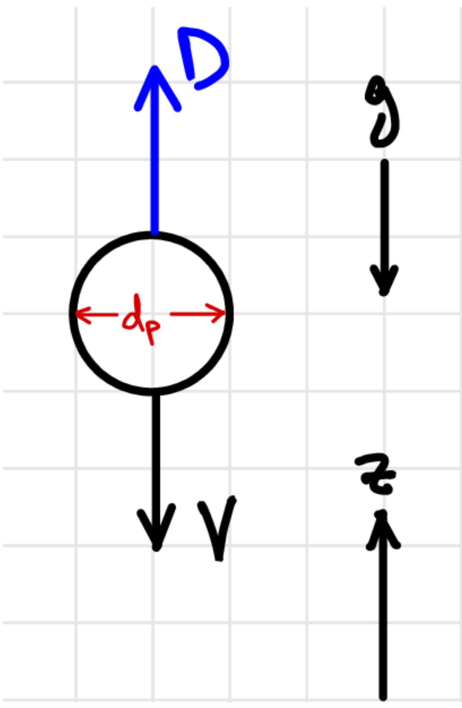
4.1.5 Problem Set: Hail particles implementation

Bookmark this page

In this part of the pset we will consider the acceleration of hail as it falls. As the particle moves, the aerodynamic drag force, D, opposes that motion such that the differential equation arising from application of Newton's 2nd Law is,

$$m\frac{\mathrm{d}V}{\mathrm{d}t} = mg - D \tag{4.1}$$

where m is the mass of the hail particle, V is the velocity of the hail particle, and g is gravitational acceleration.



We will assume that the hail is approximately spherical in shape and model its mass as,

$$m = \frac{\pi}{6}\rho_p\, {d_p}^3 \tag{4.2}$$

where d_p is the diameter of the hail and ρ_p is the density (mass per unit volume) of the hail.

The drag acting on the hail can be modeled as follows,

$$D = \frac{1}{2}\rho_a V^2 A_p C_D \tag{4.3}$$

where ρ_a is the density of the air in the atmosphere, A_p is the cross-sectional area of the hail, i.e.

$$A_p = \frac{\pi}{4}{d_p}^2 \tag{4.4}$$

and C_D is known as the drag coefficient. In general, the drag coefficient of the hail will depend on its speed V, however, for this hail model, we will assume it is constant. Also, the density of the air in the atmosphere generally depends on the altitude but we will again assume it is constant.

We will also solve for the altitude, $z(t)$, from the following differential equation,

$$\frac{\mathrm{d}z}{\mathrm{d}t} = -V \tag{4.5}$$

Thus, the system of differential equations for the hail IVP is,

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} V \\ z \end{bmatrix} = \begin{bmatrix} g - D/m \\ -V \end{bmatrix} \tag{4.6}$$

For this pset, use the following initial conditions and parameter values:

$$t_I = 0, \qquad t_F = 20\,\mathrm{s}, \qquad \mathrm{V}\left(\mathrm{t_I}\right) = 0 \tag{4.7}$$

$$\rho_a = 1\,\mathrm{kg/m^3}, \qquad g = 9.81\,\mathrm{m/s^2}, \qquad z\left(t_I\right) = 5000\,\mathrm{m} \tag{4.8}$$

$$\rho_p = 700\,\mathrm{kg/m^3}, \qquad d_p = 0.02\,\mathrm{m}, \qquad C_D = 0.5 \tag{4.9}$$

Note that in the provided template code `hail.py`, the values of the initial conditions and parameters are set in the main body of the code and then used to instantiate the `hail_IVP` object by calling the `HailIVP` constructor.

For testing of your implementation, you can compare your velocity from the numerical methods with the analytic solution which is possible to derive with our simplifying assumptions that ρ_a and C_D are constants. Specifically, the analytic solution to Equation (4.1) is,

$$V\left(t\right) = V_{\mathrm{term}} \tanh\left(\frac{gt}{V_{\mathrm{term}}} + C\right) \tag{4.10}$$

where V_{term} is the terminal velocity given by,

$$V_{\mathrm{term}} = \sqrt{\frac{2mg}{\rho_a A_p C_D}} \tag{4.11}$$

and C will depend on the initial velocity $V_0 = V\left(0\right)$. Specifically, for $V_0 = 0$ the value of $C = 0$. This analytic solution is not to be used in the numerical methods, but rather only for comparing with the numerical solution and for calculating the error in the numerical solution.

1. **Complete the implementation of `hail.py`.** Specifically:

   - **Complete `HailIVP.evalf(self, u, t)`.**

     Note: The parameters $\{g, \rho_a, \rho_p, ...\}$ are already loaded into `self`'s parameter dictionary when `self` was instantiated. To access these values, call the getter `self.get_p(key)`, as `get_p` is a method defined in the `IVP` class. The constant π is implemented in Python as `math.pi`.

   - Complete `hail_Verror(hail_IVP, t, V)`. Assume the initial condition $V_0 = 0$. Remember to call the `get_p` method on `hail_IVP` to obtain the relevant parameter values.

   - Implement `hail_Veplot(...)` so that it produces the plot shown in Figure 4.2, as well as a similar one using the `step_RK4` method.

- Implement `hail_Vzplot(...)` so that it produces the plot shown in Figure 4.3, as well as a similar one using the `step_RK4` method.

- **Note:** For the plotting functions, remember to return the `Axes` object (or array of `Axes`) provided by matplotlib's `subplots` function. The autograder needs this in order to inspect your plots. See `subplots`'s documentation here.

- **Note:** Your plots must label the axes and legends exactly as in the provided plots. The data being plotted should also match ours within machine precision. However, you do not need to get the colors to match, nor do you need to set the ranges of the x- and y-axes.

- **Note:** When plotting the dots in Figures 4.2 and 4.3, please use pyplot's `plot` function with the proper markers, rather than `scatter`. In this pset, the autograder will not read data plotted using `scatter`.

Then, run `hail.py` which will execute your functions with both the Forward Euler and RK4 methods, specifically:

- For $\Delta t = 1\,\mathrm{s}$, plot the numerical and exact $V(t)$, the error $e(t)$, and the altitude $z(t)$. If your implementations are correct, your plots should look like Figures 4.2 and 4.3.

- For $\Delta t = [0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 1.28]\ \mathrm{s}$, plot how the maximum error depends on Δt and the number of forcing evaluations (i.e. calls to `HailIVP.evalf`). If your implementations are correct, your plots should look like Figures 4.4 and 4.5.

FAIR GAME WARNING FOR THE EXAM: Given plots like Figures 4.4 and 4.5, you should be able to determine the rate of convergence of each method with Δt from the plot of maximum error versus Δt. Further, you should also be able to answer and explain the following: what is the ratio of function evaluations required for FE to achieve an error of 0.1 m/s compared to an error of 0.001 m/s? For RK4?

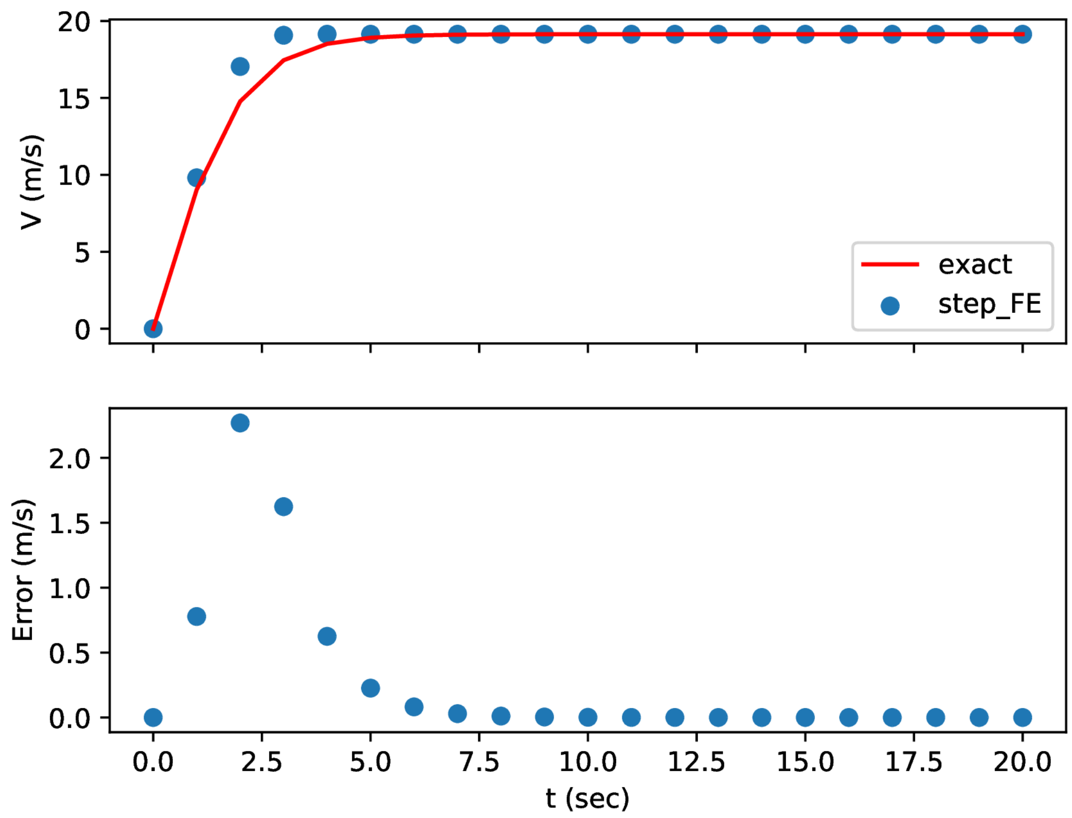


Figure 4.2: Desired output from `hail.hail_Veplot`

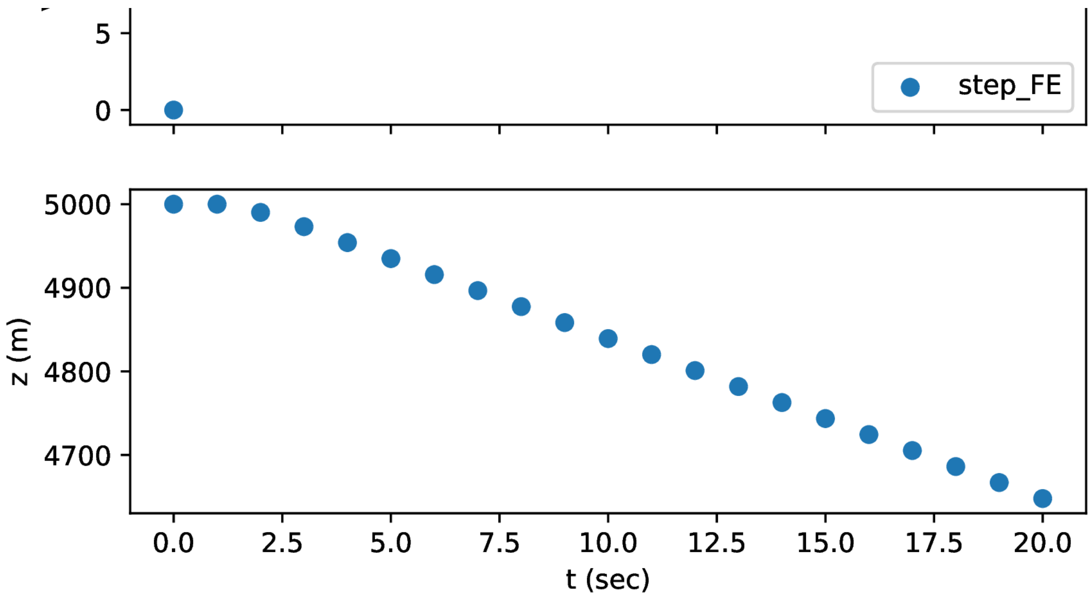


Figure 4.3: Desired output from `hail.hail_Vzplot`

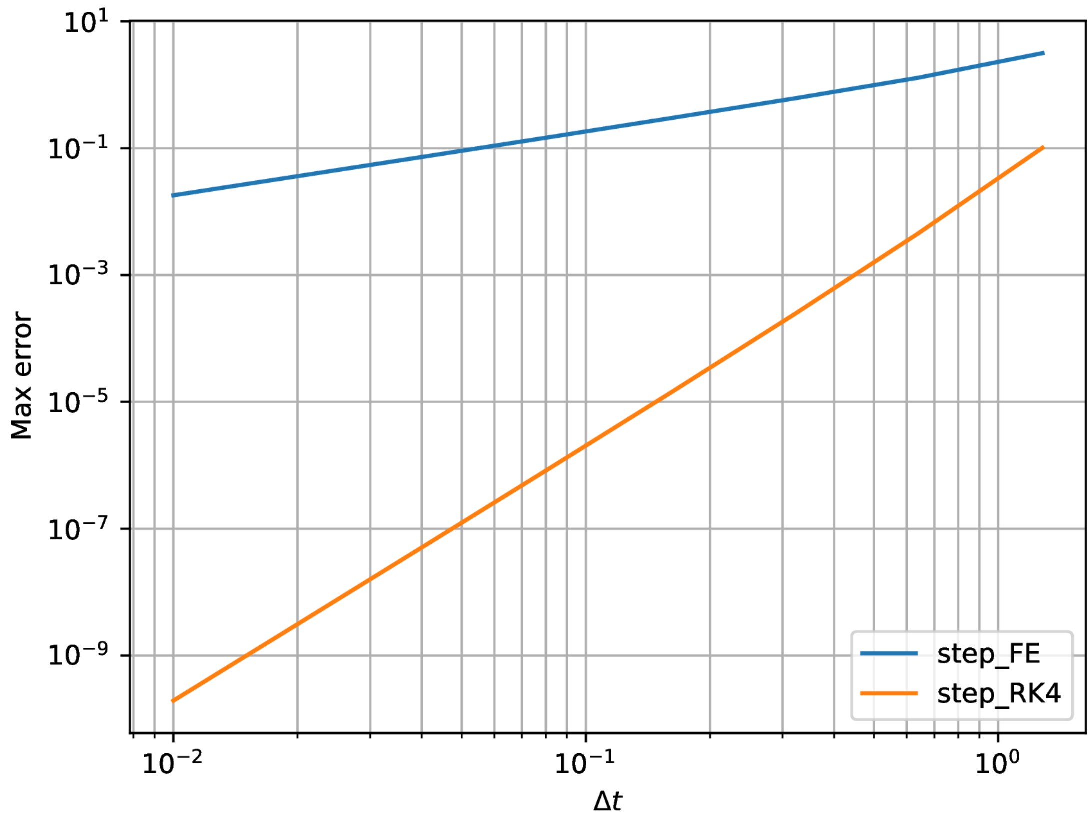
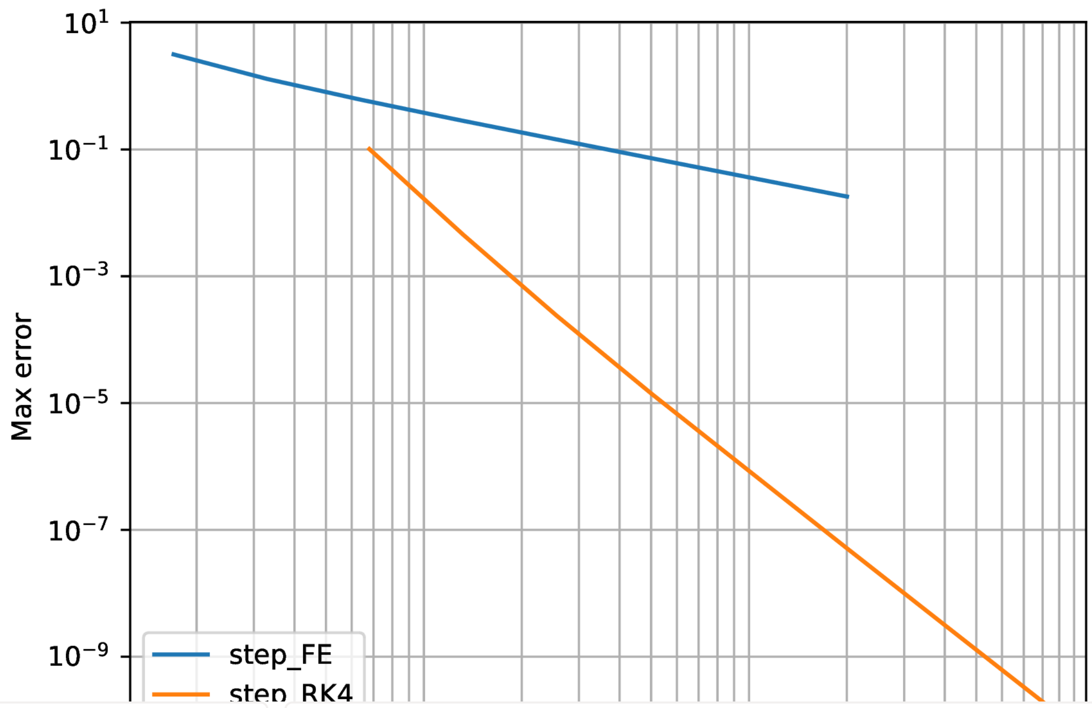


Figure 4.4: Correct results for max e versus Δt



< Previous
Next >