A crude introduction to MATHEMATICA

K. Şimşek

September 28, 2021

Contents

1 Introduction

Mathematica is a symbolic computation program. It was invented by Stephen Wolfram, originating during his PhD program as he was looking for a way to analytically compute the integrals he encountered. It is *precompiled*, in the sense that when you launch a new Mathematica *notebook*, it starts the *kernel* with lots of predefined functions. This makes Mathematica to bloat and slow down really fast[1]. However, the symbolic manipulation that Mathematica introduces can be said to be second to none[2].

2 Download and install

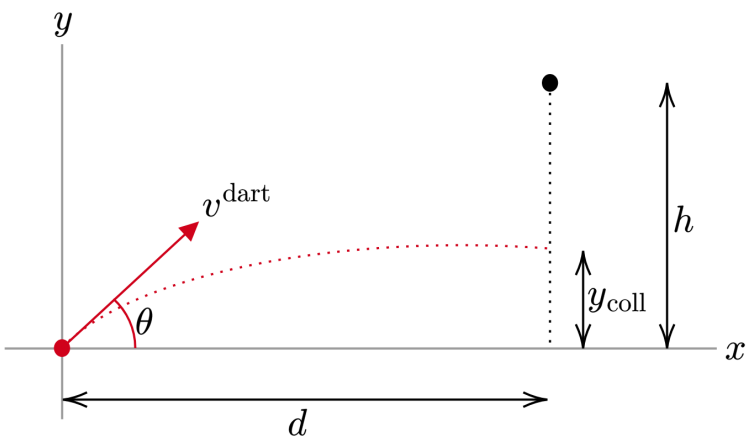
Wolfram is available freely to Northwestern faculty, staff, and students. Click here to visit the Wolfram User Portal, log in or create a new user account with your `@northwestern` email address, request a new activation key, download, install, and activate as instructed.

3 Example: Monkey on a tree

Let's solve the problem of *monkey on a tree* to go over the syntax and some basics of the Mathematica language.

[1] For a recent example from my work, what Python does in around 400 second, Mathematica does in a twice as much time, yet Fortran does it in around 20 seconds.

[2] There is a symbolic module for Python but with limited capabilities.



We have the following initial conditions:

$$x_0^{\text{dart}} = 0 \tag{1}$$
$$y_0^{\text{dart}} = 0 \tag{2}$$
$$v_{0,x}^{\text{dart}} = v^{\text{dart}} \cos(\theta) \tag{3}$$
$$v_{0,y}^{\text{dart}} = v^{\text{dart}} \sin(\theta) \tag{4}$$

and

$$x_0^{\text{monkey}} = d \tag{5}$$
$$y_0^{\text{monkey}} = h \tag{6}$$
$$v_{0,x}^{\text{monkey}} = 0 \tag{7}$$
$$v_{0,y}^{\text{monkey}} = 0 \tag{8}$$

There is no initial condition for the acceleration because acceleration defines the motion. The acceleration will be unique, created by the Earth, and common for all the bodies:

$$a_x = 0 \tag{9}$$
$$a_y = 0 \tag{10}$$

We can form the following vectors:

$$\mathbf{r}^{\text{dart}} = \begin{pmatrix} x^{\text{dart}} \\ y^{\text{dart}} \end{pmatrix}, \ \mathbf{v}^{\text{dart}} = \begin{pmatrix} v_x^{\text{dart}} \\ v_y^{\text{dart}} \end{pmatrix} \tag{11}$$

and

$$\mathbf{r}^{\text{monkey}} = \begin{pmatrix} x^{\text{monkey}} \\ y^{\text{monkey}} \end{pmatrix}, \ \mathbf{v}^{\text{monkey}} = \begin{pmatrix} v_x^{\text{monkey}} \\ v_y^{\text{monkey}} \end{pmatrix} \tag{12}$$

and

$$\mathbf{a} = \begin{pmatrix} a_x \\ a_y \end{pmatrix} \tag{13}$$

Let's define these objects in Mathematica. Launch a new Mathematica notebook. Start typing. You will see at the right-hand side of the window, there will be a square bracket closing. This indicates a *cell*. In Mathematica, instead of evaluating your entire code, you can divide it into cells and only compile what you need.

You can evaluate a single cell by clicking anywhere in that cell and pressing SHIFT ENTER. You can select multiple cells by pressing and holding CONTROL and then clicking on the above-mentioned *closing bracket at the right-hand side* of each cell, followed by SHIFT ENTER again. On a Mac, replace ENTER by RETURN and CONTROL by COMMAND.

Let's copy the following to a Mathematica notebook[3]:

```
x0Dart = 0;
y0Dart = 0;
v0xDart = vDart Cos[theta];
v0yDart = vDart Sin[theta];

x0Monkey = d;
y0Monkey = h;
v0xMonkey = 0;
v0yMonkey = 0;

ax = 0;
ay = -g;

r0vecDart = {x0Dart, y0Dart};
v0vecDart = {v0xDart, v0yDart};

rvecDart = {xDart, yDart};
vvecDart = {vxDart, vyDart};

r0vecMonkey = {x0Monkey, y0Monkey};
v0vecMonkey = {v0xMonkey, v0yMonkey};

rvecMonkey = {xMonkey, yMonkey};
vvecMonkey = {vxMonkey, vyMonkey};

avec = {ax, ay};
```

The breakdown of the code so far:

- Variable names can contain letters and numbers only — no underscores.
- Variables names cannot start with a number, which is common for many languages.

[3]Depending on the font-rendering, you may actually need to write it down on Mathematica instead of selecting the text here and pasting it.

- If you know C, you will recall that we need to put a semicolon at the end of each line. Here, we don't have to. We put a semicolon only when we want to suppress the output. Try it. See what happens when you remove the semicolon.

- The array notation in Mathematica is the curly braces. A vector like

$$\mathbf{V} = 2.1\hat{\mathbf{i}} + 3.5\hat{\mathbf{j}} + 1.4\hat{\mathbf{k}} = 2.1\hat{\mathbf{x}} + 3.5\hat{\mathbf{y}} + 1.4\hat{\mathbf{z}} = \begin{pmatrix} 2.1 \\ 3.5 \\ 1.4 \end{pmatrix} \tag{14}$$

  can be defined as[4]

```
V = {2.1, 3.5, 1.4};
```

  If you want to display the output as a vector, you can run

```
V // MatrixForm
```

  Mathematica is row-major, that is to say, if you want to enter a 2×2 matrix, say

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \tag{15}$$

  you would enter it like

```
{
 {A11, A12},
 {A21, A22}
}
```

- You don't need to declare variables.

- You can used undefined variables — we haven't defined `d, h, g`, or `xDart, yDart, vxDart, vyDart` and similarly for the monkey's variables, yet we are able to use and manipulate them as we wish. That's the best thing about a symbolical language.

- If you leave a space between to variables, it will imply a multiplication. You don't need to put a `*` in between. This is not true for the addition, subtraction, or division.

- Functions take their argument(s) inside a square braket, [ ], not a parentheses, ( ).

[4] If you are doing a purely symbolic job, you may want to avoid inserting numbers until the very end. However, if you really have to, then you may want to use integers. If there is a float-point number like `2.0` or simply `2.`, then Mathematica will try to convert everything into a float-point number. For instance, I wouldn't write the 1/2 factor in Eq. (20) as `1.0 / 2.0` or `1./2.` or `0.5`. I would really keep it as 1/2. It is a pure number, as far as our analysis is concerned. It comes from integration — it is not an input number like the gravitational acceleration would be, namely `g = 9.81`.

- Default functions, functions that have already been defined in the Mathematica kernel, start with a capital letter.

Now let's go back to the problem and think what happens. We have the global kinematics defined through the equation

$$x(t) = x_0 + v_{0,x}t + \frac{1}{2}a_x t^2 \tag{16}$$

$$y(t) = y_0 + v_{0,y}t + \frac{1}{2}a_y t^2 \tag{17}$$

for constant acceleration. Both the dart and the monkey will satisfy these two equations. Let's introduce them to Mathematica:

```
xDart[t_] := x0Dart + v0xDart t + 1/2 ax t^2
yDart[t_] := y0Dart + v0yDart t + 1/2 ay t^2

xMonkey[t_] := x0Monkey + v0xMonkey t + 1/2 ax t^2
yMonkey[t_] := y0Monkey + v0yMonkey t + 1/2 ay t^2
```

Here, we have defined functions of t. The underscore means *pattern-matching*. That is to say,

```
f[x_] := x^2
g[x] := x^2
```

are two very different things. For instance, `f[4]` returns 16 but `g[4]` returns just `g[4]` — Mathematica doesn't know what it is, so it just treats this object as a variable. However, `g[x]` does indeed return x^2.

Now comes the crucial part. At $t = t_{\text{coll}}$, we expect a collision. That is, both the monkey and the dart will occupy the same points in the $2D$ space:

$$x^{\text{monkey}}(t_{\text{coll}}) = x^{\text{dart}}(t_{\text{coll}}) \tag{18}$$

$$y^{\text{monkey}}(t_{\text{coll}}) = y^{\text{dart}}(t_{\text{coll}}) \tag{19}$$

In fact, we can further write

$$x^{\text{monkey}}(t_{\text{coll}}) = x^{\text{dart}}(t_{\text{coll}}) = x_{\text{coll}} \tag{20}$$

$$y^{\text{monkey}}(t_{\text{coll}}) = y^{\text{dart}}(t_{\text{coll}}) = y_{\text{coll}} \tag{21}$$

and we know $x_{\text{coll}} = d$ but we don't know d, so we will not evaluate it further:

```
xColl = d;
```

For our calculations to make sense, we have to obey the physical reality here:

- $t_{\text{coll}} > 0$
- $h > y_{\text{coll}} > 0$

- $g > 0$
- $d > 0$
- $v^{\text{dart}} > 0$
- $\frac{\pi}{2} > \theta > 0$

Without giving Mathematica this info, it may not simplify certain things for you — if you don't tell Mathematica that $t_{\text{coll}} > 0$, Mathematica will treat it as a complex number, which is the set of number that contains the real numbers, so Mathematica likes to keep things general. The syntax to tell Mathematica that the above-mentioned variables are positive is this:

```
$Assumptions = {
 tColl > 0, h > yColl > 0, g > 0, d > 0, vDart > 0, Pi/2 > theta > 0
};
```

Actually, θ can be anything between 0 and 2π but we know that having a θ greater than 90° does not make sense in this problem.

Next, let us introduce the equations we need to solve:

```
equations = {
 xDart[tColl] == xColl,
 xMonkey[tColl] == xColl,
 yDart[tColl] == yColl,
 yMonkey[tColl] == yColl
};
```

A single equality sign is a variable assignment, but two of them implies comparison, as you would imagine. We have just defined Eqs. (20) and (21) to Mathematica. Now the last step is to solve these equations in terms of θ, v^{dart}, and t_{coll}:

```
Solve[
 equation,
 {theta, vDart, tColl}
] // FullSimplify
```

That's it. Now save your file to a suitable location. Then, click on `Evaluation` at the top of the Mathematica window and then select `Evaluate Notebook`. At the end of the notebook, an output should be printed just like the following:

$$\left\{\left\{\text{theta} \to \text{ArcTan}\left[\frac{h}{d}\right],\ \text{vDart} \to \frac{\sqrt{\frac{g\left(d^2+h^2\right)}{h-\text{yColl}}}}{\sqrt{2}},\ \text{tColl} \to \sqrt{2}\ \sqrt{\frac{h-\text{yColl}}{g}}\right\}\right\}$$

***

There is another way to compile a Mathematica code but it comes with a catch, of course — which I know how to do it on a Unix-based machine. The catch is, you need to tell Mathematica what it should print; otherwise, it will be a legitimate blackbox. Here it is:

- Copy the following into a text editor:

```
x0Dart = 0
y0Dart = 0
v0xDart = vDart Cos[theta]
v0yDart = vDart Sin[theta]

x0Monkey = d
y0Monkey = h
v0xMonkey = 0
v0yMonkey = 0

ax = 0
ay = -g

xDart[t_] := x0Dart + v0xDart t + 1/2 ax t^2
yDart[t_] := y0Dart + v0yDart t + 1/2 ay t^2

xMonkey[t_] := x0Monkey + v0xMonkey t + 1/2 ax t^2
yMonkey[t_] := y0Monkey + v0yMonkey t + 1/2 ay t^2

xColl = d

$Assumptions = {
 tColl > 0, h > yColl > 0, g > 0, d > 0, vDart > 0, 0 < theta < Pi/2
}

equations = {
 xDart[tColl] == xColl,
 xMonkey[tColl] == xColl,
 yDart[tColl] == yColl,
 yMonkey[tColl] == yColl
}

Print[Solve[equations, {theta, vDart, tColl}] // FullSimplify]
```

  You will immediately notice that I dropped all the semicolons at the end of each line. But this time, I had to tell Mathematica that I want to see the output of `Solve[...]` on the screen when it is done compiling.

- Save the text as, say, `monkey.m`.

- Open a terminal and go to the directory that contains the file `monkey.m`.
- Run

  ```
  wolframscript -script monkey.m
  ```

  in the terminal.

Learning Mathematica this way has certain advantages. One of them is performance-wise. It is claimed that when you compile a Mathematica code, it runs faster if you do it via terminal, i.e. without using a notebook. This may be overkill, though. Mathematica is quite fast and the notebook interface is quite powerful.