# ME 2450 Lab 09b: System of Ordinary Differential Equations

## Objective

Two computer programs, `falling_ball` and `moving_train`, written in Lab 09a, will be modified so that the equations of motion are integrated by the $4^{\text{th}}$ order Runge-Kutta method. To receive credit for the lab, write the $4^{\text{th}}$ order Runge-Kutta function and incorporate it into the previously mentioned programs. Specifically:

- ☐ (4 points) write a function `rk4` implementing the fourth-order Runge-Kutta method to solve a system of ordinary differential equations with an arbitrary number of equations. The function is to be implemented in MATLAB or Python.
- ☐ (3 points) modify programs `falling_ball` and `moving_train` that you wrote in Lab 09a to use your `rk4` solver instead of `euler`.
- ☐ (2 points) Generate figures containing your results, as described in the ”Lab Assignment” section below.
- ☐ (1 point) Write a short explanation of your results. Specifically, address the following questions:
  1. At what step size does your solution for the ball problem converge using RK4?
  2. What is it about your plot that leads you to this conclusion?
  3. How would you choose the ”best” step size in this problem and others like it? What are the factors to consider?

## References

- A Brief Introduction to MATLAB, *MatlabIntro.pdf*.
- A Brief Introduction to Python, *PythonIntro.pdf*.
- An Introduction to Test Driven Development, *TestDrivenDevelopment.pdf*.
- Optional Function Arguments in MATLAB and Python, *OptionalArgs.pdf*.
- An Introduction to Function Handles, *FunctionHandles.pdf*.
- System of Ordinary Differential Equations, *Lab09a.pdf*.
- Lecture 14
- Chapter 25, Page 735, *Chapra, Numerical Methods for Engineers (7th Edition)*.

## Physical Model

See the description of the physical model in Lab09a.pdf.

## Numerical Methods

### Fourth-Order Runge-Kutta

In general, the solution to first-order ODE takes the form:

$$y_{i+1} = y_i + \phi h \tag{1}$$

where $y_i$ is value of the independent variable $y$ at $(t_i)$, $\phi$ is a slope, $h$ is the step size, and $y_{i+1}$ is the estimate of the value of $y$ at $t = t_i + h$. $t$ does not necessarily represent time, but any monotonically increasing independent variable such as space, or temperature, etc. The difference between the different Runge-Kutta methods (suce as Euler's method or Heun's method) is in how $\phi$ is calculated. In Euler's method, $\phi = f\left(t_i, y_i\right)$, or simply the value of the derivative function. For the fourth-order Runge-Kutta method, the slope is defined by:

$$\phi = \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \tag{2}$$

This is a weighted average of the estimates of the slopes $k_1$, $k_2$, $k_3$ and $k_4$, which are defined by the formulas:

$$\begin{aligned}
k_1 &= f\left(t_i, y_i\right) \\
k_2 &= f\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right) \\
k_3 &= f\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right) \\
k_4 &= f\left(t_i + h, y_i + k_3 h\right)
\end{aligned}$$

To implement the fourth-order Runge-Kutta method, start with a working Euler's method function and simply change the slope calculations to implement the above formulas.

## Lab Assignment

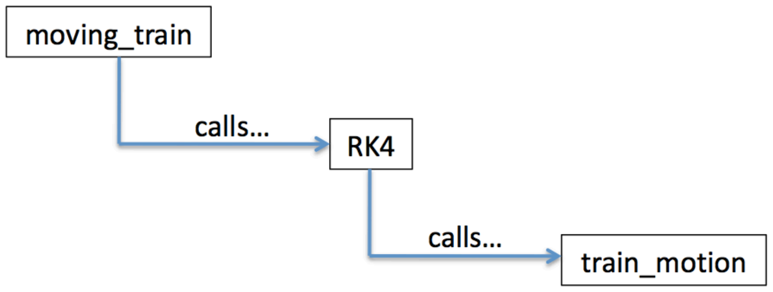

Figure 1: Structure of Lab 9b code. The driver script `moving_train` calls `RK4`, which in turn calls `train_motion` during each iteration.

### Fourth-Order Runge-Kutta

Write a function `[t,y] = rk4(odefun, tspan, y0)` that implements the fourth-order Runge-Kutta method (2) for a system of equations, where `tspan=[t0 t1 ... tf]`, that integrates a differential equation $y' = f(t,y)$ from `t0` to `tf` with initial conditions `y0`. Each row of the solution array `y` corresponds to a value in `t`.

**Input Arguments:**

- `odefun` (callable) - Function to solve. The function `dydt = odefun(t,y)`, for a scalar `t` and a vector `y`, must return a vector `dydt` having the same shape as `y` that corresponds to $f(t,y)$. `odefun` must accept both input arguments `t` and `y`, even if one of the arguments is not used in the function.
- `tspan` (array) - Time values of integration. The elements in `tspan` must be all increasing.

  The solver imposes the initial conditions given by `y0` at the initial time given in the first entry of `tspan`, then integrates from the first to the last entry in `tspan`.
- `y0` (vector) - Initial conditions. `y0` contains the initial conditions for the equation defined in `odefun`.

**Output Arguments:**

- `t` (array) - Evaluation points. `t` is the same as `tspan`.
- `y` (array) - Solution returned as an array. Each row in `y` corresponds to the solution at the value returned in the corresponding row of `t`.

### Falling Ball and Moving Train

1. Modify `falling_ball` to use `RK4`. Run the ball simulation for 10 seconds using a step size of 2 s. Plot position vs. time for this simulation. (This plot will look bad. This is intentional.) Next, run the simulation with smaller time steps, and watch what happens to the position vs. time plot. Turn in a plot with four curves on it - one using a step size of 2 s, and three with smaller step

sizes of your own choosing. Choose step sizes that illustrate your answers to the questions in the short write-up.

2. Modify `moving_train` to use `rk4`. Generate figures showing the velocity and position of the train for a duration of 10 seconds and a step size of 1 second.

Use the following system properties:

**Ball properties**

- Acceleration of gravity: $9.8$ m/s$^2$
- Air density: $1.0$ kg/m$^3$
- Ball mass: $0.01$ kg.
- Ball radius: $0.03$ m.
- Drag coefficient: $0.4$
- Initial conditions on ball: $z_0 = -2$ and $v_0 = 0$.

**Train properties**

- Acceleration of gravity: $9.8$ m/s$^2$
- Air density: $1.0$ kg/m$^3$
- Train mass: $10$ kg
- Total frontal area of train: $0.05$ m$^2$
- Drag coefficient: $0.4$
- Rolling resistance coefficient: $0.002$
- Propulsion force: $1.0$ N
- Initial conditions on train: $x_0 = 0$ and $v_0 = 0$.