exercise_4

May 23, 2023

[ ]:
```
from phi.torch.flow import*
import pylab as plt
```

[ ]:
```
g = 9.81
```

$\Delta x = v_0 t + \frac{1}{2} g t^2$

$x = x_0 + \Delta x$

[ ]:
```
def analytical_solution(p0, v0, t):
    return math.vec(x = p0['x'] + v0['x'] * t,
                    y = p0['y'] + v0['y'] * t - 0.5 * g * t ** 2)
```

$\frac{\partial p}{\partial t} = v + at$

$p_{i+1} = p_i + \Delta t \frac{\partial p}{\partial t}$

[ ]:
```
def euler_soution(p, v, dt):
    return math.vec(x = p['x'] + v['x'] * dt, y = p['y'] + v['y'] * dt - g * dt␣
↪** 2 ), math.vec(x = v['x'], y = v['y'] - 9.81 * dt)
```

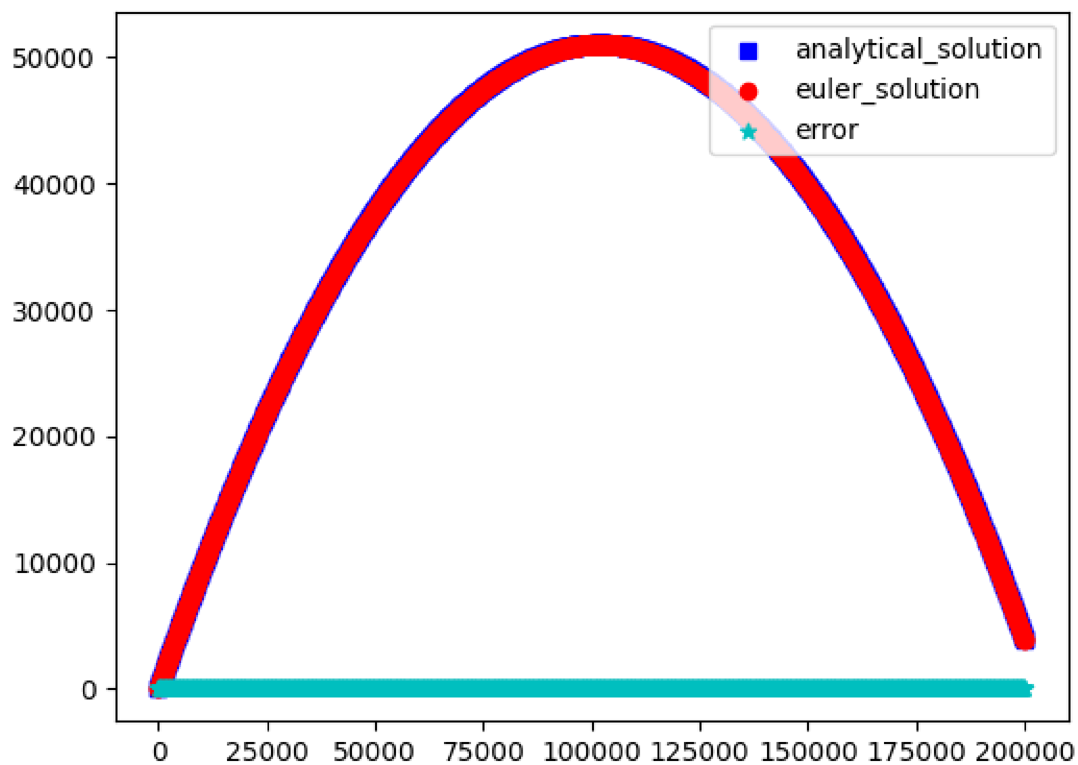
[ ]:
```
p0 = math.vec(x = 0, y = 0)
v0 = math.vec(x = 1e3, y = 1e3)
```

[ ]:
```
T = 2e2 # total time to observe ball
def animate_euler(p0, v0, dt):
    X,V = [p0], [v0]
    for t in range(int(T / dt) - 1):
        x, v = euler_soution(X[-1], V[-1], dt)
        X.append(x)
        V.append(v)
    return X, V

def animate_analytical_solution(p0, v0, dt):
    analytical_trj = []
    return [analytical_solution(p0, v0, dt*t) for t in range(int(T / dt))]

def plot_solutions(euler_solution, analytical_solution):
```

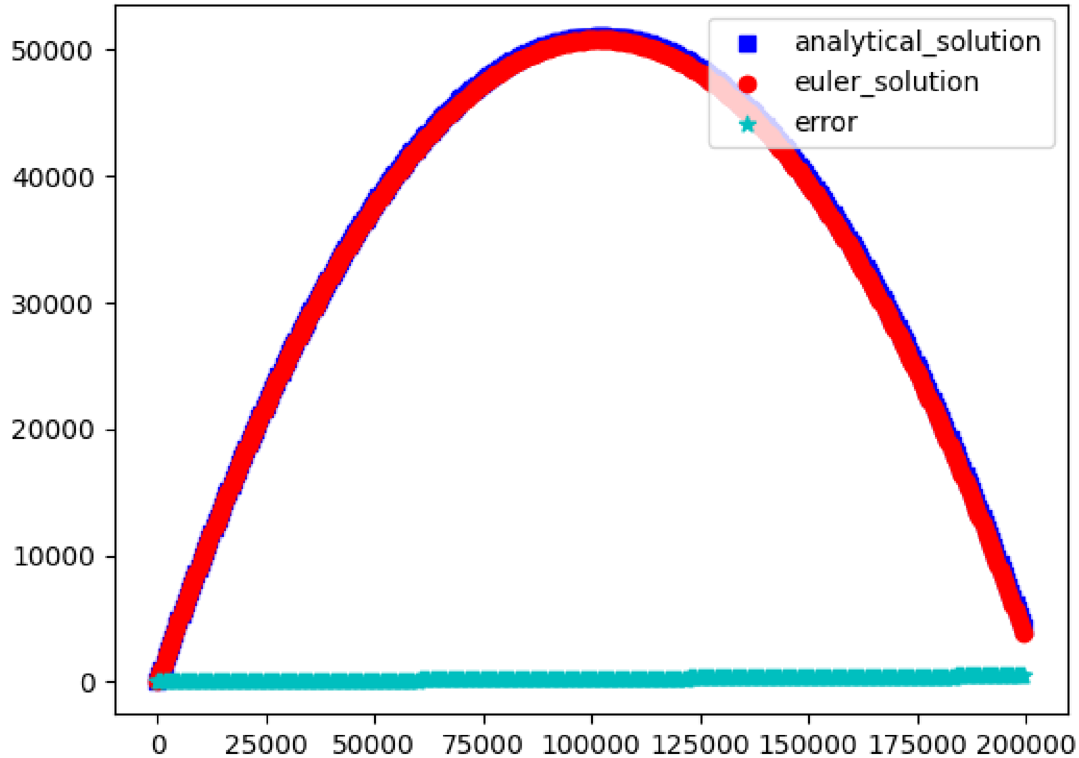
```
    x_analtical, y_analitical = [x for x in analytical_solution.points[:].
↪vector['x']], [x for x in analytical_solution.points[:].vector['y']]
    x_euler, y_euler = [x for x in euler_solution.points[:].vector['x']], [x␣
↪for x in euler_solution.points[:].vector['y']]
    error = [i for i in analytical_solution.points[:]['y'] - euler_solution.
↪points[:]['y']]

    fig = plt.figure()

    ax1 = fig.add_subplot(111)
    ax1.scatter(x_analtical, y_analitical,  c='b', marker="s",␣
↪label='analytical_solution')
    ax1.scatter(x_euler, y_euler, c='r', marker="o", label='euler_solution')
    ax1.scatter(x_analtical, error, c='c', marker="*", label='error')
    plt.legend(loc='upper right')
    plt.show()
```

Try with different time steppings

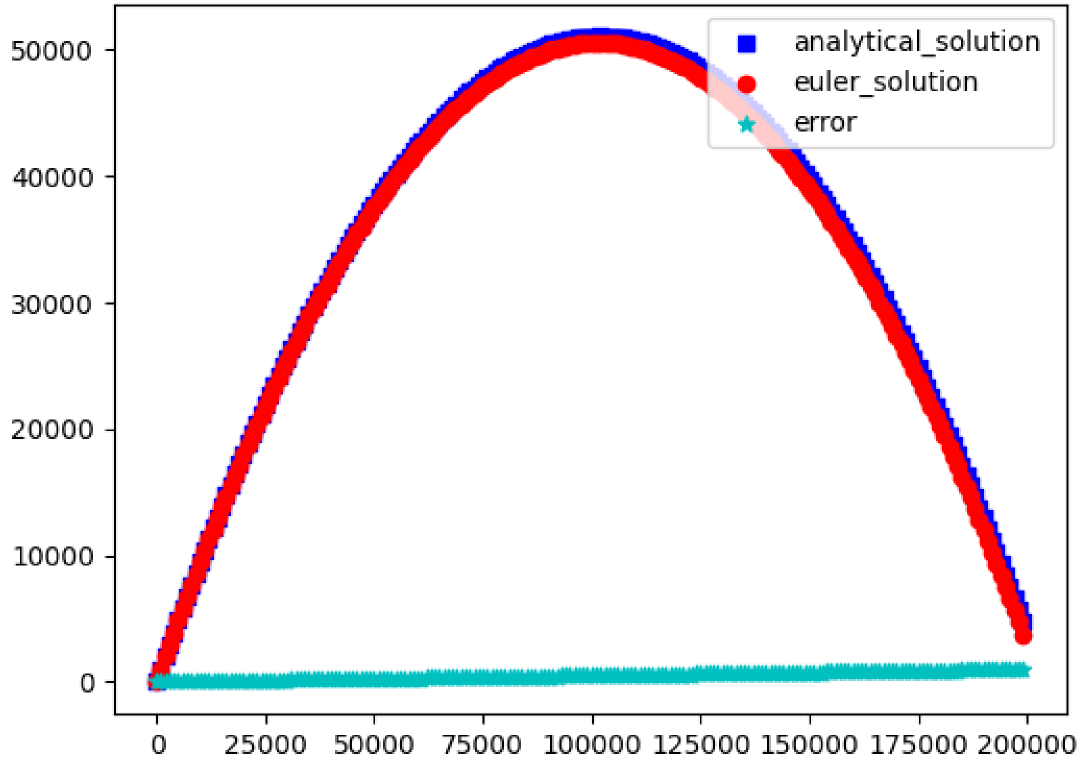
```
[ ]: dt = 0.1
analytical_trj_1 = stack(animate_analytical_solution(p0, v0, dt),␣
 ↪instance('points'))
euler_1 = stack(animate_euler(p0, v0, dt)[0], instance('points'))
plot_solutions(euler_1, analytical_trj_1)
```



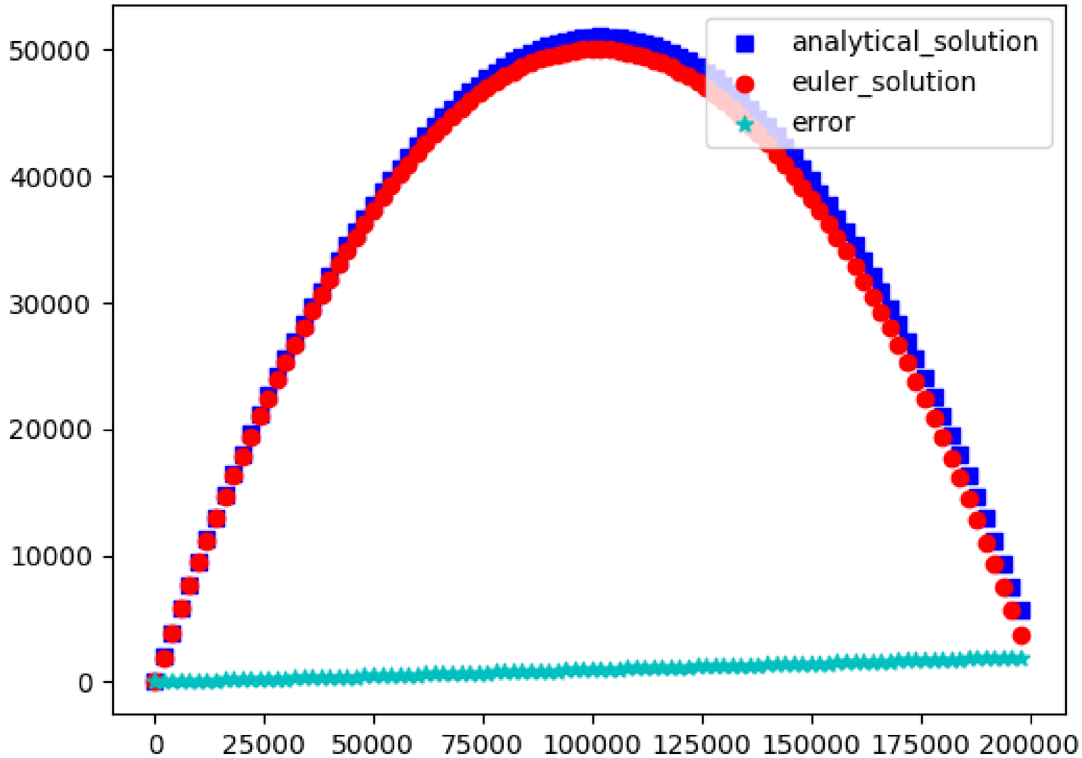
[ ]:
```
dt = .5
analytical_trj_2 = stack(animate_analytical_solution(p0, v0, dt),␣
    ↪instance('points'))
euler_2 = stack(animate_euler(p0, v0, dt)[0], instance('points'))
plot_solutions(euler_2, analytical_trj_2)
```



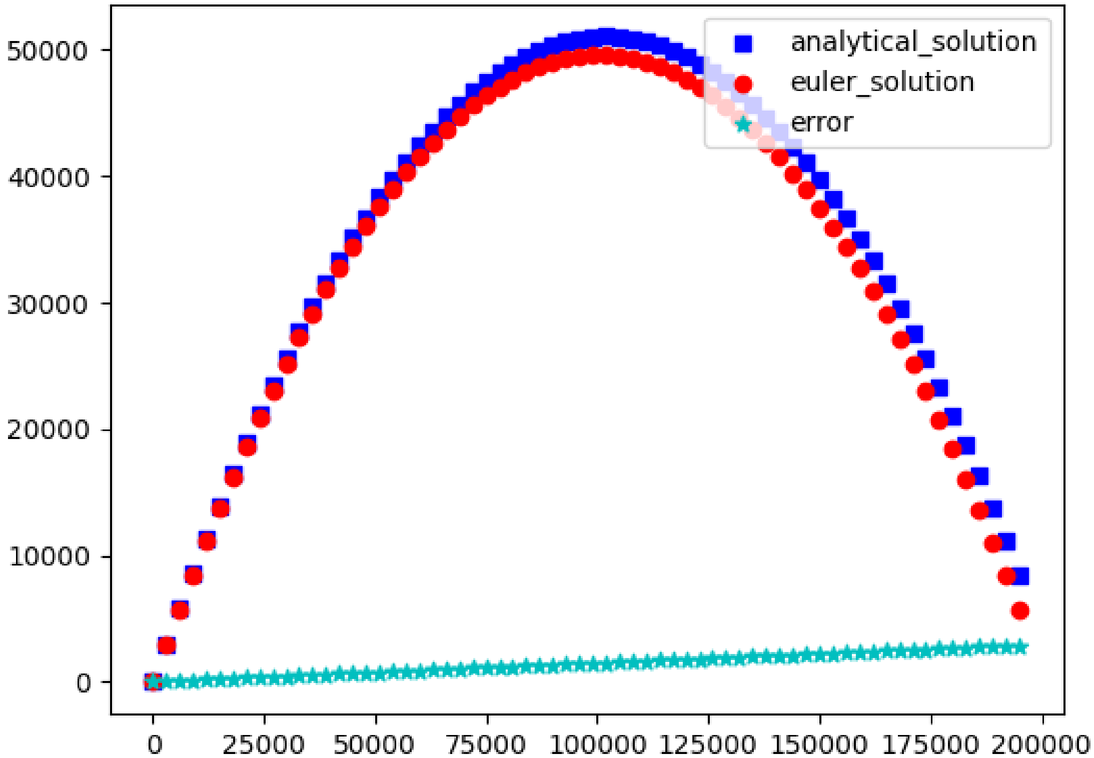
[ ]:
```
dt = 1
analytical_trj_3 = stack(animate_analytical_solution(p0, v0, dt),␣
    ↪instance('points'))
euler_3 = stack(animate_euler(p0, v0, dt)[0], instance('points'))
plot_solutions(euler_3, analytical_trj_3)
```



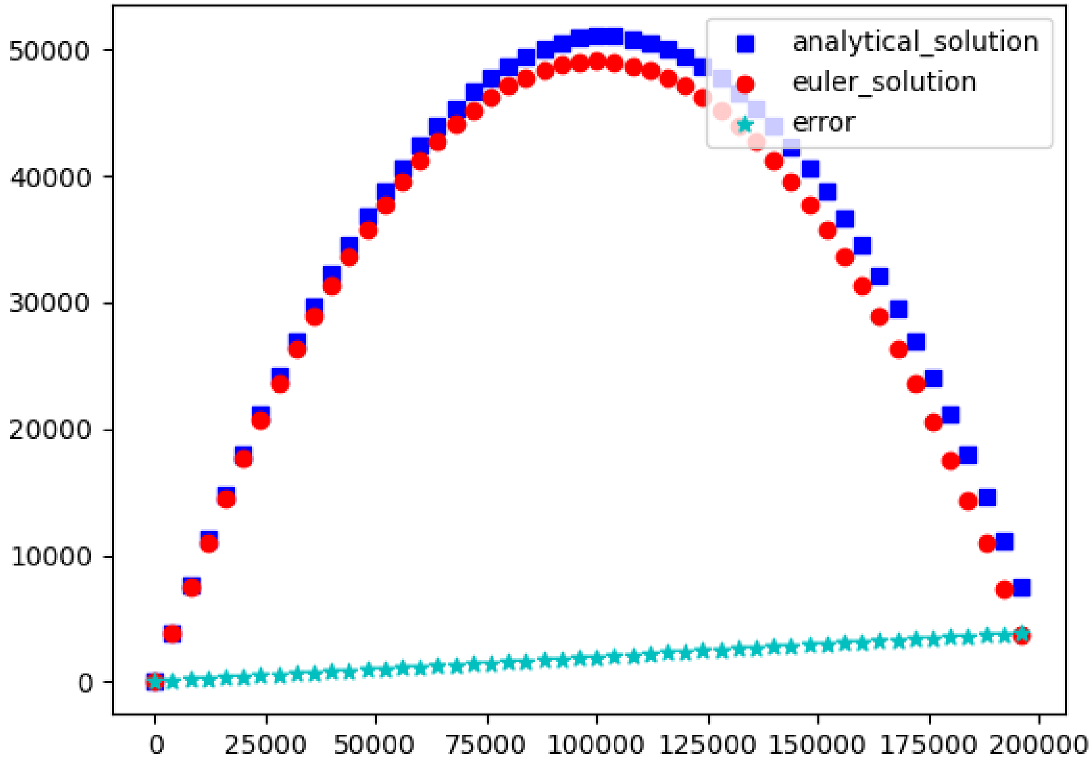
[ ]:
```
dt = 2
analytical_trj_4 = stack(animate_analytical_solution(p0, v0, dt),␣
 ↪instance('points'))
euler_4 = stack(animate_euler(p0, v0, dt)[0], instance('points'))
plot_solutions(euler_4, analytical_trj_4)
```



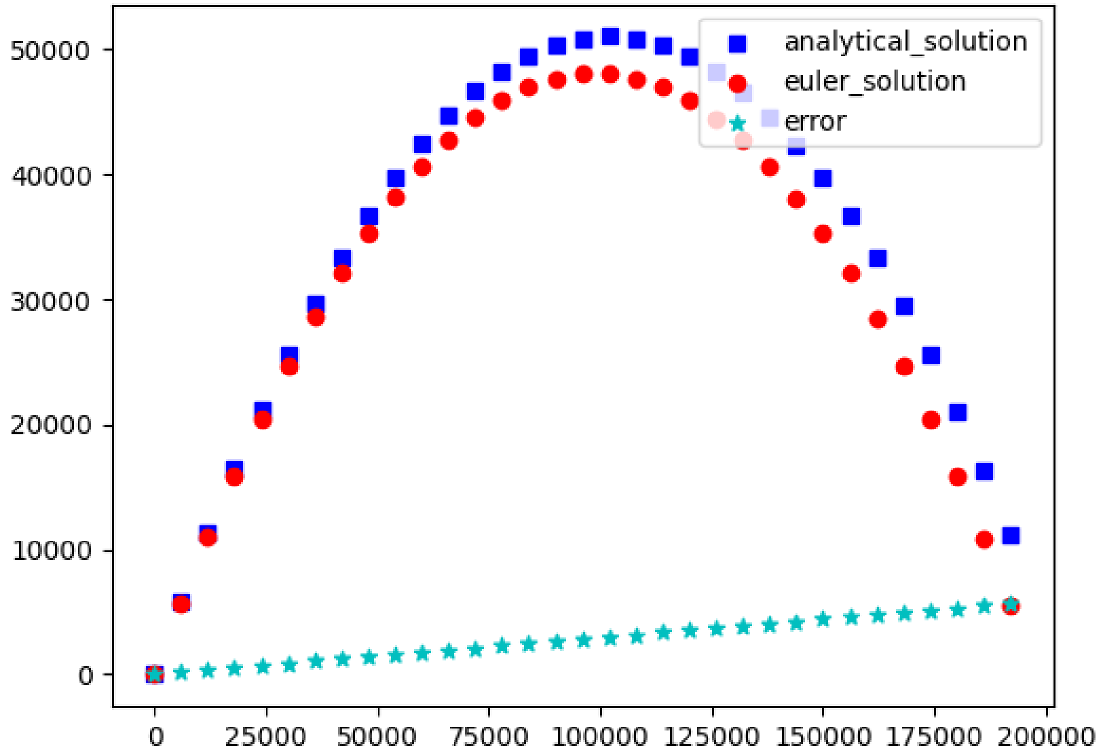
[ ]:
```
dt = 3
analytical_trj_5 = stack(animate_analytical_solution(p0, v0, dt),␣
  ↪instance('points'))
euler_5 = stack(animate_euler(p0, v0, dt)[0], instance('points'))
plot_solutions(euler_5, analytical_trj_5)
```



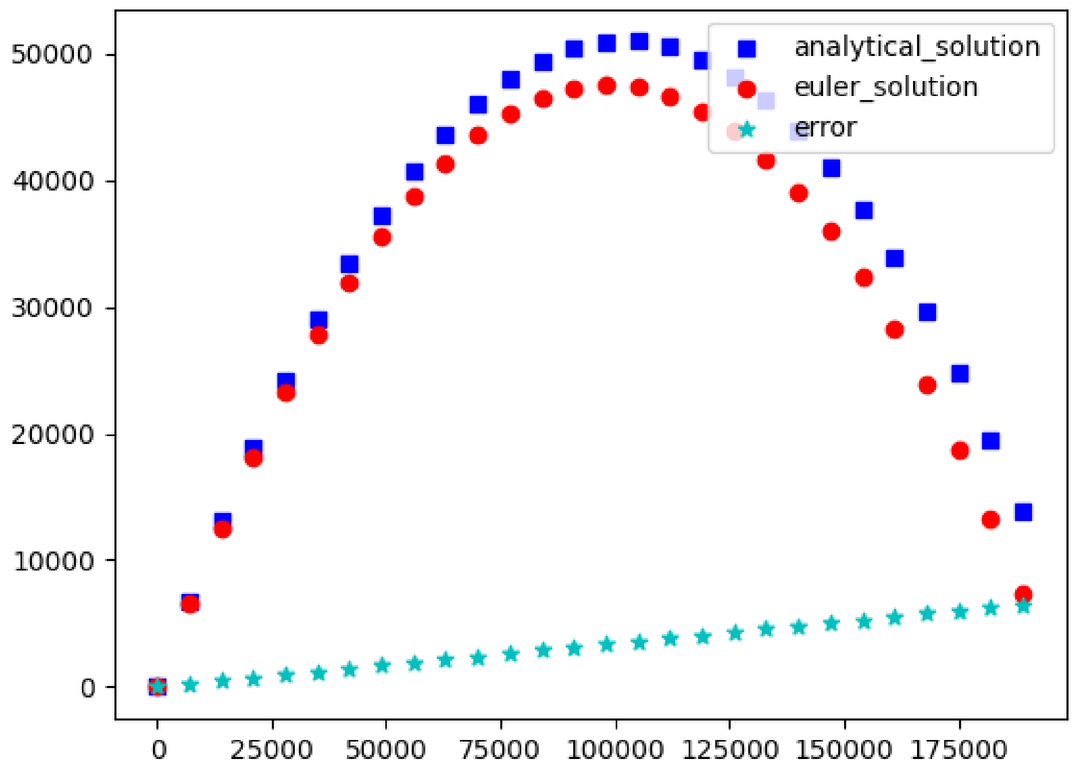

```
[ ]: dt = 4
     analytical_trj_6 = stack(animate_analytical_solution(p0, v0, dt),␣
      ↪instance('points'))
     euler_6 = stack(animate_euler(p0, v0, dt)[0], instance('points'))
     plot_solutions(euler_6, analytical_trj_6)
```



[ ]:
```
dt = 6
analytical_trj_7 = stack(animate_analytical_solution(p0, v0, dt),␣
↪instance('points'))
euler_7 = stack(animate_euler(p0, v0, dt)[0], instance('points'))
plot_solutions(euler_7, analytical_trj_7)
```




```
[ ]: dt = 7
analytical_trj_8 = stack(animate_analytical_solution(p0, v0, dt),␣
 ↪instance('points'))
euler_8 = stack(animate_euler(p0, v0, dt)[0], instance('points'))
plot_solutions(euler_8, analytical_trj_8)
```



[ ]:
```
math.seed(1001)
```

- As Δt increases, error increases as well. This is expected since in y direction due to gravity, Euler is not able to provide exact solution. Due to the accumulation of the error through the simulation later points are further away from the exact solution.

- Choose Δt such that there occurs a clear error beteen the exact and approximated solution

[ ]:
```
dt = math.random_uniform(low = 6, high=7)
dt
```

[ ]: 6.395438

[ ]:
```
x_euler, v_euler = animate_euler(p0, v0, dt)
x_ref = animate_analytical_solution(p0, v0, dt)
```

- input features: solution of the Euler approximation
- output features: correction values for Δx

[ ]:
```
input_features = int(T / dt)  # euler solution for del_x
output_fatures = int(T / dt)  # correction for del_x at each point
network = dense_net(in_channels=input_features, out_channels=output_fatures,␣
 ↪layers=[8, 8], activation='ReLU')
```

```
optimizer = adam(net=network, learning_rate=1e-3)
```

[ ]:
```
x_analtical, y_analitical = [float(x['x']) for x in x_ref], [x['y'] for x in␣
↪x_ref]
y_euler = [x['y'] for x in x_euler] # input network input
dx  = [y1 - y2 for y1, y2 in zip(y_euler, y_analitical)] # target: difference␣
↪between the exact solution and similation
```

- approximated solution -> network -> compare network output with analytical soluiton Δx for each time step
- Since there is no acceleration in the x direction, we do not need to work on values in x direction. Euler makes linear approximations which is the exact motion in x direction. Thus, take y values and train network to guess Δy values to recorrect.

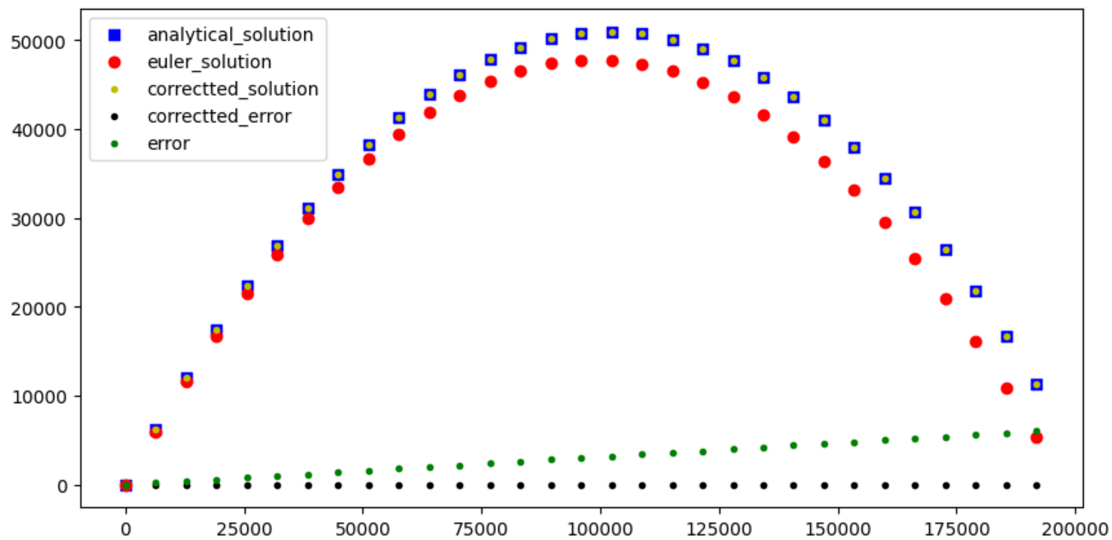
[ ]:
```
def loss_function(data, target):
    data, target = tensor(data), tensor(target)
    prediction = math.native_call(network, data)
    return math.l2_loss(prediction - target), prediction
```

[ ]:
```
print(f"Initial loss: {loss_function(data=y_euler, target = dx)}")
for i in range(1000):
    loss, prediction = update_weights(network, optimizer, loss_function,␣
↪data=y_euler, target = dx)
    if i % 100 == 0:
        print(f"iteration: {i} loss: {loss} ")
print(f"Final loss: {loss}")
```

```
Initial loss: (423714340.0, (vector =31) -1.21e+03 ±

4.4e+03 (-8e+03…6e+03))
iteration: 0 loss: 423714340.0
iteration: 100 loss: 86382340.0
iteration: 200 loss: 2265869.0
iteration: 300 loss: 4031.084
iteration: 400 loss: 3.1522207
iteration: 500 loss: 0.0010117802
iteration: 600 loss: 1.1046544e-05
iteration: 700 loss: 9.343847e-06
iteration: 800 loss: 6.1842857e-06
iteration: 900 loss: 7.93877e-06
Final loss: 4.684969e-06
```

[ ]:
```
# prepare data to plot
network_correction  = [float(x) for x in prediction.value.vector]
y_corrected = [float(y2 - y1) for y1, y2 in zip(network_correction, y_euler)]

corrected_error = [float(y1 - y2) for y1,y2 in zip(y_corrected, y_analitical)]
error = [float(y1 - y2) for y1,y2 in zip(y_analitical, y_euler)]
```

```
fig = plt.figure(figsize=(10,5))
ax1 = fig.add_subplot(111)
ax1.scatter(x_analtical, [float(x) for x in y_analitical],  c='b', marker="s",␣
 ↪label='analytical_solution')
ax1.scatter(x_analtical, [float(x) for x in y_euler], c='r', marker="o",␣
 ↪label='euler_solution')
ax1.scatter(x_analtical, y_corrected, c='y', marker=".",␣
 ↪label='correctted_solution')
ax1.scatter(x_analtical, corrected_error, c='black', marker=".",␣
 ↪label='correctted_error')
ax1.scatter(x_analtical, error, c='g', marker=".", label='error')
plt.legend(loc='upper left')
plt.show()
```



- After the correction of the Euler from the output of the network, soution matches with the analytical solution