# Assignment 2

COMP SCI 2ME3 and SFWR ENG 2AA4

March 3, 2021

## 1 Dates and Deadlines

**Assigned:** February 1, 2021

**Part 1:** February 12, 2021

**MSAF/SAS Deadline:** February 16, 2021

**Receive Partner Files:** February 17, 2021

**Part 2:** February 25, 2021

**Last Revised:** March 3, 2021

All submissions are made through git, using your own repo located at:

`https://gitlab.cas.mcmaster.ca/se2aa4_cs2me3_assignments_2020/[macid].git`

where `[macid]` should be replaced with your actual macid. The time for all deadlines is 11:59 pm. If you notice problems in your Part 1 `*.py` files after the deadline, you should fix the problems and discuss them in your Part 2 report. However, the code files submitted for the Part 1 deadline will be the ones graded.

## 2 Introduction

The purpose of this software design exercise is to write a Python program that follows the given formal specification. The given specification is for simulating the physics of a scene where a shape moves through 2D space. Simulations of this sort are used for animations and video game physics.

The simulation is based on the familiar equation for Newton's second law:

$$F = m\frac{d^2r}{dt^2} = m\frac{dv}{dt} = ma$$

where $F$ is the unbalanced force, $m$ is the mass, $r$ is the position of the body, $v$ is the velocity of the body and $a$ is the acceleration of the body. The motion of the body is calculated about the centre of mass (cm). Figure 1 provides an example of the possible output generated by the modules in this assignment.

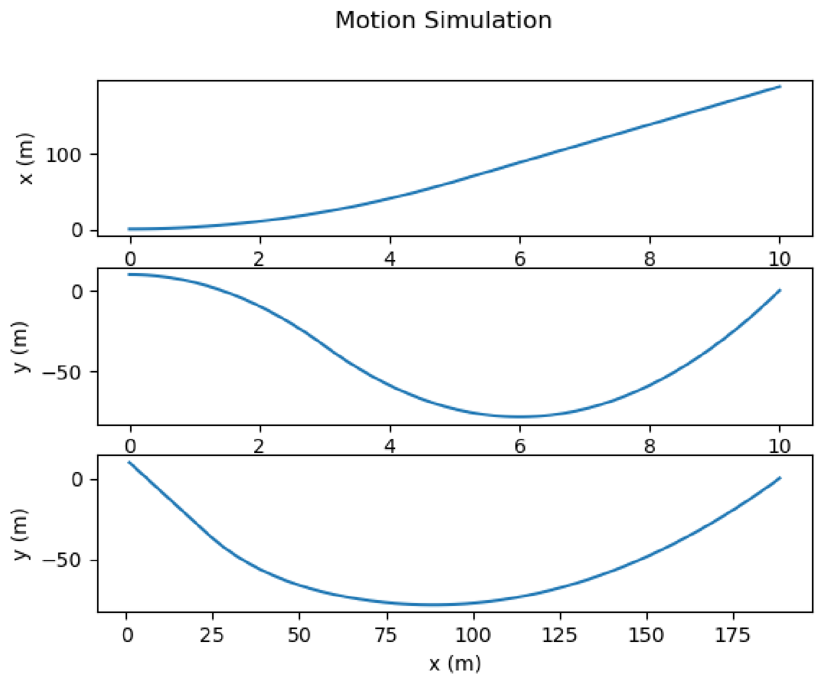

Figure 1: Motion simulation for an object shown as plots of $x$ vs $t$, $y$ vs $t$ and $y$ vs $x$

For those that are interested, additional information can be found in the following papers: Physics, The Next Frontier and Physics, Part 2: Angular Effects.

As for the previous assignment, you will use doxygen, make, LaTeX and Python (version 3). In addition, this assignment will use pytest for unit testing and flake8 to verify that its pep8-inspired standard is enforced. This assignment also takes advantage of functional programming in Python.

All of your code, except for the testing files, should be documented using doxygen. Using doxygen on the testing files is optional. Your report should be written using LaTeX. Your code should follow the given specification exactly. In particular, you should not add public methods or procedures that are not specified and you should not change the number or order of parameters for methods or procedures. If you need private methods or procedures, please use the Python convention of naming the files with the double underscore (`__methodName__`) (dunders). **Please follow specified naming EXACTLY. You do not want to lose marks for a simple mistake.**

For the purpose of understandability, the specification provided at the end of the assignment uses notation that is slightly different from the Hoffman and Strooper notation. Specifically the types for real and natural numbers are represented by $\mathbb{R}$ and $\mathbb{N}$, respectively. (In this specification, the natural numbers are assumed to include 0.) Booleans are represented by $\mathbb{B}$. Also, subscripts are used for indexing a sequence. For instance, $x_i$ means the same thing as $x[i]$. An overview of our mathematical notation and MIS templates can be found in the MIS Format document.

## 2.1 Installing scipy and matplotlib

This assignment uses two external modules that we have not previously installed: scipy and matplotlib. To install them on your virtual machine, do the following:

```
pip3 install scipy
pip3 install matplotlib
```

## 2.2 Installing flake8

We will use flake8 to ensure your Python code meets the style conventions of the course. You have likely already installed flake8 on your VM or local machine. If not, you will need to install two 'pip' packages. This is the standard way to install packages for Python.

First run the following command in your terminal:

```
pip --version
```

If the output includes 'Python 3,' then run the following instructions:

```
pip install flake8
pip install pep8-naming
```

**If it does not,** then use `pip3` for the following alternate instructions.

```
pip3 install flake8
pip3 install pep8-naming
```

### 2.3 Running flake8

Run the following command in your A2 directory. This will inform you of the location and types of style violations. You can find more information on what each error means here: https://lintlyci.github.io/Flake8Rules/

```
flake8
```

## Part 1

## Step 1

Write the modules `Shape.py`, `CircleT.py`, `TriangleT.py`, `BodyT.py`, `Scene.py`, `Plot.py` following the specification given at the end of the assignment.

## Step 2

Write a module (named `test_driver.py`), using pytest, that tests the following modules: `CircleT.py`, `TriangleT.py`, `BodyT.py` and `Scene.py`. (`Plot.py` should be tested manually.) For `Scene.py` you do not have to test the getters or setters for the functions $F_x$ and $F_y$. The given makefile `Makefile` has a rule `test` for running your tests. Each procedure/method should have at least one test case. Record your rationale for test case selection and the results of using this module to test the procedures in your modules. (You will submit your rationale with your report in Step 6.) Please make an effort to test normal cases, boundary cases, and exception cases. Your test program should compare the calculated output to the expected output and provide a summary of the number of test case that have passed or failed.

Testing the output of the `sim` method from `Scene.py` is made challenging because the output is sequences of real numbers. Use the following relative error formula when comparing a calculated sequence $x_{\text{calc}}$ to the true solution $x_{\text{true}}$:

$$\frac{||x_{\text{calc}} - x_{\text{true}}||}{||x_{\text{true}}||} < \epsilon \tag{1}$$

Subtracting two sequences is done by subtracting corresponding elements to create a new sequence. The double vertical bars around a sequence mean taking the norm of the sequence, where the norm is a measure of the magnitude of the sequence. There are many different possible norms. The simplest is to find the maximum absolute value in the sequence. The two sequences are considered close to being equal if the left hand side of the equation is less than some small number $\epsilon$. You are free to select a value for $\epsilon$ that you feel is reasonable.

In testing you may want to consider some common cases, as follows:

- The shape falling under the force of gravity in the negative $y$ direction: $F_x(t) = 0$, $F_y(t) = -gm$, $v_x = 0$, $v_y = 0$.
- Projectile or ballistics motion: $F_x(t) = 0$, $F_y(t) = -gm$, $v_x = v\cos(\theta)$, $v_y = v\sin(\theta)$ where $v$ is the magnitude of the initial velocity and $\theta$ is the angle from the horizontal.

To get more interesting plots, you may want to experiment with conditional expressions, like that shown in `test_expt.py`. You can run this file via the make rule (`make expt`).

## Step 3

Test the supplied `Makefile` rule for `doc`. This rule should compile your documentation into an html and LaTeX version. Along with the supplied `Makefile`, a doxygen configuration file is also given in your initial repo. You should not have to change these files.

## Step 4

Submit (add, commit and push) all Python files using git. (Of course, you will be doing this throughout the development process. This step is to explicitly remind you that the version that will be graded is the one we see in the repo.) Please **do not change** the names and locations for the files already given in your git project repo. For Part 1, the only files that you should modify are the Python files.

Changing other files could result in a serious grading penalty, since the TAs might not be able to run your code and documentation generation. You should NOT submit your generated documentation (html and latex folders). In general, files that can be regenerated are not put under version control.

Optionally, you can tag your final submission of Part 1 of the assignment with the label `A2Part1`.

## Part 2

Your `CircleT.py`, `TriangleT.py`, `BodyT.py` and `Scene.py` files will automatically be pushed to your partner's repo. **Including your name in your partner code files is optional.**

## Step 5

After you have received your partner's files, temporarily replace your corresponding files with your partner's. (You are just moving the files for the purpose of testing; you will not replace your version of the code in the git repo.) Do not initially make any modifications to any of the code. Run your test module and record the results. Your evaluation for this step does not depend on the quality of your partner's code, but only on your discussion of the testing results. If the tests fail, for the purposes of understanding what happened, you are allowed to modify your partner's code.

## Step 6

Write a report using LaTeX (`report.tex`) following the template given in your repo. The report should include the following:

1. Your name and macid.
2. Your updated Python files.
3. Your partner's files.
4. The results of testing your files (along with the rational for test case selection). The summary of the results should consist of the following: the number of passed and failed test cases, and brief details on any failed test cases.
5. The results of testing your files combined with your partner's files. Please include a discussion of the test results and what you learned doing the exercise. List any problems you found with your program and/or your partner's module. Compare the results of this exercise for A1 and A2.
6. The specification of this assignment imposed design decisions on you. Please provide a critiques of the specified design. What did you like? What areas need improvement? How would you propose changing the design? Your answer should incorporate the quality criteria we discussed in class, including consistency, essentiality, generality, minimality, high cohesion, low coupling and opacity. Does the interface provide the programmer with checks that will allow them to avoid generating an exception?
7. Answers to the following questions:

   a) Should getters and setters be unit tested? Please justify your answer.

b) The assignment says that you do not have to test the setters or getters that are functions ($F_x$ and $F_y$) in `Scene.py`. If you were required to test the getters and setters for these state variables, how might you do that?

c) The assignment does not require automated tests for `Plot.py`. If automated tests were required, how might you do them? Hint: matplotlib can generate a file for any plots that you build.

d) Write a mathematical specification for a function called close_enough that takes two arguments $x_{\text{calc}}$ and $x_{\text{true}}$ and returns true if the sequences represented by the arguments are close to being equal, as expressed by Equation 1.

e) The given specification has exceptions for non-positive values of shape dimensions and mass, but not for the $x$ and $y$ coordinates of the centre of mass. Should there be exceptions for negative coordinates? Why or why not?

f) TriangleT has a state invariant that $s > 0 \wedge m > 0$. Informally prove that this invariant is always satisfied by the given specification.

g) Write a Python list comprehension statement that generates a list of the square roots of all odd integers between 5 and 19 (inclusive).

h) Define a Python function that takes a string and returns the string, but with all upper case letters removed.

i) How are the principles of abstraction and generality related?

j) If we have high coupling between modules we can have a cases where a module *uses* many other modules or a module is *is used* by many other modules. Which of these two scenarios would in general be better? Why?

The writing style for the report should be professional, but writing in the first person is fine. Some of your ideas can be summarized in lists, but most of the report should be written in full sentences. Spelling and grammar is important.

Commit and push `report.tex` and `report.pdf`. Although the pdf file is a generated file, for the purpose of helping the TAs, we'll make an exception to the general rule of avoiding version control for generated files. If you have made any changes to your Python files, you should also push those changes.

Optionally, the final submission can have the tag `A2Part2`.

## Notes

1. Your git repo will be organized with the following directories at the top level: `A1`, `A2`, `A3`, and `A4`.

2. Inside the `A2` folder you will start with initial stubs of the files and folders that you need to use. Please do not change the names or locations of any of these files or folders.

3. Please put your name and macid at the top of each of your source files, except for those that you share with a partner. Including your name and macid is optional for those files.

4. Your program must work on mills when compiled with its versions of Python (version 3), LaTeX, doxygen, make, pytest, pytest-cov, and flake8. (Since mills does not provide a gui, `Plot.py` does not have to be tested on mills.)

5. Python specifics:

    - The exceptions in the specification are intentionally selected to use the names of existing Python exceptions. There is no need to define new exceptions.
    - Although efficient use of computing resources is always a good goal, your implementation will be judged on correctness and not on performance.
    - Since the specification is silent on this point, for methods that return an object, or use objects in their state, you can decide to either use references or construct new objects. The implementation will be easier if you just work in terms of references to objects.
    - For the interface modules, use Python's Abstract Base Class (ABC). An illustrative example of ABC is given at `https://ca.godaddy.com/engineering/2018/12/20/python-metaclasses/`.
    - You should use the scipy library for odeint.
    - A sample program (`test_expt.py`) that uses the modules in this specification is available in the repo. You can use this to do an initial test that your interface matches the specification.
    - Exceptions are documented with `@throws`.
    - The marking scheme is available in the course repo. The marking scheme includes grades based on git usage, unit testing, and flake8.
    - If a flake8 check contradicts the use of a symbol used in the MIS specification, ignore this flake8 warning and continue to use the symbol as given in the MIS. A `.flake8` configuration file will be pushed to your repo that should ignore any pep8 errors that are allowed, such as using doxygen comments.

6. **Your grade will be based to a significant extent on the ability of your code to compile and its correctness. If your code does not compile, then your grade will be significantly reduced.**

7. **Any changes to the assignment specification will be announced in class. It is your responsibility to be aware of these changes. Please monitor all pushes to the course git repo.**

# Shape Interface Module

## Interface Module

Shape

## Uses

None

## Syntax

### Exported Constants

None

### Exported Types

None

### Exported Access Programs

| **Routine name** | **In** | **Out** | **Exceptions** |
|---|---|---|---|
| cm_x | | $\mathbb{R}$ | |
| cm_y | | $\mathbb{R}$ | |
| mass | | $\mathbb{R}$ | |
| m_inert | | $\mathbb{R}$ | |

## CircleT Module

### Template Module inherits Shape

CircleT

### Uses

Shape

### Syntax

#### Exported Constants

None

#### Exported Types

CircleT = ?

#### Exported Access Programs

| Routine name | In | Out | Exceptions |
|---|---|---|---|
| new CircleT | $x_s : \mathbb{R}$, $y_s : \mathbb{R}$, $r_s : \mathbb{R}$, $m_s : \mathbb{R}$ | CircleT | ValueError |
| cm_x | | $\mathbb{R}$ | |
| cm_y | | $\mathbb{R}$ | |
| mass | | $\mathbb{R}$ | |
| m_inert | | $\mathbb{R}$ | |

### Semantics

#### State Variables

$x : \mathbb{R}$
$y : \mathbb{R}$
$r : \mathbb{R}$
$m : \mathbb{R}$

#### State Invariant

$r > 0 \wedge m > 0$

**Assumptions**

The arguments provided to the access programs will be of the correct type.

**Access Routine Semantics**

new CircleT($x_s, y_s, r_s, m_s$):

- transition: $x, y, r, m := x_s, y_s, r_s, m_s$
- output: $out :=$ self
- exception: $(\neg(r_s > 0 \wedge m_s > 0) \Rightarrow \text{ValueError})$

cm_x():

- output: $out := x$
- exception: none

cm_y():

- output: $out := y$
- exception: none

mass():

- output: $out := m$
- exception: none

m_inert():

- output: $out := \frac{mr^2}{2}$
- exception: none

# TriangleT Module

## Template Module inherits Shape

TriangleT

## Uses

Shape

## Syntax

### Exported Constants

None

### Exported Types

TriangleT = ?

### Exported Access Programs

| **Routine name** | **In** | **Out** | **Exceptions** |
|---|---|---|---|
| new TriangleT | $x_s : \mathbb{R}$, $y_s : \mathbb{R}$, $s_s : \mathbb{R}$, $m_s : \mathbb{R}$ | TriangleT | ValueError |
| cm_x | | $\mathbb{R}$ | |
| cm_y | | $\mathbb{R}$ | |
| mass | | $\mathbb{R}$ | |
| m_inert | | $\mathbb{R}$ | |

## Semantics

### State Variables

$x : \mathbb{R}$
$y : \mathbb{R}$
$s : \mathbb{R}$ *#side length (equilateral triangle)*
$m : \mathbb{R}$

### State Invariant

$s > 0 \wedge m > 0$

### Assumptions

The arguments provided to the access programs will be of the correct type.

### Access Routine Semantics

new TriangleT($x_s, y_s, s_s, m_s$):

- transition: $x, y, s, m := x_s, y_s, s_s, m_s$
- output: *out* := self
- exception: $(\neg(s_s > 0 \wedge m_s > 0) \Rightarrow \text{ValueError})$

cm_x():

- output: *out* $:= x$
- exception: none

cm_y():

- output: *out* $:= y$
- exception: none

mass():

- output: *out* $:= m$
- exception: none

m_inert():

- output: *out* $:= \frac{ms^2}{12}$
- exception: none

## BodyT Module

### Template Module inherits Shape

BodyT

### Uses

Shape

### Syntax

#### Exported Constants

None

#### Exported Types

BodyT = ?

#### Exported Access Programs

| **Routine name** | **In** | **Out** | **Exceptions** |
|---|---|---|---|
| new BodyT | $x_s$ : seq of $\mathbb{R}$, $y_s$ : seq of $\mathbb{R}$, $m_s$ : seq of $\mathbb{R}$ | BodyT | ValueError |
| cm_x | | $\mathbb{R}$ | |
| cm_y | | $\mathbb{R}$ | |
| mass | | $\mathbb{R}$ | |
| m_inert | | $\mathbb{R}$ | |

### Semantics

#### State Variables

$cmx : \mathbb{R}$
$cmy : \mathbb{R}$
$m : \mathbb{R}$
$moment : \mathbb{R}$

#### State Invariant

$moment \geq 0 \land m > 0$

**Assumptions**

The arguments provided to the access programs will be of the correct type.

**Access Routine Semantics**

new BodyT($x_s, y_s, m_s$):

- transition:

$$\begin{aligned} \mathit{cmx}, \mathit{cmy}, m, \mathit{moment} := {} & \mathrm{cm}(x_s, m_s), \mathrm{cm}(y_s, m_s), \mathrm{sum}(m_s), \\ & \mathrm{mmom}(x_s, y_s, m_s) - \mathrm{sum}(m_s)(\mathrm{cm}(x_s, m_s)^2 + \mathrm{cm}(y_s, m_s)^2) \end{aligned}$$

- output: $\mathit{out} := \mathrm{self}$

- exception: $(\neg(|x_s| = |y_s| = |m_s|) \Rightarrow \mathrm{ValueError}\ |\neg(\wedge\mu : \mathbb{R}|\mu \in m_s : \mu > 0) \Rightarrow \mathrm{ValueError})$

cm_x():

- output: $\mathit{out} := \mathit{cmx}$

- exception: none

cm_y():

- output: $\mathit{out} := \mathit{cmy}$

- exception: none

mass():

- output: $\mathit{out} := m$

- exception: none

m_inert():

- output: $\mathit{out} := \mathit{moment}$

- exception: none

## Local Functions

sum : seq of $\mathbb{R} \to \mathbb{R}$
$\text{sum}(m_s) \equiv (+\mu : \mathbb{R} | \mu \in m_s : \mu)$

cm : seq of $\mathbb{R} \times$ seq of $\mathbb{R} \to \mathbb{R}$
$\text{cm}(z, m) \equiv (+i : \mathbb{N} | i \in [0..|m_s| - 1] : z_i m_i)/\text{sum}(m)$

mmom : seq of $\mathbb{R} \times$ seq of $\mathbb{R} \times$ seq of $\mathbb{R} \to \mathbb{R}$
$\text{mmom}(x, y, m) \equiv (+i : \mathbb{N} | i \in [0..|m| - 1] : m_i(x_i^2 + y_i^2))$

# Scene Module

## Template Module

Scene

## Uses

Shape, SciPy

## Syntax

### Exported Constants

None

### Exported Types

Scene = ?

### Exported Access Programs

| **name** | **In** | **Out** | **Exceptions** |
|---|---|---|---|
| new Scene | Shape, $\mathbb{R} \to \mathbb{R}$, $\mathbb{R} \to \mathbb{R}$, $\mathbb{R}$, $\mathbb{R}$ | Scene | |
| get_shape | | Shape | |
| get_unbal_forces | | $\mathbb{R} \to \mathbb{R}$, $\mathbb{R} \to \mathbb{R}$ | |
| get_init_velo | | $\mathbb{R}$, $\mathbb{R}$ | |
| set_shape | Shape | | |
| set_unbal_forces | $\mathbb{R} \to \mathbb{R}$, $\mathbb{R} \to \mathbb{R}$ | | |
| set_init_velo | $\mathbb{R}$, $\mathbb{R}$ | | |
| sim | $\mathbb{R}$, $\mathbb{N}$ | seq of $\mathbb{R}$, seq of (seq [4] of $\mathbb{R}$) | |

## Semantics

### State Variables

$s$ : Shape
$F_x : \mathbb{R} \to \mathbb{R}$ # *unbalanced force function in x dir*
$F_y : \mathbb{R} \to \mathbb{R}$ # *unbalanced force function in y dir*
$v_x : \mathbb{R}$ # *initial velocity in x dir*
$v_y : \mathbb{R}$ # *initial velocity in y dir*

## State Invariant

None

## Assumptions

None

## Access Routine Semantics

new Scene($s', F_x', F_y', v_x', v_y'$):

- transition: $s, F_x, F_y, v_x, v_y = s', F_x', F_y', v_x', v_y'$
- output: $out :=$ self
- exception: None

get_shape():

- output: $out := s$
- exception: none

get_unbal_forces():

- output: $out := F_x, F_y$
- exception: none

get_init_velo():

- output: $out := v_x, v_y$
- exception: none

set_shape(s'):

- transition: $s := s'$
- exception: none

set_unbal_forces($F_x', F_y'$):

- transition: $F_x, F_y := F_x', F_y'$
- exception: none

set_init_velo($v'_x, v'_y$):

- transition: $v_x, v_y := v'_x, v'_y$
- exception: none

sim($t_{\text{final}}$, $nsteps$):

- output: $out := t, \text{odeint}(\text{ode}, [s.\text{cm\_x}(), s.\text{cm\_y}(), v_x, v_y], t)$
  where $t = [i : \mathbb{N} | i \in [0..nsteps - 1] : \frac{i \cdot t_{\text{final}}}{nsteps - 1}]$
  *# for the sequence of t values the values should be in the natural ascending order for the i values*
- exception: none

## Local Functions

ode : (seq [4] of $\mathbb{R}$) $\times \mathbb{R} \rightarrow$ seq [4] of $\mathbb{R}$
$\text{ode}(w, t) \equiv [w[2], w[3], F_x(t)/s.\text{mass}(), F_y(t)/s.\text{mass}()]$

# Plot Module

## Module

Plot

## Uses

None

## Syntax

### Exported Constants

None

### Exported Access Programs

| **Routine name** | **In** | **Out** | **Exceptions** |
|---|---|---|---|
| plot | seq of (seq [4] of $\mathbb{R}$), seq of $\mathbb{R}$ | | ValueError |

## Semantics

### Environment Variables

win: two dimensional sequence of coloured pixels

### State Variables

None

### State Invariant

None

### Assumptions

None

**Access Routine Semantics**

plot$(w, t)$

- transition: modify win so that it displays three appropriately labelled $x$-$y$ graphs of the data points in $w$ showing the following plots

    1. $x$ versus $t$
    2. $y$ versus $t$
    3. $y$ versus $x$

  where $t$ is the supplied argument and

    - $x = \langle i : \mathbb{N} | i \in [0..|w| - 1] : w[i][0] \rangle$
    - $y = \langle i : \mathbb{N} | i \in [0..|w| - 1] : w[i][1] \rangle$

  (We are abusing the sequence notation with the assumption that the sequence will be built in the order of increasing $i$ values.) The plot should have the same structure as Figure 1.

- exception: $(\neg(|w| = |t|) \Rightarrow \text{ValueError})$

### 2.3.1 Considerations

Implementation of `Plot.py` is facilitated by the `matplotlib` package.

# SciPy Module

## 2.4 Module

SciPy

## 2.5 Uses

None

## 2.6 Syntax

### 2.6.1 Exported Access Programs

| Name | In | Out | Except. |
|---|---|---|---|
| odeint | (seq [4] of $\mathbb{R}$) $\times \mathbb{R} \rightarrow$ seq [4] of $\mathbb{R}$, seq [4] of $\mathbb{R}$, seq of $\mathbb{R}$ | seq of (seq [4] of $\mathbb{R}$) | ODE_ERR |

## 2.7 Semantics

### 2.7.1 State Variables

None

### 2.7.2 Access Routine Semantics

*#Solving* $\frac{d}{dt}\mathbf{y} = \mathbf{f}(\mathbf{y}(t), t)$
*# Bold font is used to indicate variables that are a sequence type*
odeint($\mathbf{f}$, $\mathbf{y}_0$, $\mathbf{t}$):

- output: $out := \langle i : \mathbb{N} | i \in [0..|t|-1] : \langle y_o(t_i), y_1(t_i), y_2(t_i), y_3(t_i) \rangle \rangle$ where

$$\mathbf{y}(t) = \mathbf{y}_0 + \int_{t_0}^{t_{\text{fin}}} \mathbf{f}(s, \mathbf{y}(s)) ds$$

- exception: $exc :=$ ( ODE Solver Fails $\Rightarrow$ ODE_ERR)

### 2.7.3 Considerations

This function is implemented by the Python library scipy. The syntax of odeint exactly matches what is given here. The output is a sequence of sequences. ODE_ERR stands for any error that can be raised by the scipy implementation of odeint.