

Contact: David Holmes (david.geo.holmes@gmail.com)

STEMCstudio Overview

What is STEMCstudio?

STEMCstudio is a live-code editor with intelligent tooling and an integrated output view. It aims to provide the essential tools that students and researchers need for a rapid code-execute cycle on non-trivial computational modeling projects. Optimized for scientific educational and research applications, STEMCstudio provides a new representation for model-based inquiry.

Why STEMCstudio?

STEMCstudio improves on existing code editors for education and research in a number of crucial ways. Most notably, STEMCstudio is intelligent, flexible, mathematical, easy to use, and was designed for education and research use.

Designed for Education and Research

STEMCstudio was designed to provide a coding environment that would be most appropriate for a student wanting to use computing as a way to experience STEM concepts. In particular it closes the gap between coding and execution, provides modern tooling and scientific capabilities, and delivers it all through the browser and so requiring no installation.

Intelligent Tooling

STEMCstudio provides immediate real-time feedback and assistance through semantic analysis of the program code. Errors and hints are reported directly in the code as the user types, and are based upon the meaning of the code, not just its syntax.

Flexibility to Create What You Want

STEMCstudio allows you to create any kind of web application because it does not impose any framework, libraries, or otherwise constraining paradigm. This allows STEMCstudio to be used broadly in a wide range of STEM computational problem domains, from traditional sciences and

mathematics through modern complex science. STEMCstudio can be used standalone, or as a front end to the cloud or a supercomputer for computing at scale.

STEM Concept Understanding

STEMCstudio intrinsically understands the meaning of a wide range of mathematical operators in code, and LATEX notation for formulas in documentation. Through libraries specially developed for scientific use and popular open-source libraries, STEMCstudio can perform calculations using geometric algebra and S.I. units of measure, and present the results as 3D GPU-accelerated graphics.

Simplicity and Ease of Use

STEMCstudio is delivered through a web browser, so there is no installation required. Code is executed directly in the browser, so there is no engineering build/release step. The feature list of STEMCstudio has been informed by best practices in modern software engineering and has been curated to transfer maximum value in an educational and research context without overwhelming the user.

Comparison with Other Tools

If you are not familiar with code editor terminology, please feel free to skim or skip this section and return later, using it as a summary.

STEMCstudio has much in common with existing modeling tools, and tries to improve upon them where either newer technology offers an advantage, or by incorporating proven best practices from software engineering that add pedagogical value. The following table of features listed vertically and environments listed horizontally provides a summary of the essential improvements.

	VPython	Glowscript	Pyret	STEMCstudio
IntelliSense	No	No	No	Yes
Semantic Checking	No	No	Yes (Batch)	Yes (Realtime)
Syntax Checking	Yes (Batch)	Yes (Batch)	Yes (Batch)	Yes (Realtime)
Mainstream Language	Yes (Python)	No (RapydScript)	No (Pyret)	Yes (TypeScript)

Zero Install	No	Yes	Yes	Yes
Multi-file or Modules	Yes	No	No	Yes (ES2015)
URL Sharing	No	Yes	Yes	Yes
External Libraries	Yes	No	No	Yes
General Purpose	Yes	No	No	Yes
README	No	No	No	Yes (markdown)
Geometric Algebra	No	No	No	Yes
Real-time Collaboration	No	No	No	Yes (Shared Editing)
SI units	Not integrated	No	No	Yes (library)
Testing Framework	Not integrated	No	Yes (Proprietary)	Yes (Jasmine)
3D Graphics	Yes (OpenGL)	Yes (WebGL)	No	Yes (WebGL)
Physics Engine	No	No	No	Yes (NEWTON)
GitHub Integration	No	No	No	Yes (Gist)
GPU Programming	No	No	No	Yes (GLSL)
Code Linting	No	No	No	Yes (tslint)
Web Metaphor	No	No	No	Yes (HTML-JS-CSS)
Search	No	No	Yes, (Google Drive)	Yes (STEMC arXiv)
Prevent Infinite Loops	N/A	?	Yes	Yes
Code Snippets	No	No	No	Yes
Greek Symbol Keyboard	No	No	No	Yes

STEMCstudio vision

“Expressing Ideas through Computational Modeling”.

The central idea behind STEMCstudio is that active exploration is essential to student learning and that computational modeling offers an essential, unambiguous, tangible, and dynamic modeling representation, which is impossible to achieve with mathematical notation alone. A more complete philosophy that motivated and guided the development of STEMCstudio may be summarized as follows.

STEMCstudio principles, philosophy, and goals

- The purpose of computing is insight, not numbers (Richard Hamming, 1962).
- Leverage the synergies between computing, mathematics, and physics to make the whole student learning experience more satisfying and productive.
- Support the modernization of the mathematics and science curriculum.
- Support learning fundamentals and essential subject elements.
- Support access for all.
- Make mathematical and physical modeling an active learning experience.
- Support a constructive, modeling and inquiry-based pedagogy.
- Support an integrated computing and STEM curriculum.
- Consolidate the tools needed for computing across all STEM disciplines, reducing cognitive load for students and teachers alike.
- Support new ways of teaching Geometry and Geometric Algebra by using 3D Computer Graphics.
- Introduce modern software development best practices to the student.
- Use state-of-the-art technology to improve the student learning experience.
- Support the development of fundamental skills, not gee-whiz technology.
- Anticipate the need for essential future skills such as Machine Learning development.

STEMCstudio Summary

STEMCstudio in a nutshell

STEMCstudio is a general-purpose online programming environment that has been optimized for educational use at the high school and university level. STEMCstudio lowers the software engineering barrier for students and teachers and yet still provides a rewarding and powerful experience that would be familiar to a modern software developer.

A Modern Programming Environment optimized for Learning STEM

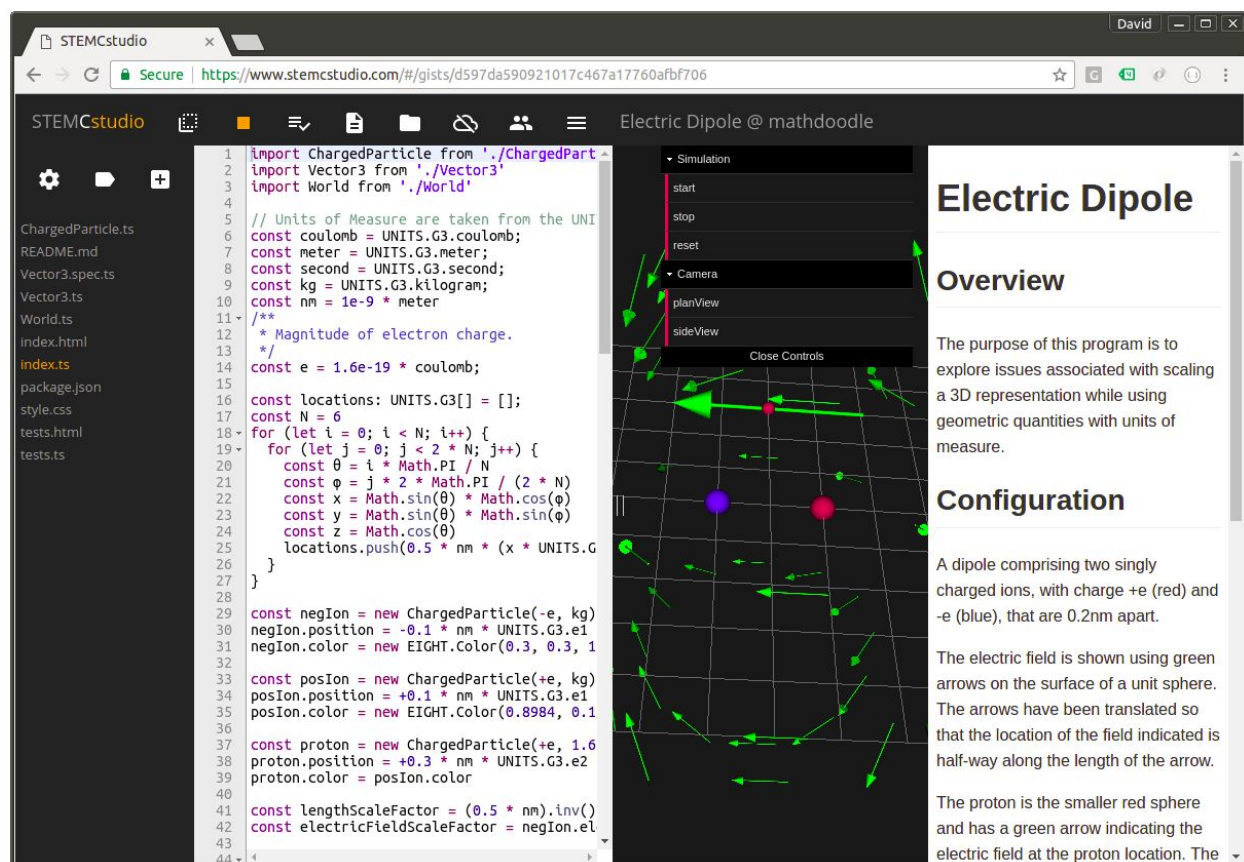


Figure: STEMCstudio Integrated Modeling Environment (Electric Dipole simulation)

The screenshot above shows STEMCstudio running in Google's Chrome browser; no install or setup is required. The application runs from a secure URL, <https://www.stemcstudio.com>, which prevents malicious attacks. On the left is the file explorer. STEMCstudio supports multi-file and multi-module projects which is an essential organizational feature for non-trivial real-world projects. To the right of the explorer is the code editor with syntax coloring, IntelliSense, and

linting to help the student understand the code and avoid errors. The output window is center-right and is completely defined by the user's program; it is a web application running inside STEMCstudio. This side-by-side arrangement provides rapid feedback and avoids the software engineering hassles of application deployment.

New Modeling Representations for Modeling-Based Inquiry

Computational Modeling in STEMCstudio provides two new modeling representations to a Modeling-Based Inquiry pedagogy. The first is the representation of mathematical formulae and algorithms in computer code. This is shown in the previous figure. The second is the graphical modeling of the system behavior, which can involve 3D graphics, kinematic graphs, and tabular data. The following figure shows a rigid body dynamics simulation with 3D GPU-accelerated graphics and a realtime graph of the system energies. The 3D graphics library (EIGHT) has been carefully designed to avoid presenting irrelevant details while at the same time maintain a convincing real-world view.

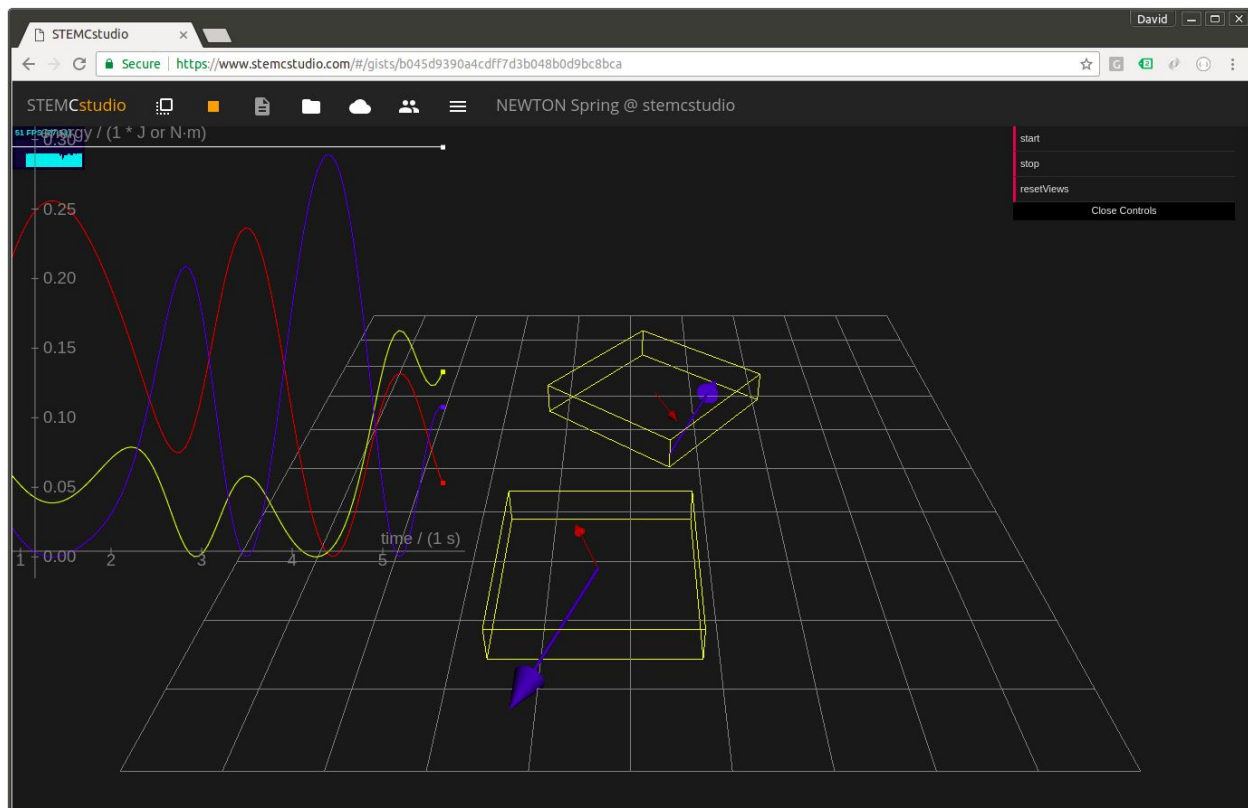


Figure: Rigid Body Dynamics simulation in STEMCstudio

Challenging Physical Misconceptions and Strengthening Mathematical Knowledge

The previous Electric Dipole and Rigid Body Dynamics simulations also illustrate the point that STEMCstudio can be used in one of two soft and overlapping modes, explicit/imperative and implicit/declarative. These two modes are described below. Another possible mode is goal-based programming using Machine Learning (ML). Libraries and an Application Programming Interface (API) for ML are anticipated.

Creating simulations from scratch (explicit/imperative mode)

The Electric Dipole simulation involves explicit coding of all the physical details; Coulomb's Law, Lorentz Force Law, and Newton's equations, as well as the coding of mathematical concepts such as algebra, geometry, and calculus. This is the mode in which the student is most likely to do their modeling work.

Creating demonstrations from a Physics Engine (implicit/imperative mode)

The Rigid Body Dynamics simulation, on the other hand, was rapidly constructed using the NEWTON library, which contains a Physics Engine, physical quantities (scalars, vectors, spinors, multivectors), optional units of measure, and graphing, and is intended as a demonstration or virtual experiment. Such demonstration may be used to challenge misconceptions in the Force Concept Inventory.

Multimedia Learning and Documentation

On the far right of the STEMCstudio workspace is the documentation window, which is a rendering of the README.md project file. The documentation window may contain embedded media such as videos, LATEX mathematical notation, and custom text. The documentation is useful for both instructor guidelines and student project notes. Each window (code, output, documentation) may be independently shown or hidden using the buttons on the toolbar. The following screenshot shows the output hidden while attention is drawn to the embedded video.

STEMCstudio

Secure | <https://www.stemcstudio.com/#/gists/4872683759c799265ab55f6b118bb92d>


STEMCstudio YouTube Embedding @ stemcstudio

README.md
fs.gls
index.html
index.ts
package.json
style.css
vs.gls

```
1 const engine = new EIGHT.Engine('canvas')
2   .size(500, 500)
3   .clearColor(0.1, 0.1, 0.1, 1.0);
4
5 const camera = new EIGHT.PerspectiveCamera()
6   camera.eye.z = 5
7
8 const trackball = new EIGHT.TrackballControls(camera)
9   trackball.subscribe(engine.canvas)
10
11 const ambients: EIGHT.Facet[] = [];
12 ambients.push(camera)
13
14 const primitive: EIGHT.Primitive = {
15   mode: EIGHT.BeginMode.LINE_STRIP,
16   attributes: {
17     X: {values: [0, 0, 0, 1, 1, 0, 0, 0], size: 2,
18   }
19 }
20
21 const geometry = new EIGHT.GeometryArrays(engine,
22 const material = new EIGHT.HTMLScriptsMaterial(['v
23
24 const T = 4;
25 const ω = 2 * Math.PI / T;
26
27 const animate = function(timestamp: number) {
28
29   const t = timestamp * 0.001;
30   const θ = ω * t;
31
32   engine.clear();
33   trackball.update()
34
35   // We want this call to be optimized.
36   material.use();
37   // We want this call to be optimized.
38   geometry.bind(material)
39   // Nothing happens because the vertex shader doe
40
```

The Role of Programming

Gerald Jay Sussman: The Role of Programming (Dan Frie...



Gerry Sussman of M.I.T.

Figure: STEMCstudio with embedded YouTube video.

STEMCstudio Deep Dive

Features and Benefits

STEMCstudio contains features that have been “borrowed” from best practices in modern software development that promise to provide pedagogical benefits. These features have been carefully implemented so as not to be overwhelming. This section describes these features and their associated benefits.

Side-by-Side code and output to minimize context switching

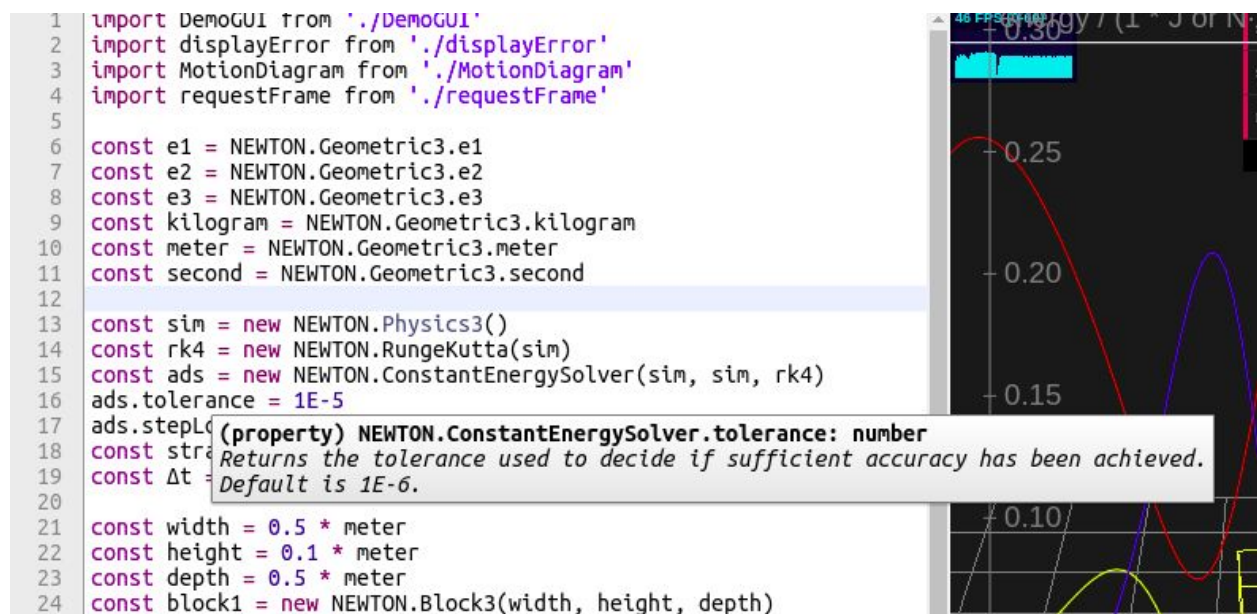


Figure: STEMCstudio Side-by-Side editing and output.

Optional Typing providing Syntax and Semantic checking

One of the most compelling reasons for using STEMCstudio is its ability to analyze your code based upon type information (semantic analysis). In order to see why this is so powerful, we first consider an example of code written in an environment that does not support type information.

In the example below, a function has been written to compute the scalar product of two vectors. Other than from the documentation comment (jsdoc), there is no indication or verification of the types of `a` and `b`. In STEMCstudio, the mouse hover popup would show the type of `a.x` to be `any`, meaning anything. Furthermore, we don't know the dimensionality of the space that the vectors

are operating in. Finally, there looks to be a bug in the `a.s * b.z` expression term, but we don't know for sure and this would not manifest until execution time.

```
/**
 * Untyped example of the scalar (dot) product, a.b, of two vectors, a and b.
 */
function dot(a, b) {
  return a.x * b.x + a.y * b.y + a.s * b.z;
}
```

any

Figure: A code example without type information.

What happens if we take the same code and drop it into STEMCstudio as the contents of a TypeScript file?

The parameters of the function become highlighted and a red error marker appears to the left in the gutter region.

```
1 /**
2  * Untyped example of the scalar (dot) product, a.b, of two vectors, a and b.
3  */
4  export function dot(a, b) {
5    return a.x * b.x + a.y * b.y + a.s * b.z;
6  }
```

Figure: A code example without type information as a STEMCstudio TypeScript file.

If we hover our mouse over the error marker we will get a pop-up textual explanation of the errors. We can also press the key combination Alt-E (Windows/Linux) or fn-F4 (Mac) to render the error annotations inline with the code for better context:

```
1 /**
2  * Untyped example of the scalar (dot) product, a.b, of two vectors, a and b.
3  */
4  export function dot(a, b) {
```

Parameter 'a' implicitly has an 'any' type.
Parameter 'b' implicitly has an 'any' type.

```
5    return a.x * b.x + a.y * b.y + a.s * b.z;
6  }
```

Figure: A code example without type information after pressing Alt-E or fn-F4.

We now begin to see the problem. For resolution, we add appropriate types to express the meaning (semantics) of *a* and *b*. The following figure show the result after we have declared the types for *a* and *b*. Notice that the type declarations have enabled the semantic analyzer to detect an error.

```
1 /**
2  * Untyped example of the scalar (dot) product, a·b, of two vectors, a and b.
3  */
4 export function dot(a: EIGHT.VectorE3, b: EIGHT.VectorE3) {
5   return a.x * b.x + a.y * b.y + a.s * b.z;
6 }
```

Property 's' does not exist on type 'VectorE3'.

Figure: A code example with type information and a diagnostic error.

Wondering what the appropriate fix might be, we hover our mouse over *a.x* to gain some extra insight. Knowing the cartesian expression for the scalar product confirms that the factor *a.s* should be changed to *a.z*

```
1 /**
2  * Untyped example of the scalar (dot) product, a·b, of two vectors, a and b.
3  */
4 export function dot(a: EIGHT.VectorE3, b: EIGHT.VectorE3) {
5   return a.x * b.x + a.y * b.y + a.s * b.z;
6 }
```

(property) EIGHT.VectorE3.x: number
The magnitude of the projection onto the standard *e1* basis vector.
The Cartesian x-coordinate.

Figure: Hovering the mouse over the x-coordinate of a vector type.

Delighted with our progress, we hover our mouse over the name of the function and see the full function prototype juxtaposed vertically with the function jsdoc description. This is shown below. Notice that the analyzer has inferred the return type of the function to be 'number'. This confirms and corresponds to our mathematical understanding of the scalar product as being a scalar-value function of two vector arguments. The inference of the return type is also a manifestation of the concept of optional typing; we only need to supply as many types as are necessary to have the analyzer infer the rest. In this case we may prefer to explicitly define the return type of the *dot* function so that its type is manifest without hovering and as a way to future-proof against implementation bugs that try to return something other than a number.

```
1  /**
2   * Untyped example of the scalar (dot) product, a·b, of two vectors, a and b.
3   */
4  export function dot(a: EIGHT.VectorE3, b: EIGHT.VectorE3) {
5      ... return a.x *
6  }
7
```

function dot(a: EIGHT.VectorE3, b: EIGHT.VectorE3): number
Untyped example of the scalar (dot) product, a·b, of two vectors, a and b.

Figure: Hovering the mouse over the function name symbol.

IntelliSense

IntelliSense is a general term for a collection of features: Listing properties, Auto Completion, and Quick Information. These features help you understand the code and allow the user to make changes accurately and more quickly. A popup listing object properties appears after typing a period, '.', following an object. As shown below, the listing also includes the type of the property.

```
53  const graph = new NEWTON.EnergyTimeGraph('graph', sim.varsList)
54
55  graph.axes.hAxisScale =
56  graph.axes.vAxisScale =
57
58  const diagram = new Moti
59
60  const gui = new DemoGUI(
61
62  const stats = new Stats(
63  stats.domElement.style.p
64  stats.domElement.style.t
65  document.body.appendChild(stats.domElement)
66
```

Dimensions class
DisplayAxes class
DoubleRect class
DrawingMode enum
EnergySystem interface
EnergyTimeGraph class
EulerMethod class
Force3 class

Figure: STEMStudio Context Assistant shows properties of an object.

The context assistant may also be invoked using the key combination Ctrl-Space (Windows/Linux/Mac), which is very useful for discovering imports and available properties for options.

The communication of coding errors using markers has already been described. The analyzers for syntactic, semantic, and linting errors are able to show all errors at once rather than halting after the first error is discovered. This can make it easier to resolve the root cause of an error

and quickly resolve multiple errors. In the following screenshot, the cause of the 'width' argument error is the incorrect specification of the unit of measure.

```

13 const sim = new NEWTON.Physics3()
14 const rk4 = new NEWTON.RungeKutta(sim)
15 const ads = new NEWTON.ConstantEnergySolver(sim, sim, rk4)
16 ads.tolerance = 1E-5
17 ads.stepLowerBound = 1E-7
18 const strategy = new NEWTON.DefaultAdvanceStrategy(sim)
19 const Δt = 0.01 * second
20
21 const width = 0.5 * meters
22
23 const height = 0.1 * meter
24 const depth = 0.5 * meter
25 const block1 = new NEWTON.Block3(width, height, depth)

```

Cannot find name 'meters'.

constructor NEWTON.DefaultAdvanceStrategy(simulat

Figure: STEMCstudio multiple error reporting and hover tooltip.

Mainstream Language (TypeScript) with extensive Community support

The question of what is the best programming language for students is often contentious. The following table summarizes the main differences:

Aspect	TypeScript	Python	Comment
Optional Typing (Explained Above)	Mature specification and implementation. Available at version 2.x and post version 1.x for over a year.	Discussed in Python 3.6. Provisional in `C` implementation. Not available in transpilers to JavaScript.	The pedagogical benefits of optional typing completely outweigh any other consideration (discussed below).
Assignment	`const` and `let` keywords distinguish two kinds of variable declaration.	No concept of variable declaration.	Python failing to distinguish between defining a variable and assigning a value leads to shadowing bugs and complex global environment model.
Lambda Expressions	Any general function, multiple lines allowed.	Must be one-line expression	Multiline Lambda expressions not supported in Python

			because of clashes with whitespace significant. See http://stackoverflow.com/questions/1233448/no-multiline-lambda-in-python-why-not
Scoping	Functions are primary scoping mechanism and extent is delimited by braces. Whitespace is insignificant.	Scopes are delimited by indenting.	Python is apparently simpler but has issues for defining anonymous (lambda) functions. Additionally, the requirement for colons `:` at the end of Python class and function declarations throws into question whether Python is any clearer.
Runtime	Same as JavaScript	Extra built-in types e.g big integers, lists.	Trade off between adding extra libraries (TypeScript) and runtime bloat (Python).
Language Fragmentation and Interoperability	Not an issue. Strong cohesive community behind official Microsoft project. A TypeScript design philosophy is to be plain JavaScript with optional typing. Easy to interoperate with any JavaScript library.	In the browser, transpilers have an engineering trade-off between performance and faithfulness to the Python execution model. E.g. Skulpt is more like Python 2.x but runs < 1/10 speed of similar JavaScript code. RapydScript performs much better but has a JavaScript execution model. External libraries may need	Using Python in the browser also presents a documentation problem: Any external JavaScript library that is usable using a shim must also have appropriate API documentation.

		shim layers.	
Utility	JavaScript is the language of the web browser. It is a MUST for front-end web development. It can also be used on back-end servers.	Advocates claim that Python is more widely used in academia.	The arguments may be of little consequence because the imperative programming skills that are common are easily transferable, specific technological skills are less important than fundamentals.

TypeScript has been chosen for STEMCstudio mostly for the powerful intelligent editing features that it provides which result from the optional typing feature. Optional typing in TypeScript clarifies the meaning of code, helps programmers (students and teachers) to write code that is free from type errors, and most of all, **data types in a computer language have the potential to improve the learning of mathematics when students make connections to computer programming.**

This final point is worth elaborating because it informs the discussion of an integrated mathematics and computing curriculum. Contrary to popular thinking, mathematical notation is sloppy. Mathematical notation is precise enough to communicate ideas between humans of a similar level of mathematical maturity, but it sacrifices precision to be expedient, and only provides clarity for those already somewhat acquainted (Gerry Sussman, M.I.T. <https://www.youtube.com/watch?v=arMH5GjBwUQ>). Some examples:

1. The difference between a function, f , and the value returned from a function, $f(x)$.
2. Suppressing arguments to functions.
3. The meaning of the notation dy/dx for a derivative.
4. The difference between inverse notation (exponent -1) and reciprocal.
5. The notation of a vector type as a real raised to the number of dimensions.

All of these ambiguities are exposed and clarified when a student establishes a bidirectional correspondence between mathematical notation and computer code. This is best done by having the student implement mathematical objects in code.

JavaScript is misunderstood by many because for a long time it was simply the language that Netscape developed as the scripting code for its browser over twenty years ago. However, since the introduction of Node.JS, interest in JavaScript has risen dramatically and the language is now undergoing major enhancements. TypeScript adds type information to plain JavaScript.

The benefits accrued to the student are similar to those for a modern software professional - easier to understand and scale a code base to solve real-world applications.



Figure: Microsoft TypeScript from <https://www.typescriptlang.org/>

Multi-module/file projects for optimal code organization and reuse

STEMCstudio allows you to break up your code into multiple files or code modules. The converse or anti-pattern is to write all code into a single file. Such an organization is difficult to navigate and the lack of layering makes it difficult to penetrate.

Automatic Code Formatting

The ability to automatically reformat code is a software best practices because it improves readability and prevents whitespace-only differences from being flagged as real code differences. For students, who may have no experience with code layout and indenting, the formatter provides a good way to learn about code structure. Suppose the student enters:

```
1  /**
2   * Computes a random number in the range [min, max]
3   */
4  export function random(min: number, max: number): number {
5    return min+(max-min)*Math.random();
6  }
7
```


Figure: Code layout before automated formatting

Using the key combination Ctrl-Shift-I (I suggests Indenting), the code whitespace is adjusted to give a more readable layout and with trailing whitespace removed.

```
1  /**
2   * Computes a random number in the range [min, max]
3   */
4  export function random(min: number, max: number): number {
5      return min + (max - min) * Math.random();
6  }
7
```

Figure: Code layout after automated formatting

Although this is a somewhat contrived example, automatic code formatting, is a time-saver for educators and students alike in more complicated structured code.

Linting - Code Housekeeping and Style Standardization

Code linting, the enforcement of standards that affect non-whitespace elements as well as whitespace elements has benefits for students learning to code as well as experienced practitioners.

As an example, consider the following code line that has been highlighted by the linter.

```
19  ... for (let i = 0; i < normals.length; i++) {
    Expected a 'for-of' loop instead of a 'for' loop with this simple iteration
20  ... const n = normals[i];
21  ... const collision = Ldiv2ma + dot(ball.X, n) <= 0;
```

Figure: Linting error on a 'for' loop.

By making the suggested change, the resulting code is easier to read, conveys its meaning, and reduces the potential for bugs. Here is the result:

```
17  ... const Ldiv2ma = Ldiv2 - a;
18  ... for (const n of box.normals) {
19  ...     const collision = Ldiv2ma + dot(ball.X, n) <= 0;
20  ...     if (collision) {
```

Figure: Refactored code is much simpler and easier to understand.

STEMCstudio currently recommends the following selection of rules that are valuable to the professional developer and student programmer alike:

Rule name	Description
curly	Enforces braces for <code>`if`/`for`/`do`/`while`</code> statements.
comment-format	Enforces formatting rules for single-line comments.
eofline	Ensures the file ends with a newline.
forin	Requires a <code>`for ... in`</code> statement to be filtered with an <code>`if`</code> statement.
jsdoc-format	Enforces basic format rules for JSDoc comments.
no-trailing-whitespace	Disallows trailing whitespace at the end of a line.
no-var-keyword	Disallows usage of the <code>`var`</code> keyword.
one-variable-per-declaration	Disallows multiple variable definitions in the same declaration statement.
prefer-const	Requires that variable declarations use <code>`const`</code> instead of <code>`let`</code> if possible.
prefer-for-of	Recommends a <code>'for-of'</code> loop over a standard <code>'for'</code> loop if the index is only used to access the array being iterated.
semicolon	Enforces consistent semicolon usage at the end of every statement.
triple-equals	Requires <code>`===`</code> and <code>`!==`</code> in place of <code>`==`</code> and <code>`!=`</code> .

Linting is automatically applied to all projects with no effort on behalf of the user, but the user is free to override the configuration that affects which rules are applied and how.

Online and in-browser minimizing setup effort and requiring no install

STEMCstudio runs entirely in the web browser performing all of its computation using the local Central Processing Unit (CPU) and Graphics Processing Unit (GPU). The web-based architecture removes the need for complex setup and avoids the software engineering overhead of a build-deploy cycle.

General Purpose Web-Based Programming paradigm for utility and relevance

Some educational programming environments (bootstrap, Glowscript, Trinket, VPython) are frameworks into which you insert code. Their narrow focus and specialization allows a small amount of code to be written in order to complete a program that conforms to their paradigm. This can have short-term advantages, but the downside is that the tool has a lower ceiling and at best becomes cumbersome to use as you try to work around the burned-in assumptions.

STEMCstudio makes no assumptions about the kind of project you are working on. This generality makes it possible to use one tool for multiple purposes, reducing the cognitive load on students and teachers alike. An added benefit is that the tool remains relevant as the programming maturity of the user increases.

GitHub integration for archiving student portfolio and ownership

Professional software developers store their code safely in a dedicated version control system which often facilitates collaboration. STEMCstudio makes this possible by saving code to GitHub as a gist (multiple files in a flat structure). We believe that you own your code, it is part of your portfolio, and does not belong in a proprietary storage where it is not available to you.

Integrated Testing Framework (Jasmine) for student feedback and rubrics

Testing is a best practice in modern software development. In a learning environment it serves the same quality assurance purpose and makes it easier to scale to real-world projects. Testing could also provide teachers with a way of establishing a non-threatening rubrik. STEMStudio makes the implementation of testing easy. The following screenshot shows the report generated by a test run using the Jasmine testing framework.

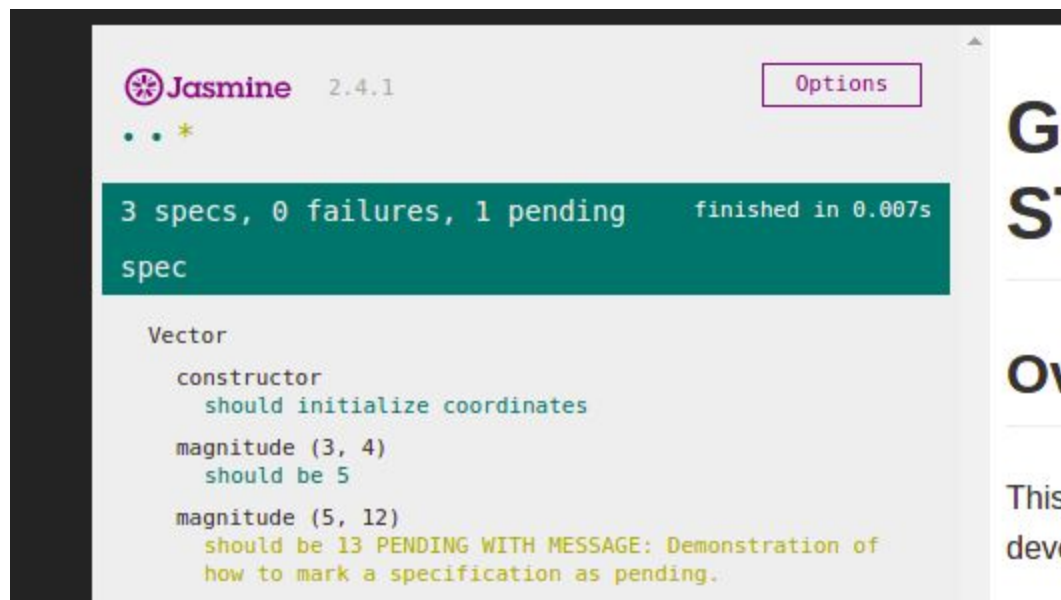
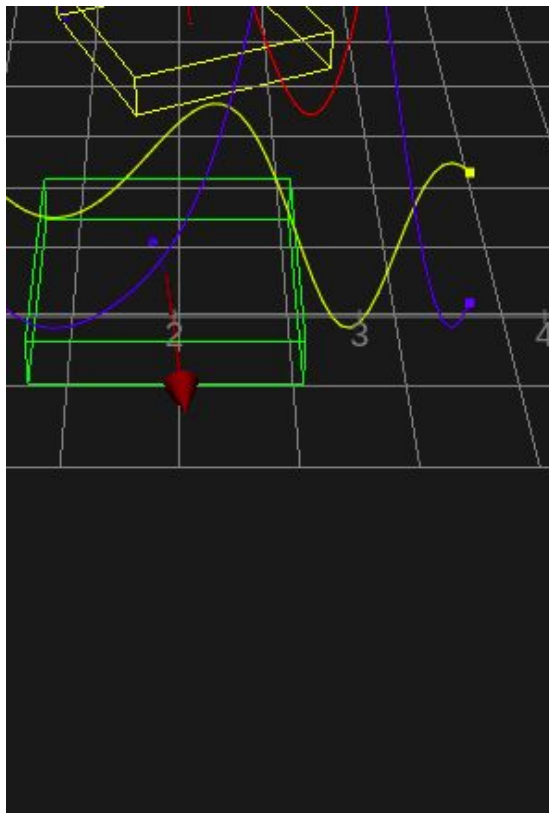


Figure: STEMStudio integrated testing using Jasmine framework.

README markdown file for instructor and student notes, LATEX enabled

A README.md is a standard best practice in most software projects. It's where you go to get orientation and an ideal place for instructor guidelines and student notes. Consistent with standard practices, STEMCstudio uses markdown for authoring which is very easy to learn. In addition, mathematical notation using LATEX syntax is supported as well as mixing of arbitrary HTML allowing for embedded media such as videos.



Mathematics

The equations of motion are

$$\dot{X}(t) = P(t)/M$$

$$\dot{R}(t) = [I^{-1}(t)J(t)]^* R(t)$$

$$\dot{P}(t) = F(t)$$

$$\dot{J}(t) = \Gamma(t)$$

In Geometric Algebra (GA), the formula for the rate of change of the attitude R is

$$\dot{R} = -\frac{1}{2}\Omega R$$

where Ω is the angular velocity bivector.

Figure: STEMCstudio README is MathJAX enabled.

Publish/Search capability for program discovery (STEMC arXiv)

A global Publish/Search capability is available, allowing you to find work, be inspired by, and learn from other authors.

Real-time distributed collaborative editing for mentoring and teamwork

STEMCstudio allows you to create a virtual room for synchronized and shared editing with other remote users. This can be useful if students encounter problems, or for students to collaborate.

Multi-programming-language (HTML, GLSL shaders, TypeScript, CSS, JSON, CSV)

STEMCstudio supports a variety of language or file formats in order to support a diverse range of programming activities. For example GLSL (Shader Language) code is used to support computer graphics using the GPU. CSV (Comma Separated Value) files are editable and a parsing library is available to easily process data files.

Graphics Processor Unit (GPU) programming using GLSL shaders and WebGL support

The modern computing platform for Deep Learning consists of multiple GPU processors. Deep Learning, the 2006 breakthrough in Artificial Intelligence (A.I.), requires fast vector-based processing available in a GPU. Every modern personal computer incorporates a GPU for hardware-accelerated graphics. A developer can access the power of the GPU easily through the WebGL API. When programming the GPU, a developer writes C-like code called GLSL. STEMCstudio supports a Graphics Library Shader Language (GLSL) editor with syntax coloring and validation. STEMCstudio automatically embeds GLSL files in HTML script tags for you, simplifying the development of 3D computer graphics code. The following screenshot shows the Mandelbrot Set and is rendered in a fraction of a second.

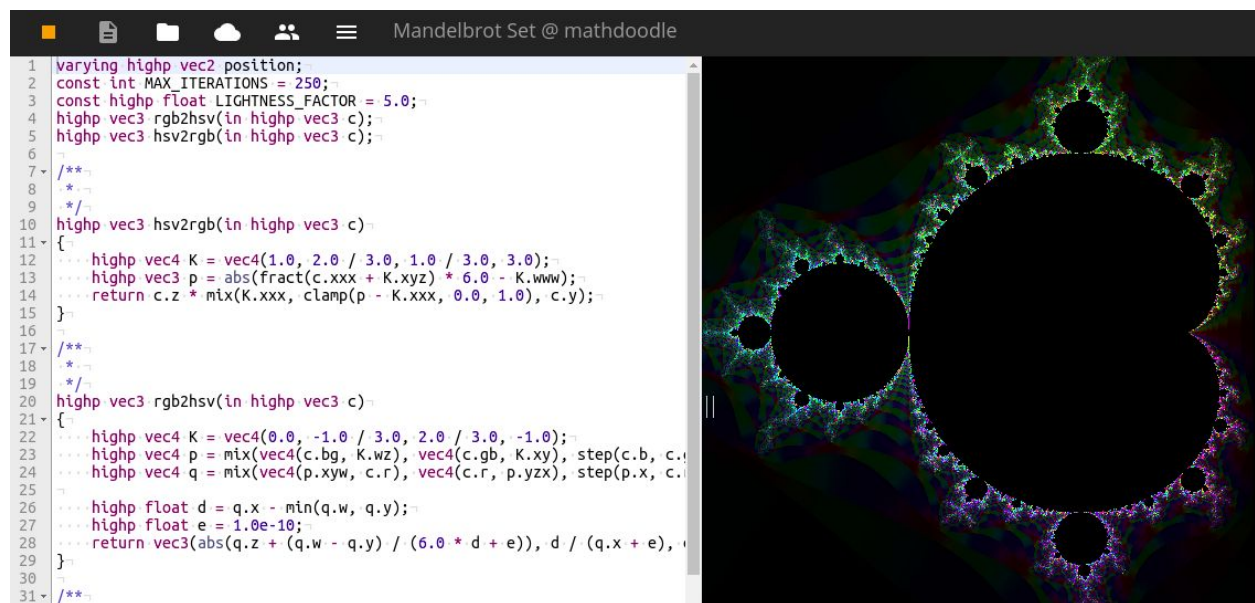


Figure: From STEMCstudio to Deep Learning; exposing the GPU and GLSL programming.

Dedicated libraries for 3D Computer Graphics

The EIGHT library is described in some detail in the next section.

Dedicated libraries for Physics, S.I. Units, Physics Engine, and Graphs

The NEWTON library is described in some detail in the next section.

Dedicated libraries for Geometric Algebra for 21st Century geometry

Geometric algebra has been described as a unified language for mathematics and physics. Geometric algebra integrates conventional (Gibbs) vector algebra into a more powerful system that subsumes quaternions and spinors. *“Over the last decade, Geometric algebra has emerged as a powerful alternative to classical matrix algebra as a comprehensive conceptual language and computational system for computer science”* (David Hestenes, Distinguished Research Professor, Department of Physics, Arizona State University).

Geometric Algebra is used consistently in both the NEWTON (Physics Engine) and EIGHT (3D Graphics) libraries.

Mathematical Operators and Geometric Algebra features for STEM applications

An important feature of a language to support mathematical programming is the ability to represent mathematical operators, e.g., + (plus), - (minus), * (multiply), and / (divide), naturally rather than having to invoke method calls. STEMCstudio allows this notation, and in addition supports a wider range of operators and precedence rules for Geometric Algebra (GA). e.g., ^ (wedge, or exterior product), | (scalar product), << (left contraction), >> (right contraction), ~ (reverse), and ! (inverse). The mechanism for adding operator overloading to a user library has been standardized, allowing a student to model and endow their own mathematical objects, e.g., vectors, spinors, matrices, complex numbers, with natural operators.

Another important feature of mathematical object design is whether the objects are immutable or mutable. Immutable objects are desirable in order that they behave like true mathematical objects without side effects. On the other hand, mutability is important for high performance graphics and achieves its goal by avoiding the creation of temporary objects. The libraries used in STEMCstudio are carefully designed to be both performant and maintain mathematical integrity and conceptual simplicity. These apparently conflicting goals are resolved in the NEWTON and EIGHT libraries by using a lock/unlock mechanism with some very simple rules.

External Library and Content Delivery Network support

You are not limited by the libraries that are packaged with STEMCstudio. The JavaScript ecosystem is enormous and any library can be imported from a server or Content Delivery Network (CDN).

Sophisticated graphing using Plotly

For data processing applications, a powerful graphing library is a must. STEMCstudio provides the open source Plotly library. In the following screenshot, a Comma Separated Value (CSV) data file has been parsed using the open source library, davinci-csv, and the data has been supplied to the plotly.js library. The student is asked to determine both the formula that fits the data and the curve parameters.

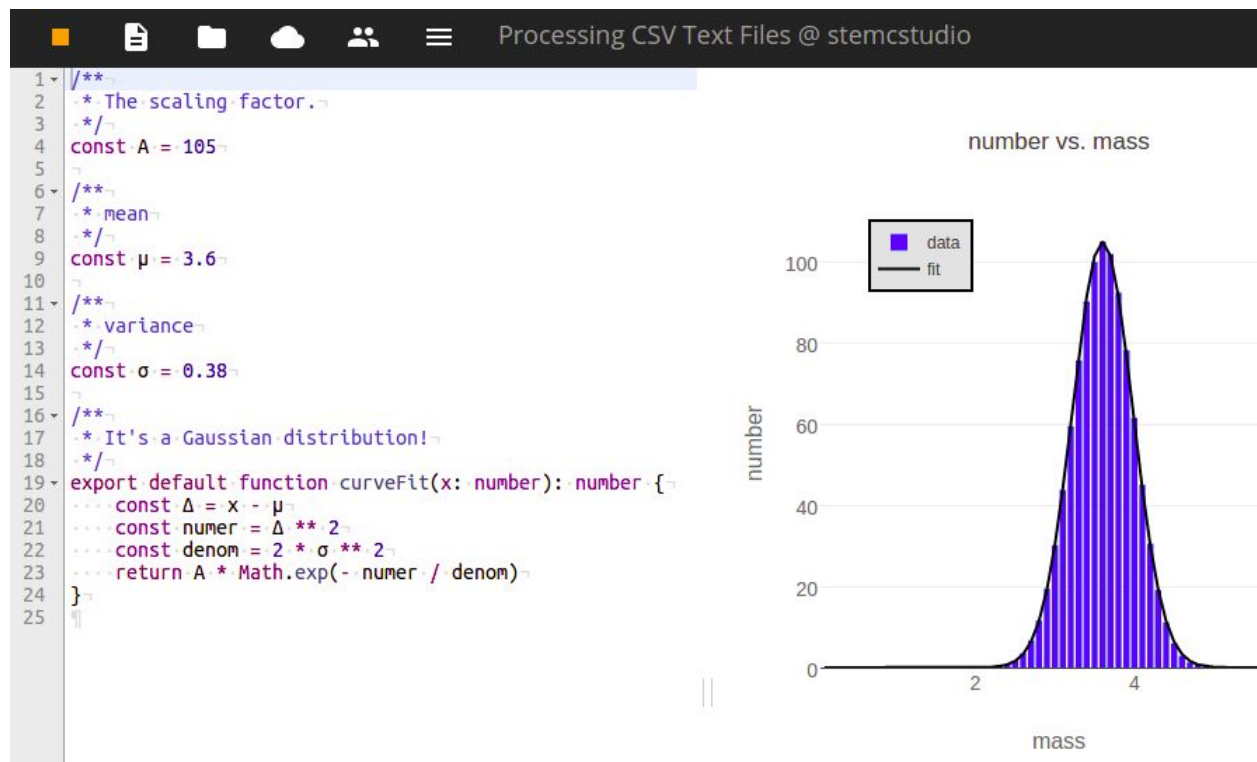


Figure: STEMCstudio allows a student to fit a curve by trial and error.

The pedagogic value of this approach over other tools that automatically fit data is that the student has to construct the model through an inquiry-based approach.

The Plotly library has an impressive collection of graph types. In addition to graphing data for discovering analytical relationships, the library has graphs that support support Data Science, as the following screenshot shows. In this example, the CSV is downloaded over the internet from a GitHub repository, parsed, filtered and finally displayed using Plotly.

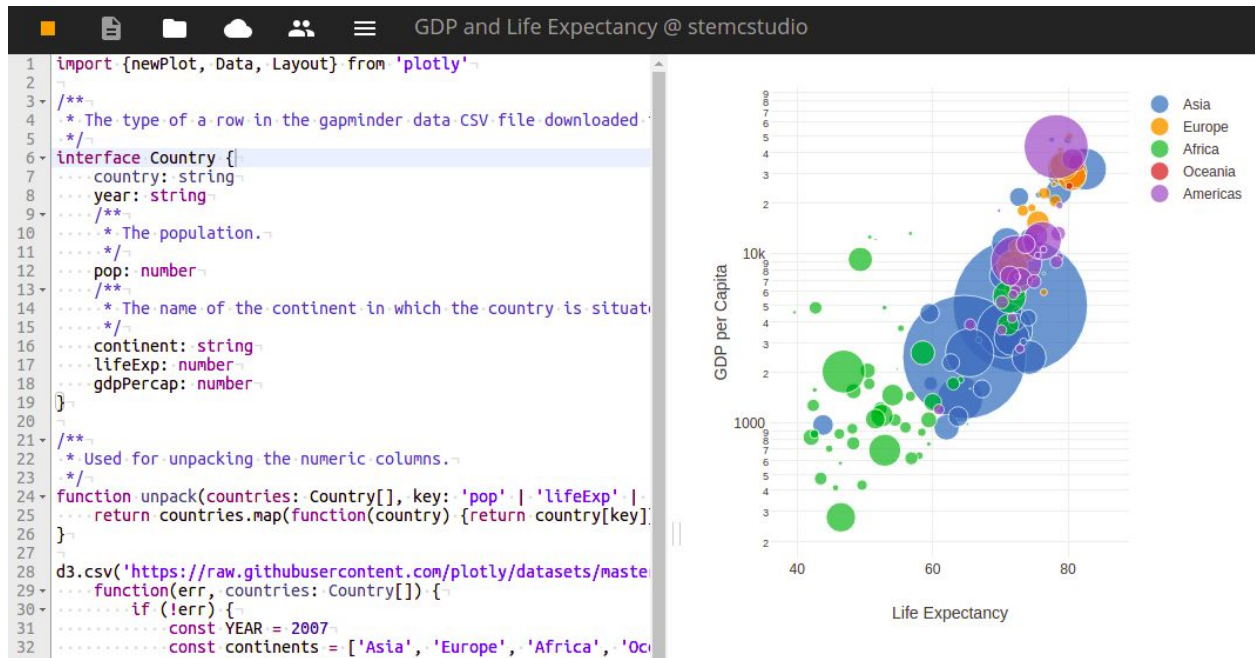


Figure: STEMCstudio with plotly facilitates Data Science

In this age of Artificial Intelligence, Deep Learning, and Big Data, the ability to manipulate data sets at scale is an important skill for a Data Scientist.

Geogebra-like scripting and diagramming capability using JSXGraph

JSXGraph is the library behind a 2D geometry tool called sketchometry (<https://sketchometry.org/en/index.html>) by the University of Bayreuth (<http://www.uni-bayreuth.de/de/index.html>). Sketchometry is dynamic geometry software with similar capability to Geogebra. Rather than use an end-user tool, STEMStudio advocates scripting the underlying library as a superior pedagogical approach. The JSXGraph library is convenient for interactive user input, and graphical output of an analytic geometric nature. Using the library rather than the user interface of a tool such as Geogebra allows the student to easily explore mathematical concepts by making connections through code.

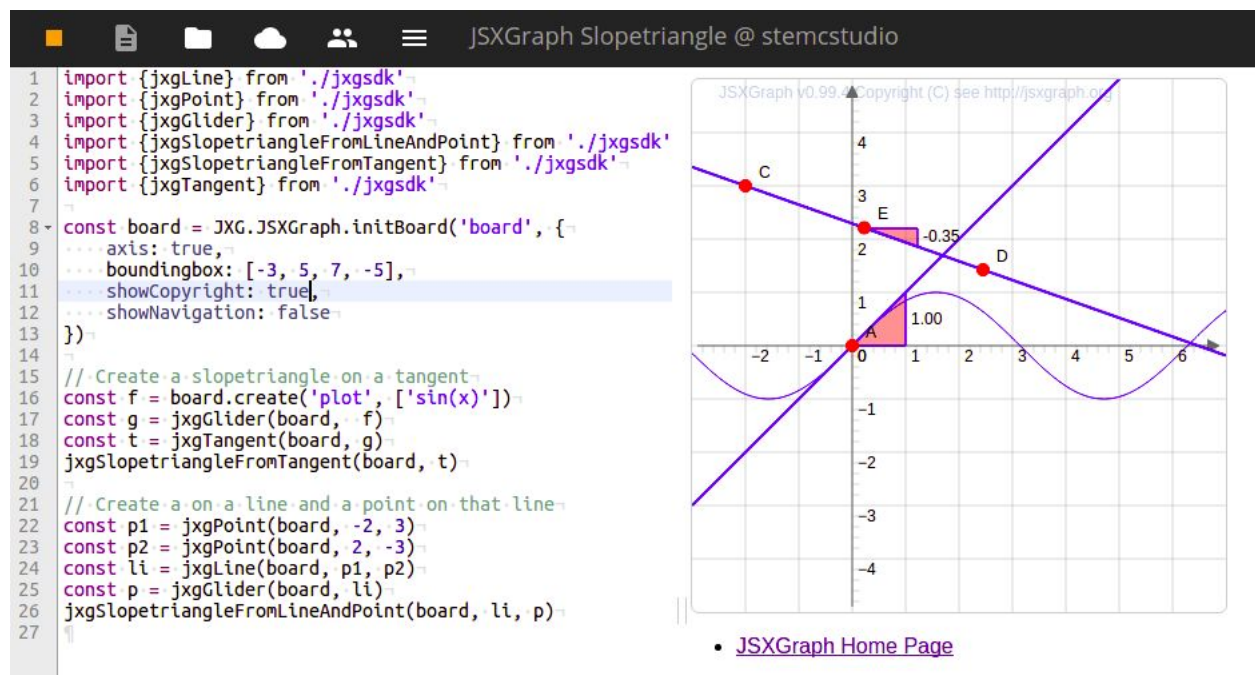


Figure: STEMStudio and JSXGraph being used to explore the derivative of a function.

In the screenshot below, JsxGraph is used to explore integration using Riemann Sums.

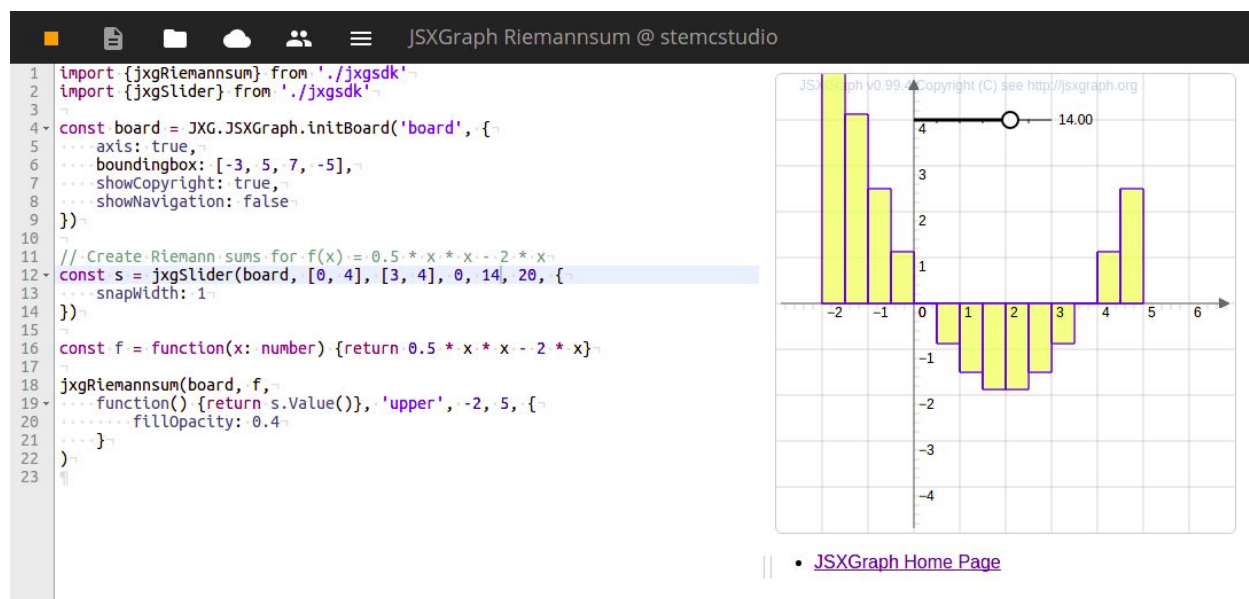


Figure: STEMcstudio being used to script JSXGraph.

URL sharing of projects

Sharing of projects increases Computational Participation (Kafai and Burke, Connected Code: Why children Need to Learn Programming). STEMcstudio allows Students and Teachers to share the URL links to their projects and assignments. Projects can also be published and discovered using the STEMc arXiv.

Developed with modern tooling to be responsive to change requests

STEMcstudio is built with some of the latest web technology in order to be easy to maintain, extensible, and responsive to emerging educational needs. All of the libraries used are open source at <https://github.com/geometryzen> and <https://github.com/stemcstudio>.

STEMCstudio Educational Libraries

While STEMCstudio is an extensible environment allowing any third-party JavaScript library to be imported, a number of libraries have been developed in support of computational modeling. This section describes some of the most important libraries for mathematics and physics.

EIGHT 3D Visualization Library

The EIGHT library is an open-source library that has been developed as an educational 3D computer graphics visualization library and is packaged with STEMCstudio for easy access and optimal integration. The motivation for EIGHT was the unsuitability of existing open-source 3D libraries which were optimized for gaming and special effects, hide mathematical details, use ad-hoc mathematics, and are designed for customization rather than progressive extension. EIGHT has a layered design allowing it to be used both as a foundation for an introductory computer science 3D computer graphics course down to the GPU level as well as provide high-level reusable objects for physics modeling. The following diagram shows a sample of the available visualization objects for physics modeling:

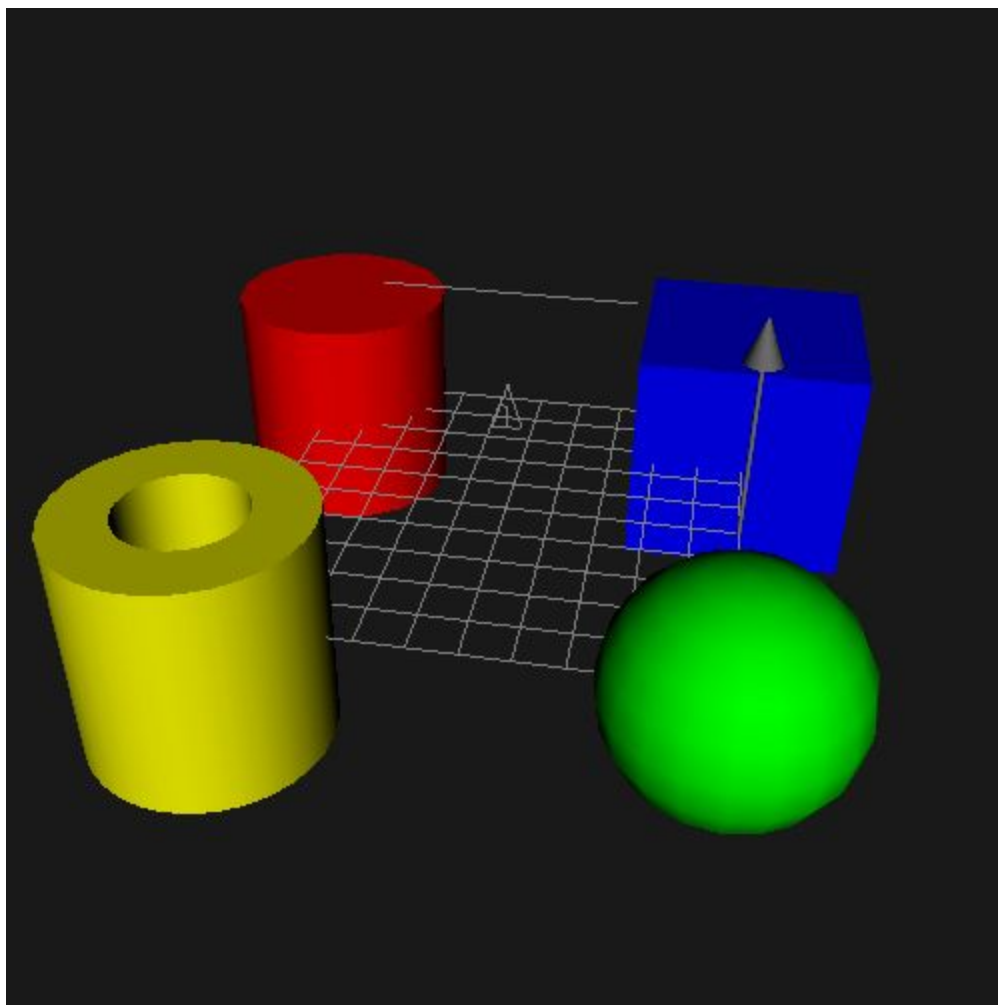


Figure: Example ready-to-go objects in the EIGHT library

All objects are available with wire-frame variants, which can be very useful:

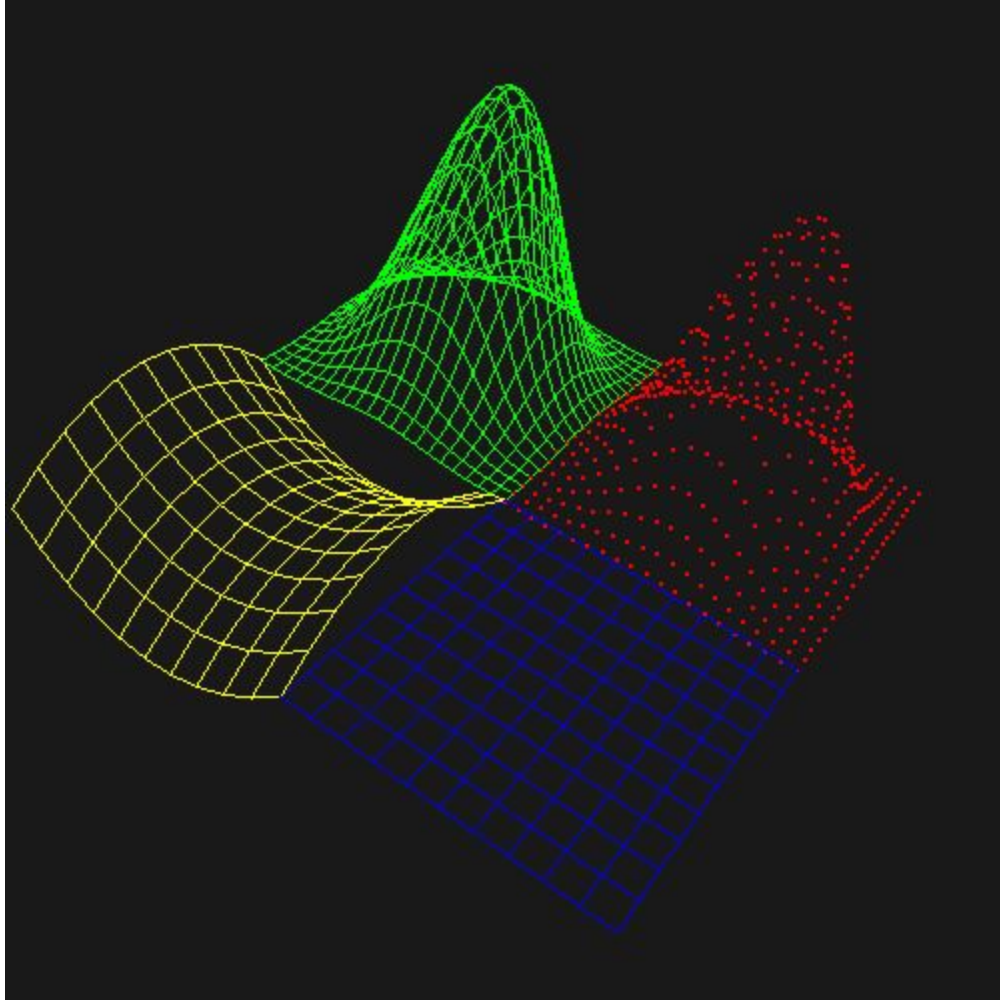


Figure: EIGHT Grid rendered as points and lines.

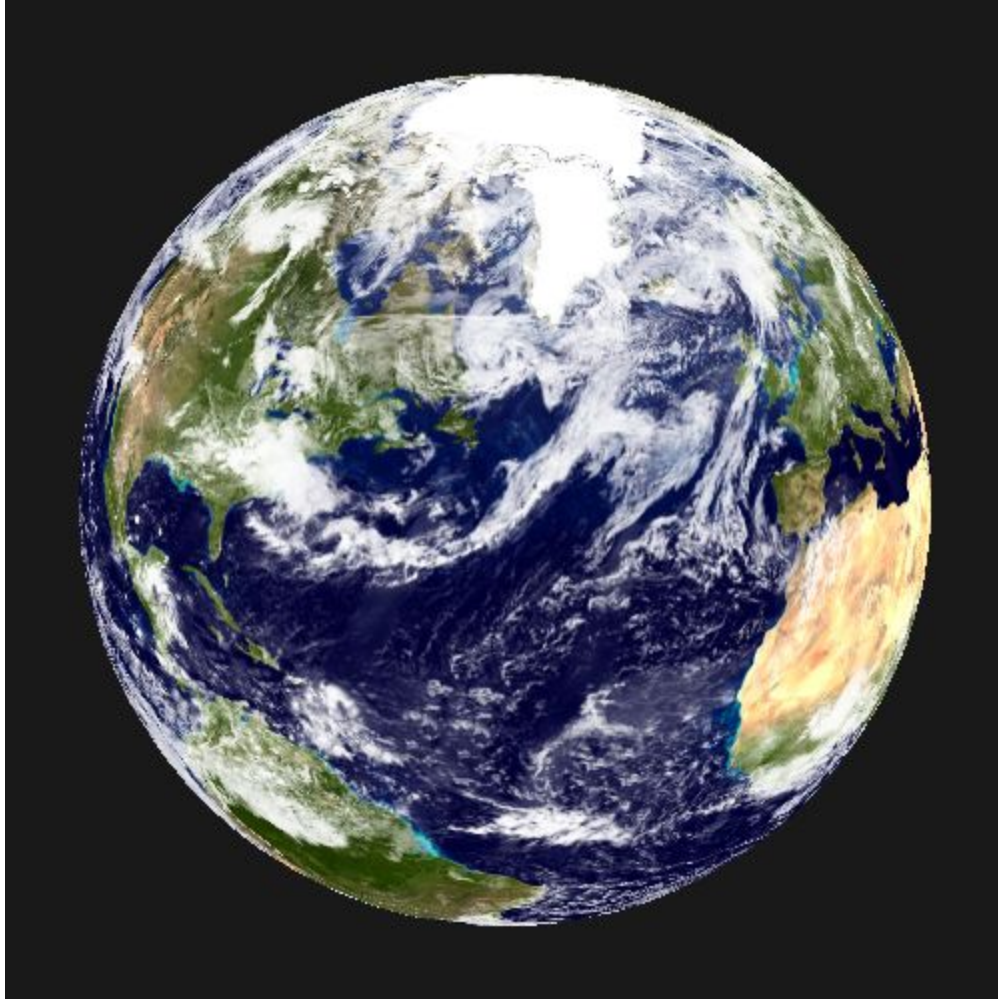


Figure: EIGHT texturing of the Earth applied to a sphere.

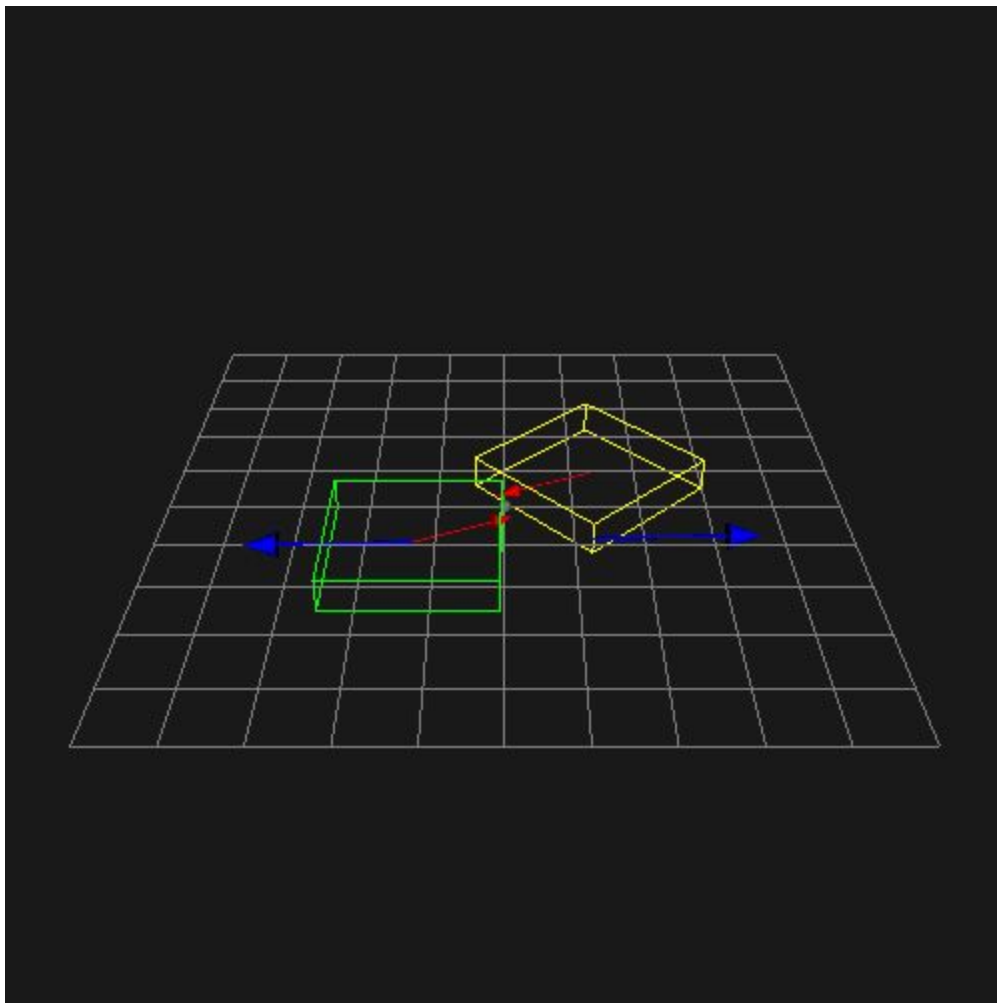
The numeric data types in EIGHT support operator overloading and immutable semantics making them suitable for mathematical computations by introductory students. At the same time these objects may be used in a mutable mode making them suitable for high-performance graphics. The numeric data types are unique in their support for Geometric Algebra.

NEWTON Physics Modeling Library

The NEWTON library is an open-source library that has been developed as an educational physics modeling library and is packaged with STEMCstudio for easy access and optimal integration. The NEWTON library was designed with two main purposes in mind:

- Easy and quick construction of accurate virtual simulations with the aim of challenging misconceptions of force concepts associated with the Force Concept Inventory (FCI).
- A collection of off-the-shelf components for the efficient construction of physical models.

The following figure shows part of the output from a NEWTON simulation that accurately models classical mechanics.



The NEWTON library consists of six main parts which may be used together or in isolation:

- Physics engine
- Rigid Body models and Interaction Laws
- Differential Equation Solvers
- Real-time graphing
- S.I. Units of Measure and Dimensions
- Geometric Algebra multivector objects for 3D with optional units.

The following output demonstrates the use of the multivector library with integrated units of measure:

```
g => -9.81*e3 m/s ** 2
g.direction() => -e3
g.magnitude() => 9.81 m/s ** 2
g.uom => m/s ** 2

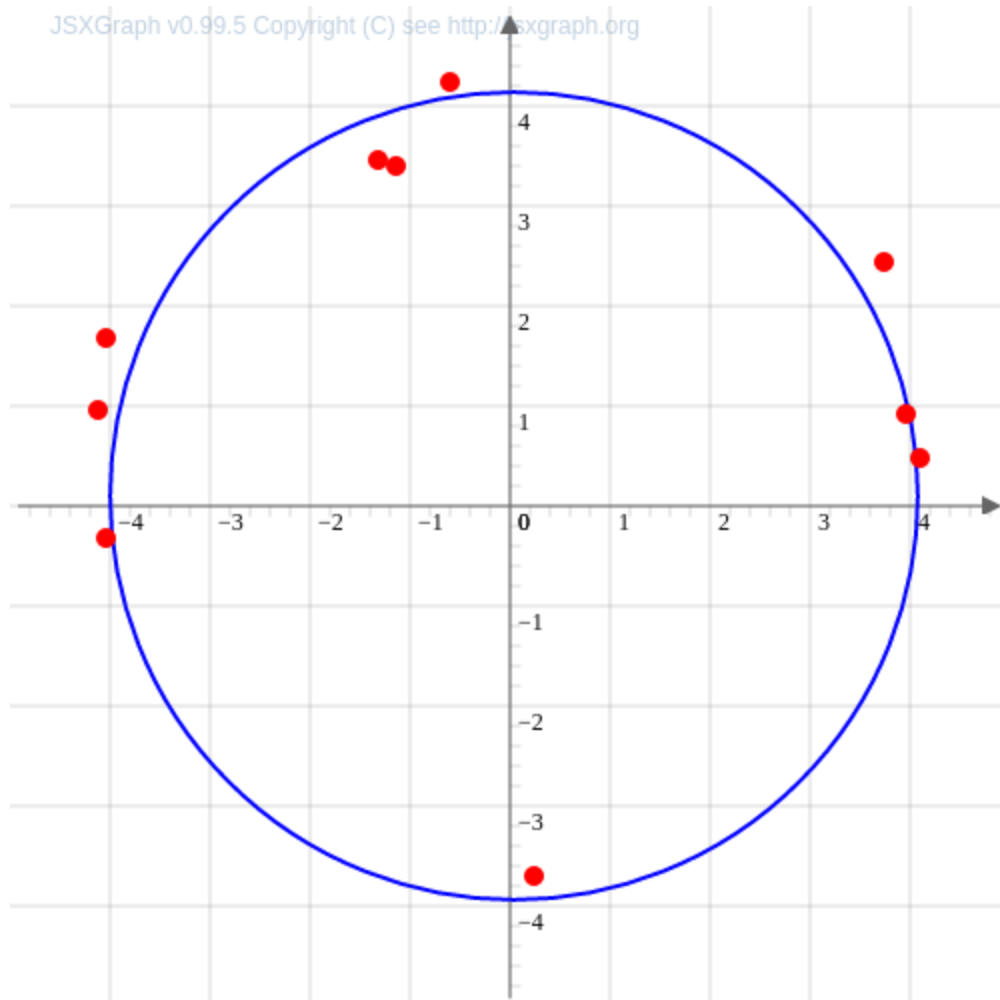
mass => 70 kg
mass.direction() => 1
mass.magnitude() => 70 kg

F = mass * g => -686.70*e3 N
F.direction() => -e3
F.magnitude() => 686.7 N

d => -2*e3 m
W = F << d => 1373.40 J
```

JSXGraph 2D Geometry Library

The JSXGraph library is a powerful 2D geometry library. It provides features similar to the popular Geogebra tool, but is a library that can be controlled by scripting. JSXGraph has been packaged with STEMCstudio for ease of use. The following diagram shows the output from a program that fits a circle to a set of points.



JSXGraph was developed by the University of Bayreuth.

