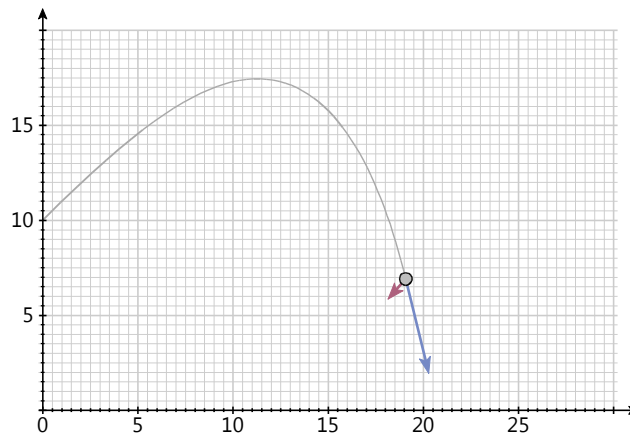


Exercise Sheet 6

Projectile Motion



In this exercise sheet the trajectory of a ball is to be calculated. During the flight, the ball experiences a flow resistance force through the air, which is directed against the movement. The trajectory depends on various parameters such as the initial position, the initial speed, the diameter of the ball and its mass. Depending on these parameters, it shall be determined how far the ball flies and how high it reaches during the flight.

The trajectory of the ball, its current speed and acceleration shall be displayed graphically. Similar to exercise sheet 5 you could program such a graphic component yourself. Instead, an existing graphic component called *MechanicsTVG* is to be used in this task sheet. This graphic component is used to visualize physical systems consisting of point-shaped masses. With *MechanicsTVG* it is possible to graphically display the trajectory, speed and inclination of one or more point-shaped masses. *MechanicsTVG* can be used in any physical system. *MechanicsTVG* contains various parameters that can be used to adapt *MechanicsTVG* to the different requirements of the physical systems. The graphics component *MechanicsTVG* will not only be used in this task sheet, but also in several other task sheets.

Preparation

Add the *mechanics.jar* library to your project. The file *mechanics.jar* can be found on the website in the area of this task. First, copy this file to your project directory. Then add this library to your classpath. In Eclipse click on the file *mechanics.jar* - right mouse button - build path - add to build path. Ready. From now on you can use the classes in this library in your project, especially the class *MechanicsTVG*.

Copy the following program code:

```

import static java.lang.Math.*;
import mechanics.tvg.MechanicsTVG;
import de.physolator.usr.*;

public class ProjectileMotion extends PhysicalSystem {

    public void initGraphicsComponents(GraphicsComponents g, Structure s, Recorder r,
        SimulationParameters sp) {
        MechanicsTVG t = new MechanicsTVG(this, s, r);
        t.geometry.setUserArea(0, 150, 0, 50);
        t.showPaths = true;
        t.showVelocity = true;
        t.showAcceleration = true;
        t.velocityScaling = 1;
        t.accelerationScaling = 1;
        t.showLabels = false;
        t.addPointMass("x", "y", "vx", "vy", "ax", "ay");
        g.addTVG(t);
    }
}

```

Explanation of the program code

This program code uses the graphics component *MechanicsTVG*. To do this, an instance of this class is first created in the *initGraphicsComponents* method. Several settings are made, with the help of which the graphic component is adapted to the requirements of the physical system *ProjectileMotion*. The visible range of the coordinate system is defined with *setUserArea* – as in exercise sheet 3. It is specified that the paths of the point-shaped masses are to be displayed (*showLabels*) and that their speed and acceleration are to be represented by arrows (*showVelocity*, *showAcceleration*). *velocityScaling* and *accelerationScaling* determine the correlation between vector length and the corresponding velocity or acceleration, respectively.

In this exercise sheet only one point-shaped mass is to be represented: the ball. The position of the ball is (x, y) , its speed (v_x, v_y) and its acceleration (a_x, a_y) . The *addPointMass* method specifies that the graphics component *MechanicsTVG* is to access the variables x, y, v_x, v_y, a_x and a_y in the physical systems in order to graphically display this point-shaped mass. The physical system *ProjectileMotion* does not yet contain these variables, but they shall be added in the first exercise.

With *MechanicsTVG* several point-shaped masses could be displayed at the same time. If there were several labels, it would make perfect sense to attach labels to the masses. In this case, there is only one point-shaped mass. This is why *showLabels* is set to *false*.

Exercise 1

A ball is thrown. Let $\begin{pmatrix} 0 \\ 10 \end{pmatrix} m$ be the starting position of the ball and let $\begin{pmatrix} 20 \\ 20 \end{pmatrix} \frac{m}{s}$ be its initial speed. The ball experiences an acceleration $g = 9,81 \frac{m}{s^2}$ due to gravity, which pulls it downwards.

Program this physical system in the same way as exercise sheet 1 by adding declarations for the physical variables, specifying the initial values of the physical variables and defining the derivation relationships between the physical variables. If required: Program a method *f* with the formulas to determine the dependent variable values.

Use the variable names x, y, v_x, v_y, a_x and a_y , where (x, y) stands for the position of the ball, (v_x, v_y) for its speed and (a_x, a_y) for the acceleration the ball experiences. After loading and starting the physical system you can see the graphic component *MechanicsTVG* on your screen. It shows the launching trace of the throw on the screen.

Exercise 2

The flow resistance has not yet been taken into account. Extend the existing physical system in such a way that the flow resistance is also taken into account when calculating the acceleration (exercise 1). It is assumed that the ball has a mass of $m=0,1\text{ kg}$ and a radius of $r=0,1\text{ m}$. The following equation applies to the flow resistance F_L . Be aware that the flow resistance F_L is always directed in the opposite direction to the direction of movement v .

$$F_L = \frac{1}{2} A c_w \rho v^2$$

In this equation $A=r^2\pi$ is the cross-sectional area of the ball, $c_w=0,4$ is the flow resistance coefficient for a spherical body and $\rho=1,2041\frac{\text{kg}}{\text{m}^3}$ is the density of air.

Due to the flow resistance, the ball flies much less far. Adjust the parameter value of `setUserArea` in a suitable way.

Exercise 3

Determine how high the ball flies. Additionally, determine the location where the ball reaches its highest altitude and determine the point in time when it reaches this location.

The point in time at which the ball reaches the highest point is characterized by the fact that at this point in time v_y has the value 0 . Initially, v_y is positive. As soon as the ball reaches the highest point, v_y falls below 0 . Insert the following code into your class:

```
public ThresholdTrigger tr1 = new ThresholdTrigger(() -> vy < 0);
```

This additional program code causes an event to be triggered whenever v_y exceeds or falls below the threshold value 0 . The central information in this piece of program is the condition $v_y < 0$. The boolean value of this conditions changes between *true* and *false* forth and back whenever the v_y exceeds or falls below the threshold value 0 .

The name of the variable *tr1* has no special meaning. If you work in a physical system with multiple triggers, you have to give each variable a different name. The triggers could then, for example, be named *tr1*, *tr2*, *tr3*, etc..

Replace the previous trigger with the following code:

```
public ThresholdTrigger tr1 = new ThresholdTrigger(() -> vy < 0)
    .setName("high point").setInfo(() -> "x="+x + " y="+y).setDoPrint(true);
```

In contrast to the previous program code, this program code assigns a name to the event, namely "high point". This additional information becomes relevant when working with different events. The method call `setDoPrint(true)` specifies that an output is printed on the console every time this event occurs. A one-line message then appears on the console stating that a threshold event has occurred, the name (if any) and the time. You can use the `setInfo` method to output other variable values in this row in addition to the time. In this case, the values of x and y are printed.

Exercise 4

Determine how far the ball flies. Determine when and where the ball reaches the ground ($y=0$). Use the techniques as in exercise 3.

Exercise 5

Determine the balls position at time $t=0.27134\text{s}$.

During simulation, the Physolator proceeds step by step and calculates the variable values for every simulation state. The step size can be varied. Depending on the selected setting, it is also possible for the Physolator to determine the step size for each step and to orientate itself towards certain accuracy targets when determining the step size. Therefore, it cannot be assumed that the Physolator determines the state of the physical system at time $t=0.27134s$. To force the state to be determined at time $t=0.27134s$, a physical event must be created at this time. The *TimeTrigger* can be used for this purpose.

Insert the following code into your *ProjectileMotion* class:

```
public TimeTrigger tr3 = new TimeTrigger(0.27134)
    .setName("time 1").setInfo(() -> "x=" + x + " y=" + y).setDoPrint(true);
```

The functionality of the *TimeTrigger* class is similar to that of the *ThresholdTrigger* class. Instead of a threshold value condition, a time is passed to the constructor. This class also has the methods *setName*, *setInfo* and *setDoPrint*.

The next step is to determine the position of the ball at several points in time: $0.27134s$, $0.4525s$ and $1.273s$. To achieve this, use the following program code instead of the previous time trigger *tr3*, without a special name for the events in this program code.

```
public TimeTrigger tr4 = new TimeTrigger(0.27134, 0.4525, 1.273)
    .setInfo(() -> "x=" + x + " y=" + y).setDoPrint(true);
```

The next step is to determine the position at several points in time, which are described by a sequence of points in time. The position is to be determined in equal intervals of $0.27134s$. To do this, use the following program code.

```
public TimeTrigger tr5 = new TimeTrigger((i)-> (i+1)*0.2713)
    .setInfo(() -> "x=" + x + " y=" + y).setDoPrint(true);
```

Remark: The first event sequence occurs at $0.27134s$. The count for this sequence of numbers starts at $i=0$. Therefore, the i^{th} event occurs at $(i+1)*0.2713s$.

Exercise 6

Let us assume, that the ball's initial speed be $v_0 = 28 \frac{m}{s}$ and that its initial position be $\begin{pmatrix} 0 \\ 10 \end{pmatrix} m$. Determine the angle at which the ball reaches its largest height and determine the angle at which the ball reaches the maximum flight distance, i.e. the maximum distance at which the ball hits the ground. The two angles for the largest height and largest flight distance shall be determined with an accuracy of 3 decimal places. In addition, the greatest possible height and maximum flight distance shall also be determined.

Recommendations: First add the constants $v0$ and phi to your program code. $v0$ stand for the initial velocity, phi stands for the starting angle measured in degrees relative to the horizontal. Calculate the initial values of v_x and v_y from $v0$ and phi . Perform several simulation runs with different values for phi and observe the height and flight distance achieved. Use interval nesting to find the phi values with the maximum height and the maximum flight distance, respectively.

Exercise 7

Now again, as in exercise 1, a throw is to be considered without the flow resistance of the air. The values for the starting position and velocity are the ones from exercise 1. In a throw without flow resistance, the trajectory of the ball describes a parabola. In this case, the throw height and throw distance can also be determined directly without simulation using paper and pencil. First determine these values with paper and pencil.

Then run a simulation. Use the previous program code with an initial speed of $\left(\frac{20}{20}\right)\frac{m}{s}$. If you set the air density ρ to 0, the flow resistance disappears. Compare your own calculated results for the throw height and throw distance with the simulation result.