Inverted Pendulum – Math and Simulation

Ivan Wely

May 16, 2019

Contents

Introduction

The inverted pendulum is a physical system which can be defined by its equations of motion. The equation of motion (singular) in this case, is a differential equation. More specifically, a second order non-linear ordinary differential equation. The solution to this equation is a function which takes time as its argument and returns an angle as its return value. Theoretically, one would just solve the equation and calculate the angle at specific time points, thus simulate the system. Practically, this is a bad choice for several reasons, mainly difficulty of solving the equation and simulation performance. This becomes even more prevalent for more complex systems like simulating a rocket for which the inverted pendulum is the precursor.

Solution

The alternative to solving the differential equations is to use a numerical approach instead. This has some disadvantages to the purely mathematical approach but which are insignificant when simulating which is why it is a good choice. The properties of using a purely mathematical approach are listed below.

- The calculations are exact without an error margin
- Avoids a state system – Calculations are independent of each other
  - Non-continuous calculations are significantly faster
- The solutions can be extremely difficult to find or impossible
- Continuous calculations are usually slower

While the approach for the numerical approach has the following properties.

- Continuous calculations are usually faster
- The solutions are extremely simple to find
- Requires a state system – Calculations depend on each other
  - Non-continuous calculations are extremely slow

Discrete Derivative

First Order

In order to use the numerical approach one must first understand what a discrete derivative is. A discrete derivative is simply a derivative which is finite and calculated as it is defined. Below is the discrete derivative for any first order derivative function.

$$y' = \frac{y_{i+1} - y_i}{h}$$

The purpose of the expression above is to be substituted into a first order differential equation such as the following.

$$y' = f(y)$$

Which can be numerically solved by simple substitution.

$$\frac{y_{i+1} - y_i}{h} = f(y)$$

$$y_{i+1} - y_i = f(y)h$$
$$y_{i+1} = f(y)h + y_i$$

The above is equivalent to the solution of the differential $y(x)$. The variable y_{i+1} and y_i are the next and current value from the function $y(x)$ respectively.

Second Order

As shown before the discrete derivative is equivalent to the solution to a differential. The same can be done for a second order derivative.

$$y'' = \frac{\frac{y_{i+1}-y_i}{h} - \frac{y_i-y_{i-1}}{h}}{h}$$

$$y'' = \frac{\frac{y_{i+1}-y_i-(y_i-y_{i-1})}{h}}{h}$$

$$y'' = \frac{y_{i+1} - y_i - y_i + y_{i-1}}{h^2}$$

$$y'' = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

A simple substitution can be done for any second order differential as below.

$$y'' = f(y)$$

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = f(y)$$

$$y_{i+1} - 2y_i + y_{i-1} = f(y)h^2$$

$$y_{i+1} = f(y)h^2 - y_{i-1} + 2y_i$$

In the case of a second order differential a third term y_{i-1} is added. This simply the previous value of $y(x)$.

Summary

Differential	Discrete Solution
$y' = f(y)$	$y_{i+1} = f(y)h + y_i$
$y'' = f(y)$	$y_{i+1} = f(y)h^2 - y_{i-1} + 2y_i$

The solution will look a bit different depending on how the differential looks but the steps are the same as presented before.

Simulation

In this section the equations of motion will be derived, code will be written, and a simulation will be executed. Below is a diagram of the inverted pendulum of interest.

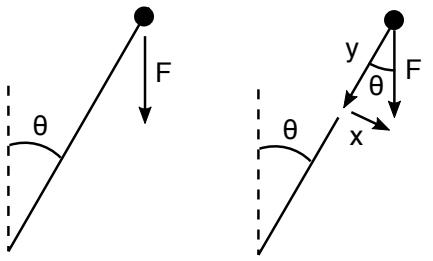

Figure 1: Inverted Pendulum

The arm of the inverted pendulum is defined to be massless and the mass at the end is defined to be a single point mass. There is no joint friction as well as no air resistance.

Equations of Motion (E.O.M)

The following calculations can be made given the diagram of the inverted pendulum.

$$F = mg \leftarrow [F_g = mg]$$

$$sin(\theta) = \frac{x}{F} = \frac{x}{mg} \leftarrow \left[sin(v) = \frac{opposite}{hypotenuse}\right]$$

$$x = sin(\theta)mg$$

$$\tau = Lx \leftarrow [\tau = r \times F]$$

$$I = mL^2 \leftarrow [I = mr^2]$$

$$Lx = mL^2\alpha \leftarrow [\tau = I\alpha]$$

$$\begin{aligned}\alpha &= \frac{Lx}{mL^2} \\ &= \frac{Lsin(\theta)mg}{mL^2} \\ &= \frac{sin(\theta)g}{L} \\ &= \frac{g}{L}sin(\theta)\end{aligned}$$

If we denote the angular acceleration α as the double derivative of the angle – $\ddot{\theta}$ the following equation of motion can be obtained for the inverted pendulum.

$$\ddot{\theta} = \frac{g}{L}sin(\theta)$$

The steps to solve the equation above are extremely complicated and so is the solution itself. However, there is one step to get the discrete solution, by simple substitution.

$$y_{i+1} = \left(\frac{g}{L}sin(\theta)\right)h^2 - y_{i-1} + 2y_i$$

Programming

Since this article exists on a webpage it would be nice to have a live example of the simulation. Therefore the webstack will be used to create the simulation. The programming language which will be used is JavaScript which is easy to read and easy to translate to any other language. Regarding a simulation which is needed for actual research purposes a language like JavaScript is not a valid choice. The only accepted programming languages are either C or C++. Choosing anything else only means you do not know C or C++. By the way, fuck off with Rust.

Proceeding, some basic functions required for the simulation to be run need to be defined. These are the functions that calculate the new y value and the $f(y)$ function.

```
const g = 9.82; //unit: m / s^2
const L = 1; //unit: m
const h = 0.001; //unit: seconds

function f(y)
{
    return (g / L) * Math.sin(y)
```

```
}

function y_next(y_last, y_now)
{
    return f(y_now) * h * h - y_last + 2 * y_now;
}
```

While the simulation runs virtual time passes. The time h passes for each calculation which means that for each calculation the program must be paused for h seconds before calculating again. This is so that the simulation runs in real time.

One problem arises though, since h is very small the actual pause will probably be smaller than the time it actually takes to call the pause function. Therefore h should be summed up until it reaches a certain limit h_{limit}, where it is reset back to $h = 0$ but the program now pauses for h_{limit} seconds.

```
function simulate()
{
    //y0 = y(i - 1)
    //y1 = y(i)
    //y2 = y(i + 1)
    var y0, y1, y2; //radians

    const h_limit = 0.1; //seconds
    var h_sum = 0; //seconds

    //initializers, define start angular velocity
    y0 = 0; //initial angle
    y1 = 0.0001; //angle after h seconds

    function loop()
    {
        y2 = y_next(y0, y1);
        y0 = y1;
        y1 = y2;

        h_sum += h;

        //quit the simulation if the angle is 90 degrees
        if(y2 >= Math.PI / 2) return;

        if(h_sum >= h_limit)
        {
            h_sum = 0;

            //multiply with 1000 because setTimeout takes milliseconds
```

```
            setTimeout(loop, h_limit * 1000);
        }
        else
        {
            loop();
        }
        return;
    }
    loop();
    return;
}
```

In order to make the graphical part of the simulation some HTML and CSS are added in separate files and the javascript is modified to work together with the graphical part. The code is available on https://github.com/fullnitrous/inverted-pendulum-example. Below is a embedded version of the simulation, you view the simulation at https://fullnitrous.com/inverted-pendulum-example/. *Embedded version is only visible if you are reading this in a web-browser.*

References

- My balls

Appendix

- https://en.wikipedia.org/wiki/Numerical_differentiation
- https://en.wikipedia.org/wiki/Differential_equation
- https://en.wikibooks.org/wiki/JavaScript/Control_structures