = -e^{-t} + \frac{12}{5}\cos t + \frac{9}{5}\sin t - \frac{2}{5}e^{2t}.$$

The numerical solution converges to zero error as h->0 (h is data resolution, the interval between data points), which is a good indication that the actual solution, the written test, and the functionality of ODEInitialValueProblem are all correct. It may be that the benchmark is simply inappropriate for this problem. Analytic investigation (of the math) is necessary to determine if this is the case.

- Barring such issues, one key difference between these tests is the number of math function calls. The 1D test case is purely polynomial and has 1 exponential in its solution, whereas the 3D test case uses the exponential function thrice in the problem and sine, cosine, and exponential functions 11 times in its solution.

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit tests | 38 | 40 |
| Accuracy performance tests | 2 | 4 |
| Time performance tests | 0 | 2 |

Consideration of other types of testing for ODEInitialValueProblem
Since numerical solving of differential systems is a non-trivial task, time performance tests would also be of interest.

Currently failing tests
- 3D accuracy performance test, as mentioned above.
- 2 out-of-bounds sad path tests are failing for GetState
- 2 out-of-bounds sad path tests are failing for SolutionData
- 2 out-of-bounds sad path tests are failing for InvalidateData
- SaveData does not save the last valid data point
- SetH and the constructor can set h to zero or negative, and the constructor can set the dimensionality to zero, which should not be allowed.

## SpringMassSystem

Minimally tested
SpringMassSystem has ample unit tests for all of its functions, but could use a numerical accuracy performance test.

| **SpringMassSystemUnit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| double Stiffness {get; set} | 2 | Yes | N/A |
| double Mass {get; set} | 3 | Yes | Yes |
| double Damping {get; set} | 2 | Yes | N/A |
| double ForwardSolveTime {get ; set} | 5 | Yes | Yes |
| SpringMassSystem(double h_in, double stiffness_in, double mass_in, double damping_in, double currentTime_in, double forwardSolveTime_in) | 1* | Yes | Part |
| void Update(double newCurrentTime) | 1* | Yes | Yes |
| double Position(double t) | 4 | Yes | Yes |
| double Velocity(double t) | 4 | Yes | Yes |

| | | | |
|---|---|---|---|
| SetInitialCondition(double t, double pos, double vel) | 1* | Yes | Yes |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit tests | 23 | 26 |
| Accuracy performance tests | 0 | 1 |

Consideration of other types of testing for SpringMassSystem

Although SpringMassSystem is a wrapper for ODEInitialValueProblem, and ODEInitialValueProblem has its own numerical accuracy performance tests, it would still be good to verify that SpringMassSystem specifically works. This would ensure, for example, that the equation for the system to solve is not written wrong.

Notes on SpringMassSystem unit testing:

- The constructor is tested together with the properties (Stiffness, Mass, Damping, ForwardSolveTime)
- SetInitialCondition and Update are tested together with Position and Velocity, and must be called in almost every test anyway. Listed numbers are specific sad path testing.

Currently failing tests

- Reading Position or Velocity out of bounds
- Setting mass to zero should not be allowed either using the property or the constructor, but currently fails the tests for this behavior.
- ForwardSolveTime can be set to a negative value

## Unity: Spring-Mass

Partially Tested

We had some success in testing spring mass because it doesn't run in real time and has no dependence on the time between update frames, so we can call Update in Edit Mode tests with predictable results. That said, CreateLines.cs is largely a single very complex function, which makes testing difficult.

| **SpringMass.cs & Lines.cs Testing : Functions** | **Num*** | **Happy** | **Sad** |
|---|---|---|---|
| Void InitializeSimulation() | 2 | Yes | Part |
| Void onGUI() | 1* | Yes | N/A |
| Void Update() | 13 | Part | Part |
| Void ToggleVRView() | 1* | Yes | N/A |
| void MenuButton_OnPress() | 1* | Yes | N/A |
| Void InitializeCameras() | 0 | No | N/A |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit tests | 12* | 40 |

Notes on SpringMass testing:

- For this table only, the numbers listed reference the number of times the function was called by the testing regime, not the number of tests written.
- Some tests simply test whether the code runs without exceptions. We do not believe this constitutes testing these functions well. In the table above, these refer to the entries:
    - onGUI
    - ToggleVRView
    - MenuButton_OnPress()
- The CreateLines.cs and Lines.cs script functions (which make up the large majority of higher-level Spring-Mass code) have proven difficult to test due to their reliance on the very fine and low-level SpringMassSystem code.

Currently Failing test notes:

- Several Update() function if-statement branches are difficult to hit. Many tests must regress from previous tests in order to test a particular outcome of a complex function. However, many failing test fortunately do not raise any exception errors.

## Unity: 3D Motion

### PositionTextRenderer (Monobehavior)

Untested

We're presently unable to test this because it requires Play Mode testing.

| **PositionTextRenderer Unit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| Follows {set} | 0 | No | No |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit testing | 0 | 3 |

Consideration of other types of testing for PositionTextRenderer

Barring getting Play Mode testing to work, we could do non-automated functional testing with a given test procedure and expected outcome.

### Motion3DPlayerController (Monobehavior)

Untested

We're presently unable to test this because it requires Play Mode testing. Further complicating this in particular is the fact that it works by keyboard input polling, and implementing the user input event system into the testing framework is an added level of complexity.

| **Motion3DPlayerController Unit Testing** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| Up | 0 | No | No |
| Down | 0 | No | No |
| Left | 0 | No | No |
| Right | 0 | No | No |
| Stop | 0 | No | No |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit testing | 0 | 10 |

Consideration of other types of testing for Motion3DPlayerController
Barring getting Play Mode testing to work, we could do non-automated functional testing with a given test procedure and expected outcome.

## Motion3DSetup

Partially tested
We have made some progress in testing this, but more needs to be done.

| **Motion3DSetup Unit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| Constructor(string expressionX_in, string expressionY_in string expressionZ_in, Dictionary<string, double> params_in, int order_in | 5 | Yes | N/A |
| ChangeParameter(string parameter, double value ); | 1* | No | Yes |
| GetParameterValue(string parameter); | 0* | No | No |
| Implicit parse testing using read-only Property FunctionVectorND F | 0 | No | No |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit testing | 6 | 12 |

Consideration of other types of testing for Motion3DSetup
Unit testing should be sufficient for Motion3DSetup

Motion3DSetup Notes:

- The public interface for this class is very small, and so the only good way to test much of its functionality is implicitly by reading the FunctionVectorND and having it evaluate expressions.

## Motion3DConfigGui (Monobehavior)

Untested

While this is a monobehavior and it does use Update() cycles, it should be able to be tested in Edit Mode. We have not gotten to testing this yet.

| **Motion3DConfigGui Unit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| SeparateCSV | 0 | No | No |
| ValidateExpression | 0 | No | No |
| UpdateStatusDisplay | 0 | No | No |
| CheckInvalidFlags | 0 | No | No |
| exportForSetup | 0 | No | No |
| Populate Fields | 0 | No | No |
| Save_Equation | 0 | No | No |
| Load_Equation | 0 | No | No |
| Overwrite_Equations * | 0 | No | No |
| Update() | 0 | No | No |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit testing | 0 | 35 |

Consideration of other types of testing for Motion3DConfigGui

Unit testing should be sufficient for Motion3DConfigGui.

Notes on Motion3DConfigGui unit testing:

- Overwrite_Equations is dead code.

## Motion3DProblemIOManager

Partially tested

This is possible to be tested with our current knowledge, but we have not gotten to testing it.

| **Motion3DProblemIOManager Unit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| Load | 5 | Yes | No |
| Save | 0 | No | No |
| Apply | 0 | No | No |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit testing | 5 | 30 |

Consideration of other types of testing for Motion3DProblemIOManager
Unit testing should be sufficient for this module.

Notes on Motion3DProblemIOManager unit testing:

- This class handles the actual file I/O for saved problem configurations, so a lot of sad paths need to be considered ideally for platform issues, read/write access, file format issues, line endings, etc.
- Save is not yet implemented

## Motion3DObject (Monobehavior)

Untested
Most of Motion3DObject could probably be tested in Edit Mode, simply testing against its transform position, rendering characteristics, etc. This would not be a complete test, since what ultimately this class needs to achieve is “does an object render in the correct spot at the correct time”, which is difficult to test directly in any automated way. Nonetheless, there are a number of tests that can be written for its functions that don’t require a live exercise. No tests are implemented for this class yet.

| **Motion3DObject Unit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| SetState | 0 | No | No |
| ResetTrail * | 0 | No | No |
| SetAs1stOrder | 0 | No | No |
| SetAs2ndOrder | 0 | No | No |
| UpdateObjectPosition | 0 | No | No |
| GetCurrentState | 0 | No | No |
| Highlight | 0 | No | No |
| Dehighlight | 0 | No | No |
| Start() | 0 | No | No |
| Update() | 0 | No | No |
| CurrentTime {get; set} | 0 | No | No |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit tests | 0 | 30 |
| Accuracy performance tests | 0 | 3 |

Notes on Motion3DObject unit testing:

- ResetTrail can only be tested in Play Mode

- It may be possible to test everything else in Edit Mode, at least to a degree, but we simply haven't gotten to testing this yet.

<u>Consideration of other types of testing for Motion3DObject</u>

- Barring getting Play Mode testing to work, we could do non-automated functional testing with a given test procedure and expected outcome.
- Since Motion3DObject is also a wrapper for ODEInitialValueProblem, it would benefit from a numerical accuracy performance test. While ODEInitialValueProblem has its own accuracy performance tests, having an additional test here would ensure that the user-defined equations of motion are appropriately abstracted to a 3D motion problem.

## ObjectTrailRenderer (Monobehavior)

Untested

We're presently unable to test this because it requires Play Mode testing.

| **ObjectTrailRenderer Unit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| ResetTrail | 0 | No | No |
| Start() | 0 | No | No |
| Update() | 0 | No | No |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit testing | 0 | 8 |

<u>Consideration of other types of testing for ObjectTrailRenderer</u>

Barring getting Play Mode testing to work, we could do non-automated functional testing with a given test procedure and expected outcome.

## Motion3DSimulationController (Monobehavior)

Untested

We're presently unable to test this because this, more than anything else, requires Play Mode testing as its major functionality lies in its Update function which relies on update frames. Some things could be tested in Edit Mode, but we have not gotten to testing this yet.

| **Motion3DSimulationController Unit Testing** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| ForwardSolveTime {get} | 0 | No | No |
| ForwardSolveTime {set} | 0 | No | No |
| GetMotion3DSetup {get} | 0 | No | No |
| CurrentTime {get} | 0 | No | No |
| CurrentTime {set} | 0 | No | No |

| | | | |
|---|---|---|---|
| AllObjectsDataLowerBound {get} | 0 | No | No |
| AllObjectsDataLowerBound {set} | 0 | No | No |
| AllObjectsDataUpperBound {get} | 0 | No | No |
| AllObjectsDataUpperBound {set} | 0 | No | No |
| Start() | 0 | No | No |
| Update() | 0 | No | No |
| ApplyConfiguration | 0 | No | No |
| ChangeParameter | 0 | No | No |
| SetStateOnCurrentObject | 0 | No | No |
| GetCurrentState | 0 | No | No |
| ResetAll | 0 | No | No |
| ResetAllTrails | 0 | No | No |
| AddObject | 0 | No | No |
| RemoveObject | 0 | No | No |
| SelectPrevObject | 0 | No | No |
| SelectNextObject | 0 | No | No |
| StartStopToggle | 0 | No | No |
| StartSimulation | 0 | No | No |
| StopSimulation | 0 | No | No |
| IncreaseSpeed | 0 | No | No |
| DecreaseSpeed | 0 | No | No |
| SaveData | 0 | No | No |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit testing | 0 | 75 |
| Stress testing | 0 | 1 |
| Time performance testing | 0 | 10 |

<u>Consideration of other types of testing for Motion3DSimulationController</u>

- 3D motion is the most creative feature of our software for the user, and the simulation controller handles all the details below the user interaction layer, so if stress tests were implemented, they would be appropriate on this class specifically.
- Generating the numerical solution for the path of motion is delegated to Motion3DObject which is in turn delegated to ODEInitialValueProblem. Numerical accuracy performance tests are best left to these other classes rather than this one.
- Time performance testing could be of interest in this case because on this class, it could measure how long it takes each frame to calculate and render for any number of objects. This has another dimension in the fact that the simulation speed can be increased or decreased, which would likely change the performance values.

- Barring getting Play Mode testing to work, we could do non-automated functional testing with a given test procedure and expected outcome.

## Motion3DVectorField (Monobehavior)

Partially tested
Some tests are written but more are needed. Some key functionality cannot be tested in Edit Mode and so must wait until we get Play Mode testing working.

| **Motion3DVectorField Unit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| Start() | 1* | Yes | No |
| Update() | 2 | Yes | No |
| InitializeVectorField() | 1* | Yes | No |
| ToggleVectorField() | 1 | Yes | N/A |
| EnableVectorField() | 0 | No | No |
| DisableVectorField() | 0 | No | No |
| IncreaseSpacing() | 1 | Yes | N/A |
| DecreaseSpacing() | 1 | Yes | N/A |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit testing | 7* | 10 |
| Accuracy performance tests | 0 | 3 |
| Time performance tests | 0 | 1 |

Consideration of other types of testing for Motion3DVectorField

- Numerical accuracy performance tests are needed. Although this is a peripheral graphics feature, we do generally want to ensure the vectors are pointing in roughly the correct direction.
- Time performance tests could also be of interest, as updating 1000 vectors according to a user-defined function could be resource-intensive.

Notes on Motion3DVectorField unit testing:

- InitializeVectorField and Start each have one test that simply tests if the code runs without exception. We do not believe this constitutes testing this function well.
- Tests for adjusting spacing could be additionally tested in edit mode by reading the objects' transform positions.
- The major functionality of this class is in Start and Update in conjunction with the public VectorFunctionND F, which would not function appropriately in Edit Mode tests.
- Testing InitalizeVectorField() reveals the following error: "Unhandled log message: [Error] Instantiating material due to calling renderer.material during edit mode. This will leak materials into the scene. You most likely want to use renderer.sharedMaterial

instead."
Through testing we were able to update and fix an obscure error that may only pop-up in production.

## Motion3DSceneController (Monobehavior)

Untested

This class is mostly a collection of GUI handlers. With the right setup, it's possible we could test this in Edit Mode. We have written some sanity tests for Start and Update, but nothing else so far.

| **Motion3DSceneController Unit Testing : Functions** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| Void Start() | 1* | No | N/A |
| Void Update() | 1* | No | N/A |
| MenuBar_MenuButtonPress | 0 | No | N/A |
| MenuBar_StartButtonPress | 0 | No | N/A |
| MenuBar_ResetButtonPress | 0 | No | N/A |
| MenuBar_ConfigButtonPress | 0 | No | N/A |
| MenuBar_SetStateButtonPress | 0 | No | N/A |
| MenuBar_SaveDataButtonPress | 0 | No | N/A |
| MenuBar_SavedProblemsButtonPress | 0 | No | N/A |
| ConfigView_OkButtonPress | 0 | No | N/A |
| ConfigView_CancelButtonPress | 0 | No | N/A |
| SetState1stOrderView_OkButtonPress | 0 | No | N/A |
| SetState1stOrderView_CancelButtonPress | 0 | No | N/A |
| SetState2ndOrderView_OkButtonPress | 0 | No | N/A |
| SetState2ndOrderView_CancelButtonPress | 0 | No | N/A |
| UserDefinedOKButtonPress | 0 | No | N/A |
| MenuBarBottom_AddObjectButtonPress | 0 | No | N/A |
| MenuBarBottom_RemoveObjectButtonPress | 0 | No | N/A |
| MenuBarBottom_NextObjectButtonPress | 0 | No | N/A |
| MenuBarBottom_PrevObjectButtonPress | 0 | No | N/A |
| TimeSlider_OnValueChanged | 0 | No | N/A |
| FileManager_LoadButtonPress | 0 | No | N/A |
| FileManager_SaveButtonPress | 0 | No | N/A |
| Keyboard input: space bar starts/stops simulation | 0 | No | N/A |
| Keyboard input: f to speed up | 0 | No | N/A |
| Keyboard input: s to slow down | 0 | No | N/A |

| | | | |
|---|---|---|---|
| Keyboard input: w to select next object | 0 | No | N/A |
| Keyboard input: q to select previous object | 0 | No | N/A |
| Keyboard input: h to toggle VR mode | 0 | No | N/A |
| Keyboard input: v to toggle vector field | 0 | No | N/A |
| Keyboard input: c to decrease vector field spacing | 0 | No | N/A |
| Keyboard input: b to increase vector field spacing | 0 | No | N/A |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit testing | 2 | 60 |

Consideration of other types of testing for Motion3DSceneController
Barring getting Play Mode testing to work, we could do non-automated functional testing with a given test procedure and expected outcome.

Notes on Motion3DSceneController unit testing:

- Listed tests are exception tests or sanity tests only.
- We don't know how to test keyboard input functionality in our testing environment. It may be possible in Play Mode testing, but we can't get that working.

## Motion3DFileManager (Monobehavior) and Motion3DFileButton (Monobehavior)

Untested
These two classes have a coupled dependence and so should be tested together. Neither of these classes are dependent on update cycles, so they should possible to test in Edit Mode. However, we haven't gotten to testing them.

| **Motion3DFileManager Unit Testing** | **Num** | **Happy** | **Sad** |
|---|---|---|---|
| RefreshFilesList() | 0 | No | No |
| DeselectAll() | 0 | No | No |
| CloseButton_OnPress() | 0 | No | No |
| SaveButton_OnPress() * | 0 | No | No |
| **Motion3DFileButton Unit Testing** | **Num** | **Happy** | **Sad** |
| Select() | 0 | No | No |
| Deselect() | 0 | No | No |
| OnClick() | 0 | No | No |

| **Total tests for this module** | Have | Estimated number needed for good testing |
|---|---|---|
| Unit testing | 0 | 10 |

Consideration of other types of testing for Motion3DFileManager and Motion3DFileButton
Unit testing should be sufficient for this module.

Notes on Motion3DFileManager unit testing:

- SaveButton_OnPress is dead code.

## Conclusions

- We have written a total of 161 tests. In order to classify our project as well-tested, we estimate needing a total of 535 tests. Thus we are approximately 30% done with testing.
- We would have had a much easier time with this had we had test-driven development at the start of our project. It may have limited us from implementing as many things as we did, but that’s irrelevant since we seem to have written too much to test in a reasonable time frame.