

令和4年度 学士論文

物理系を定義できるシミュレータ
SimSym の提案

東京工業大学 情報理工学院 数理・計算科学系
学籍番号 18B04657

木内 康介

指導教員
増原 英彦 教授

令和5年2月6日

概要

高等学校での物理学の授業では、実験は重要である。しかし実際は、生徒全員が実験を経験しているわけではない。理由としては、実験用の装置の準備や測定が難しいことや、実験を行うのに時間を要することが考えられる。そこで実験の代替として近年利用されているのが、物理実験のシミュレータである。シミュレータを用いることで、実験と同様の学習効果を得ることができる。

しかし、既存のシミュレータでは不十分な点が存在する。理論を学習する上では物理法則やそれを前提とした運動を方程式を通して学習する。一方、既存のシミュレータでは速度や位置のような物理量を数値でしか確認できず、描画されている物理系がどのような方程式によって表現されているのかわからない。

そこで本研究では、物理系を定義できるシミュレータである SimSym (Simulation with Symbols) を提案する。学習者は SimSym で系内の物体をそのパラメータとともに定義し、その物体の運動を表す方程式を立式する。この際学習者は、次元の異なる物理量の和が存在する不正な方程式を立式すると警告されるなど、正しい物理系を作成するための補助を受ける。シミュレーションを実行すると、定義した物理系に基づいて数値計算がなされ、物体の運動が可視化される。さらに、SimSym には現実の運動を正しく表現した動作例が存在し、学習者が参考にすることができる。

SimSym の利点は、現実の運動を確認できるだけでなく、学習者が定義した方程式と現実の物理法則に従う物体の運動の対応を理解できるという点である。既存のシミュレータはツール側が物理系を全て提供し、学習者はパラメータを指定するだけである。一方 SimSym では、学習者は動作例と比較しながら物理系を定義することで、現実の運動がどのような方程式で表されるものなのか理解できる。

謝辞

本研究を進めるにあたり、増原英彦教授、叢悠悠助教を始めとした増原研究室の皆様に多くのアドバイスやご指導をいただきました。また、lively.next 開発者の Robin, Linus, Jens にも助けていただきました。本論文は以上の方々のご支援がなければ存在しませんでした。この場を借りて感謝申し上げます。

目 次

第 1 章 はじめに	1
第 2 章 SimSym の構成	3
第 3 章 実装の方針	6
3.1 lively.next	6
3.2 SymPy	6
3.3 Pyodide	8
3.4 シミュレーションの実行	9
第 4 章 まとめと課題	10

第1章 はじめに

高等学校での物理学の授業では、実験は重要である。Holubova [1] は、実験室での作業は理論的な概念を検証する最も重要な方法であり、生徒は実験を通してどのような現象が起きるかを確認することができると述べている。

しかし実際は、生徒全員が実験を経験しているわけではない。林らは、2014 年に大学生を対象に物理実験の経験を調査した [2]。これによると、力学分野で最も基本的な「運動の法則」に関する実験経験は 60% であった。また、斜方投射の基本的な問題である「モンキーハンティング (1.1)」に関する実験は 10% に満たない。この理由としては、実験用の装置の準備や測定が難しいことや、実験を行うのに時間を要することが考えられる。

そこで実験の代替として近年利用されているのが、物理実験のシミュレータである。シミュレータを用いることで、実験と同様の学習効果を得ることができる。Ajredini [3] は、既存の物理実験シミュレータである PhET [4] を用いる授業と実際に実験を行う授業を実施し、テストを行った。その結果、シミュレーションによって得られる知識と実際の実験によって得られる知識の間には有意な差がないと結論づけた。

しかし、既存のシミュレータでは不十分な点が存在する。学習者が理論を学習するとき、物理法則やそれを前提とした運動を方程式を通して学習する。図 1.2 は実際に生徒が解く問題の例である。学習者は教科書等で学習した物理法則を方程式の形で利用し、結果も方程式の形で表現される。一方既存のシミュレータである PhET では、図 1.3 のように速度や質量、位置のような物理量を数値でしか確認できず、描画されている物理系がどのような方程式によって表現されているのかわからない。

そこで本研究では、物理系を定義できるシミュレータである SimSym (Simulation with Symbols) を提案する。学習者は SimSym で系内の物体をそのパラメータとともに定義し、その物体の運動を表す方程式を立式する。この際学習者は、次元の異なる物理量の和が存在する不正な方程式を立式すると警告されるなど、正しい物理系を作成するための補助を受ける。シミュレーションを実行すると、定義した物理系に基づいて数値計算がなされ、物体の運動が可視化される。さらに、SimSym には現実の運動を正しく表現した動作例が存在し、学習者が参考にすることができる。

SimSym の利点は、現実の運動を確認できるだけでなく、学習者が定義した方程式と現実の物理法則に従う物体の運動の対応を理解できるという点である。既存のシミュレータはツール側が物理系を全て提供し、学習者はパラメータを指定するだけである。一方 SimSym では、学習者は動作例と比較しながら物理系を定義することで、現実の運動がどのような方程式で表されるものなのか理解できる。

なお、SimSym は実際にシミュレーションを表示する部分以外は未実装であるが、2 章で説明するアイデアは 3 章に記す方針で実装が可能であると考えている。

本論文の構成は以下の通りである。第 2 章で、SimSym のアイデアを説明する。第 3 章で、SimSym の実装について説明する。第 4 章で、まとめと今後の展望について述べる。

小球を、位置 $(0, 0)$ から初速 v_0 、仰角 θ で発射したところ、位置 (l, h) から自由落下してくる物体に衝突した。 $\tan \theta$ の満たすべき条件を求めよ。ただし、重力加速度の大きさを g とする。

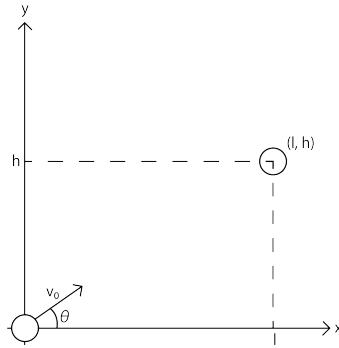


図 1.1: モンキーハンティング

(問題) 静止している質量 m の物体に大きさ F の力をかけ続ける。 t 秒後の速さを求めよ。

(解答) 物体の加速度を a とする。運動方程式より $ma = F \therefore a = \frac{F}{m}$ 。等加速度運動の公式より $v = at = \frac{F}{m}t$
答え: $\frac{F}{m}t$

図 1.2: 方程式の計算例

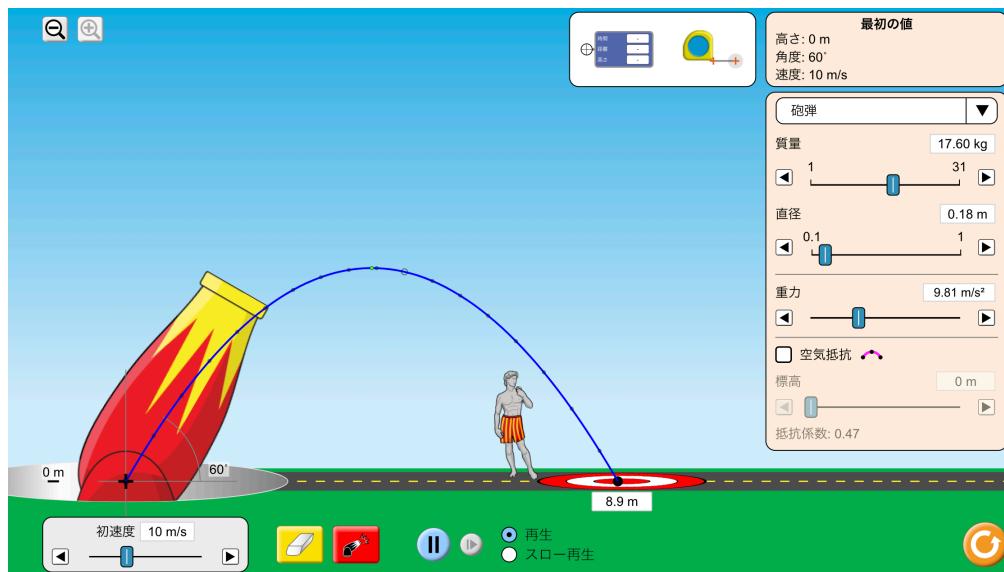


図 1.3: PhET のシミュレーション例

第2章 SimSym の構成

SimSym の画面は、図 2.1 のように左半分の物理系定義ペインと右半分の観測ペインに分かれている。物理系定義ペインで物体の作成や方程式の定義を行い、観測ペインでシミュレーションの結果を確認する。

物理系定義ペイン

学習者は、物理系定義ペインで以下の操作を行うことで、物理系を定義することができる。

- 物体の作成:** 学習者が物体名を入力すると、SimSym はその物体に紐付いた物理量に対応する次元付き変数 ($m_A[M]$, $x_A[L]$ など¹) を生成する。
- 方程式の立式:** 学習者は、1. で生成された変数を使って方程式を立式する。この際、次元付き変数は自由に追加することができる。なお、時刻を表す変数 $t[T]$ や重力加速度 $g[L/T^2]$ などの物理定数は物体に紐付かない変数として用意されている。また、不正な次元の方程式は警告される。

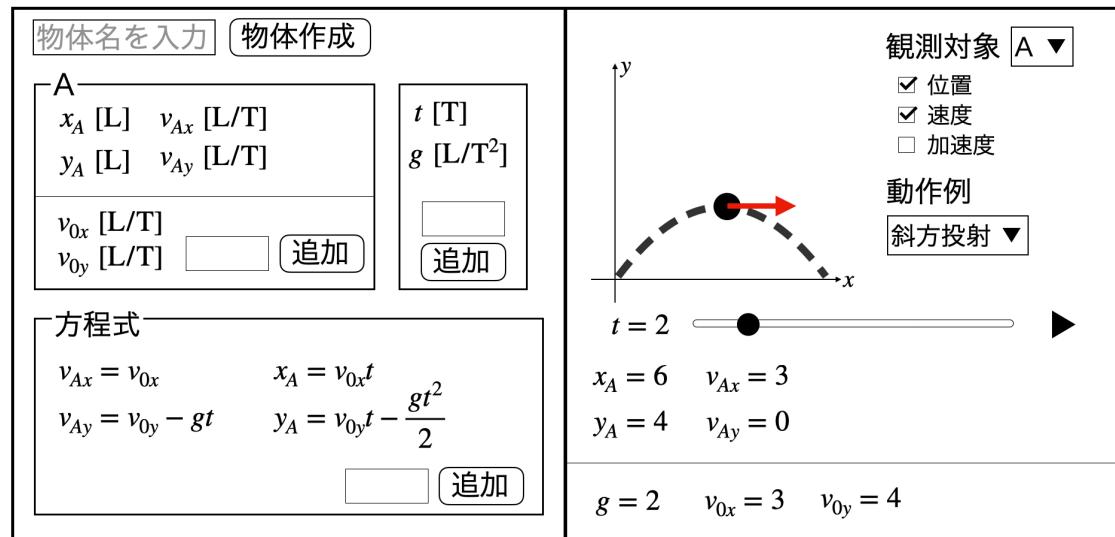


図 2.1: SimSym 上で斜方投射を表現した例

¹M: Mass(質量), L: Length(長さ), T: Time(時間)などを次元と呼ぶ。例えば速度の次元は [L/T] と表される。

観測ペイン

物理系の定義後、以下の操作を行うことで観測ペインに情報が描画される。

3. **観測対象の指定:** 学習者は観測したい物体とその物理量(位置・速度・加速度)を指定する。
4. **初期値の入力:** 観測対象を指定した際に、その値を計算するために必要な変数が観測ペイン下部に表示される。これらに値を入力する。
5. **シミュレーションの再生:** シミュレーションを再生すると、時刻 t が変化しながら観測対象の物理量が描画される。物体の位置は座標平面上の位置として、速度と加速度はベクトルとして描画される。
6. **動作例の選択(オプション):** 学習者は現実の運動を正しく表現した動作例を選択することができる。動作例と学習者が定義した物理系の運動を比較することで、現実と同じ運動を表現しているか確認することができる。

以下では具体例を見ていく。図 2.1 は、SimSym 上で x 軸方向の初速が v_{0x} , y 軸方向の初速が v_{0y} , 重力加速度の大きさが g であるような斜方投射を表現した例である。

1. 物体 A を作成すると、 x_A , y_A , v_{Ax} , v_{Ay} が生成される。初速に対応する変数 v_{0x} , v_{0y} を物体 A に追加する。
2. 1. で用意した変数を用いて方程式を立式する。
3. 観測対象の物体として A を選択し、位置と速度を選ぶ。
4. v_{Ax} , v_{Ay} , x_A , y_A を計算するのに必要な g , v_{0x} , v_{0y} に値を入力する。
5. シミュレーションを再生すると、位置と速度が描画される。
6. 動作例として斜方投射を選択し、観測ペインに破線で表示された正しい軌道と比較する。

ここで、方程式の立式の際に $v_{0x} + t$ ($[L/T] + [T]$) のように次元の一致していない式を定義しようとすると、図 2.2 のように警告される。また、重力加速度の向きを間違え $v_{Ay} = v_{0y} + gt$, $y_A = v_{0y}t + \frac{gt^2}{2}$ のように定義すると、図 2.3 の実線のような軌道を描いて運動するが、これは動作例の破線と大幅に異なる。そのため、これは現実の運動を正しく表せていないことがわかる。

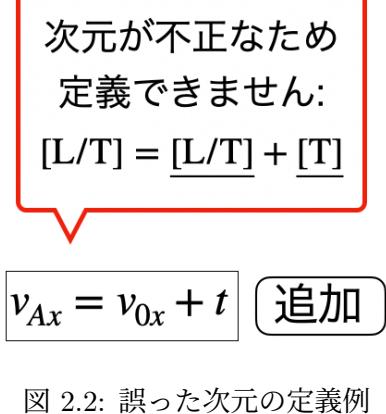


図 2.2: 誤った次元の定義例

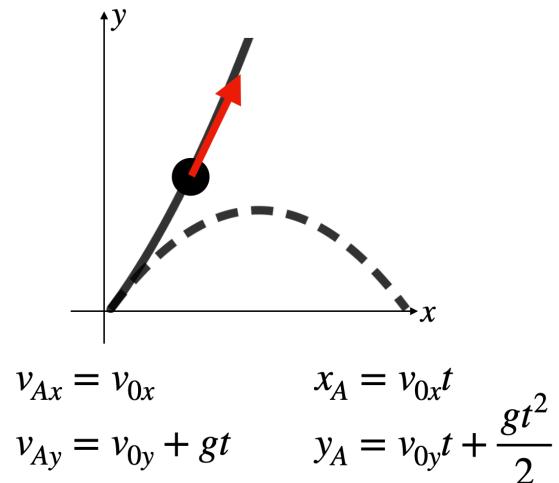


図 2.3: 誤った運動の定義例

第3章 実装の方針

フロントエンドに lively.next¹ を、方程式の処理と数値計算に SymPy [5] を用い、JavaScript と Python を組み合わせて実装する。また、Pyodide² を用いることで、ブラウザ上で全ての計算を行う。

3.1 lively.next

lively.next は Lively [6] から派生したプロジェクトで、ブラウザ上で GUI アプリケーションを作成・実行することができる Web プログラミング環境である。これを用いて、学習者が物体や方程式を定義する物理系定義ペインと、シミュレーション結果を表示する観測ペインを実装する。ブラウザ上で利用できるため、学習者はソフトウェアのインストールなどをする必要がなく、簡単に利用できる。

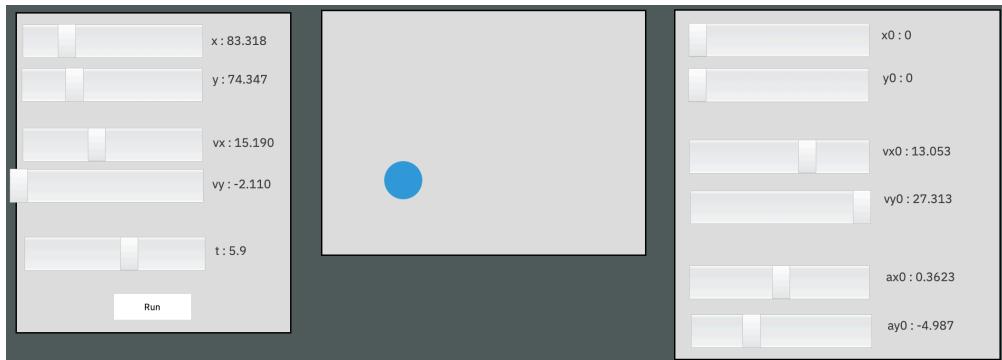


図 3.1: lively.next を用いて実装されたシミュレーション結果の表示画面

3.2 SymPy

SymPy は、記号計算のための Python ライブリaryである。次元付き変数を作成し、方程式の処理や数値計算を行うことができる。図 3.2 は、実際に SymPy で斜方投射を表す物理系を定義する例である。

SymPy は次元を扱うことはできるのだが、次元が異なる値同士も足すことができてしまう。図 3.3 では、 t に次元 $[T]$ を、 g に次元 $[T/S^2]$ を割り当てたが、 $t + g$ が計算できてしまう。そのため、このような不一致を検出する仕組みを実装する必要がある。

また、SymPy は Python のライブリaryであるが、後述する Pyodide を用いることでブラウザ上で実行することができる。

¹<https://lively-next.org>

²<https://pyodide.org/en/stable/>

```
In [ ]:
from IPython.display import display, Math
from sympy import Eq, solve, latex
from sympy.physics.units.systems import SI
from sympy.physics.units import Quantity, length, mass, time
```

```
In [ ]:
# デフォルトで用意されている変数 t, g と
# 物体 A に紐付く変数 x_A, y_A, v_{Ax}, v_{Ay} を生成
t = Quantity('t', latex_repr='t')
SI.set_quantity_dimension(t, dimension=time)
g = Quantity('g', latex_repr='g')
SI.set_quantity_dimension(g, dimension=length/time**2)
xA = Quantity('x_A', latex_repr='x_A')
SI.set_quantity_dimension(xA, dimension=length)
yA = Quantity('y_A', latex_repr='y_A')
SI.set_quantity_dimension(yA, dimension=length)
vAx = Quantity('vAx', latex_repr='v_{Ax}')
SI.set_quantity_dimension(vAx, dimension=length/time)
vAy = Quantity('vAy', latex_repr='v_{Ay}')
SI.set_quantity_dimension(vAy, dimension=length/time)
```

```
In [ ]:
# v_{0x} と v_{0y} を生成
v0x = Quantity('v0x', latex_repr='v_{0x}')
SI.set_quantity_dimension(v0x, dimension=length/time)
v0y = Quantity('v0y', latex_repr='v_{0y}')
SI.set_quantity_dimension(v0y, dimension=length/time)
```

```
In [ ]:
# 各変数が満たすべき方程式を作成
eqs = [Eq(vAx, v0x), Eq(vAy, v0y - g * t), Eq(xA, v0x * t), Eq(yA, v0y * t - g * t**2 / 2)]
for eq in eqs: display(eq)
```

$$\begin{aligned}v_{Ax} &= v_{0x} \\v_{Ay} &= -gt + v_{0y} \\x_A &= tv_{0x} \\y_A &= -\frac{gt^2}{2} + tv_{0y}\end{aligned}$$

```
In [ ]:
# 初期値を代入
param = {t: 2, g: 2, v0x: 3, v0y: 4}
for eq in eqs: display(Math(f'{latex(eq.lhs)} = {latex(eq.rhs.subs(param))}'))
```

$$\begin{aligned}v_{Ax} &= 3 \\v_{Ay} &= 0 \\x_A &= 6 \\y_A &= 4\end{aligned}$$

図 3.2: SymPy を用いて斜方投射を表す物理系を定義する例

```
In [ ]: from sympy.physics.units.systems import SI
from sympy.physics.units import Quantity, length, time
from sympy.physics.units.systems.si import dimsys_SI
t = Quantity('t', latex_repr='t')
SI.set_quantity_dimension(t, dimension=time)
g = Quantity('g', latex_repr='g')
SI.set_quantity_dimension(g, dimension=length/time**2)

foo = t + g
foo
```

```
Out[ ]: g + t
```

図 3.3: SymPy 上で単位が異なる値同士を足す例

3.3 Pyodide

Pyodide は、Mozilla が開発している WebAssembly で実装された CPython 処理系である。ブラウザ上でピュアな Python コードを実行できる。また、Python で書かれたパッケージと C 言語で書かれた一部のパッケージに対応していて、SymPy は Pyodide で実行可能である。実際に Pyodide を実行する例を図 3.4 に記す。

```
1 <!DOCTYPE HTML>
2 <head>
3   <script src="https://cdn.jsdelivr.net/pyodide/v0.22.0/full/pyodide.js"
        ></script>
4   <script>
5     async function main() {
6       let pyodide = await loadPyodide();
7       let result = pyodide.runPython(`
8 import js
9 result = js.document.getElementById("result")
10 x = 2
11 y = 10
12 result.textContent += str(x ** y)
13       `);
14     }
15   </script>
16 </head>
17 <body>
18   <script>main()</script>
19   <p id="result">結果: </p>
20 </body>
```

結果: 1024

図 3.4: Pyodide を利用する例

3.4 シミュレーションの実行

シミュレーションの実行時は、設定された時刻 t の範囲を粒度 Δt ずつ変化させる。各方程式に各 t を代入した結果を `lively.next` が受け取り、描画する。シミュレーションのリアルタイム性を確保するため、各 t を方程式に代入した値は方程式の定義時・変数の値の変更時にあらかじめ計算する。

```
In [ ]:
from IPython.display import display, Math
from sympy import Eq, solve, latex, Rational
from sympy.physics.units.systems import SI
from sympy.physics.units import Quantity, length, mass, time

In [ ]:
t = Quantity('t', latex_repr='t')
g = Quantity('g', latex_repr='g')
v0x = Quantity('vAx', latex_repr='v_{Ax}')
v0y = Quantity('vAy', latex_repr='v_{Ay}')
xA = v0x * t
yA = v0y * t - g * t**2 / 2

In [ ]:
param = {g: 2, v0x: 3, v0y: 4}
xAs = xA.subs(param)
yAs = yA.subs(param)

max_t = 4
dt_denom = 10
dt = Rational(1, dt_denom)
xA_list = [xAs.subs(t, i * dt) for i in range(dt_denom * max_t + 1)]
yA_list = [yAs.subs(t, i * dt) for i in range(dt_denom * max_t + 1)]

In [ ]:
import matplotlib.pyplot as plt
plt.scatter(xA_list, yA_list)
plt.show()
```

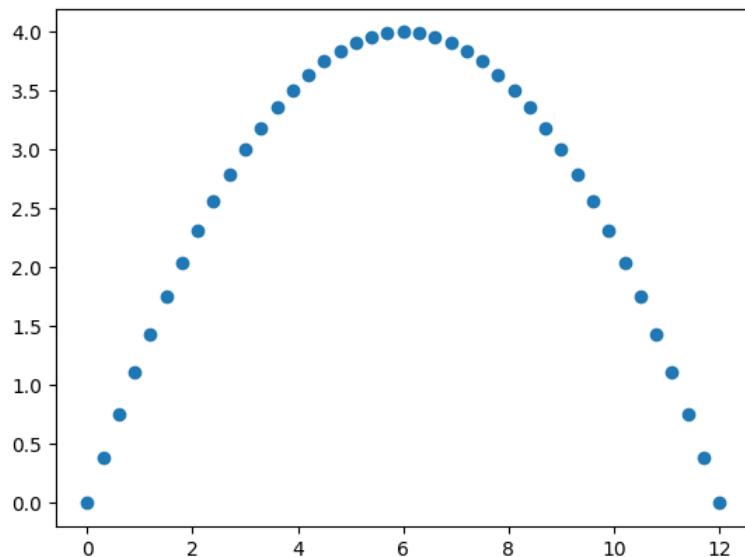


図 3.5: シミュレーション実行のための数値計算例

第4章　まとめと課題

本研究では、物理系を定義できるシミュレータ SimSym を提案した。学習者は SimSym を用いることで、自身が定義した物理系の運動と現実の運動との対応を視覚的に確認することができる。

今後は、まず実装を進めることが課題である。実装が完成したら、SimSym の教育効果を評価する必要がある。評価手法として、Hake [7] が導入した normalized gain を用いた実験を検討する。

参考文献

- [1] Renata Holubova. The impact of experiments in physics lessons – “why, when, how often?”. *AIP Conference Proceedings*, Vol. 2152, No. 1, p. 030007, 2019.
- [2] 林 壮一, 川村 康文, 村上 聰. 大学生に対する高校物理実験および放射線学習の現状調査. 物理教育, Vol. 63, No. 3, pp. 191–196, 2015.
- [3] Fadil Ajredini, Nese Izairi, and Oliver Zajkov. Real Experiments versus Phet Simulations for Better High-School Students ’ Understanding of Electrostatic Charging. *European Journal Of Physics Education*, Vol. 5, No. 1, p. 59, February 2014.
- [4] Katherine Perkins, Wendy Adams, Michael Dubson, Noah Finkelstein, Sam Reid, Carl Wieman, and Ron LeMaster. PhET: Interactive Simulations for Teaching and Learning Physics. *The Physics Teacher*, Vol. 44, No. 1, pp. 18–23, January 2006.
- [5] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. SymPy: symbolic computing in Python. *PeerJ Computer Science*, Vol. 3, p. e103, January 2017. Publisher: PeerJ Inc.
- [6] Daniel Ingalls, Krzysztof Palacz, Stephen Uhler, Antero Taivalsaari, and Tommi Mikkonen. The lively kernel a self-supporting system on a web page. In Robert Hirschfeld and Kim Rose, editors, *Self-Sustaining Systems*, pp. 31–50, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [7] Richard Hake. Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American Journal of Physics - AMER J PHYS*, Vol. 66, , 01 1998.