 [1]

Mini Project 2

BUNGEE JUMPING

Peili Guo | peili.guo.7645@student.uu.se | 09 Oct 2017

[1] World's highest commercial bungee jump at Macau Tower 233m.

(Picture curtesy: https://www.macautower.com.mo/tower-adventure/tower-adventure/bungy-jump/)

Introduction

In this report, bungee jumping is modeled as an ordinary differential equation (ODE) and solved using numerical method in matlab. Some result figures are in appendix and all matlab files are attached in Appendix.

Solution

1. NUMERICAL SOLUTIONS AS SYSTEM OF ODES

y1 is the distance travelled and y2 is the velocity. The rope will only have affect when it is stretch that is when distance travelled is larger than the length of the rope 150m.

when y1 < 150m, we have:

$$\begin{cases} y1' = y2 \\ y2' = \dfrac{mg - sign\big(y2(t)\big) * Cv * y2(t)^2}{m} \end{cases}$$

when y1 ≥ 150m, we have:

$$\begin{cases} y1' = y2 \\ y2' = \dfrac{mg - sign\big(y2(t)\big) * Cv * y2(t)^2 - 50(y1(t) - 150)}{m} \end{cases}$$

Put these equations into matlab and using the ode45 solver, set the start time at 0 and finish time at 100, initial value y1=0 and y2=0. We get the result:

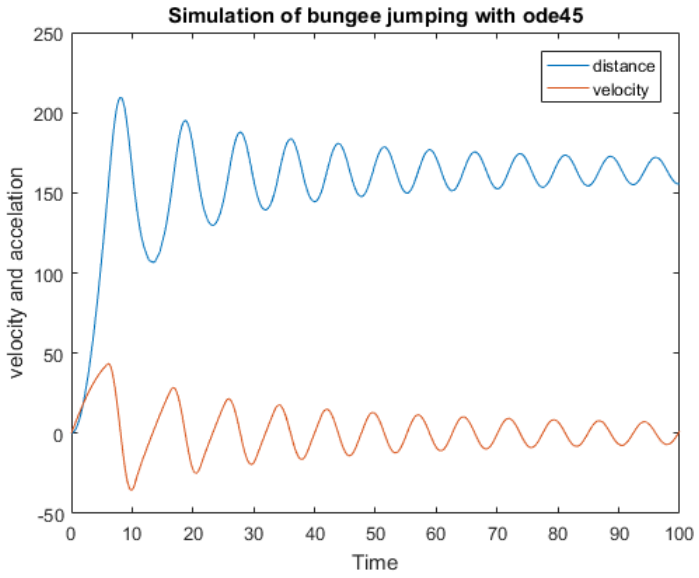

figure 1 results of bungee ODE use ode45 solver

2. Forward Euler and Classical Runge-Kutta

In this part, two functions were wrote as the numerical solver to solve ODE.

forwardeuler(func, t, h , y_0) solves ODE with forward Euler method

rungekutta(func, t, h , y_0) solves ODE with Runge-Kutta method.

Below are some results when we take t = [0 100], y_0 = [0 ; 0] for both method with different h (step of time):

When using forward euler method, choose h ≤ 1 will give stable solution and using Runge-Kutta method, choose h ≤ 4 will give stable solution. We can see that in figure 5 - figure 8 in appendix, forward euler got unstable result (it explode when we iterate) when we chose step size = 2. But when h ≤ 1, the results are stable and not getting closer to the result from ode45 solver. And figure 9 - figure 14we can see that Runge-Kutta got unstable result when h=5, and when we have h=0.1, the results are already good!

The error analysis are showed as follows when choose final time = 5, and plot log(h) and log(error). Forward euler is of order 1 in accuracy and Runge-Kutta is of oder 4 in accuracy.

Set h=1, 0.1, 0.01, 0.001 for forward euler and we can plot log(h) and log(error), it give a straight line and the slope shows the order of accuracy as in figure 2 that the order of accuracy for forward euler is 1. And similar for Runge-Kutta, the results are shown in figure 3, that the order of accuracy is 4.

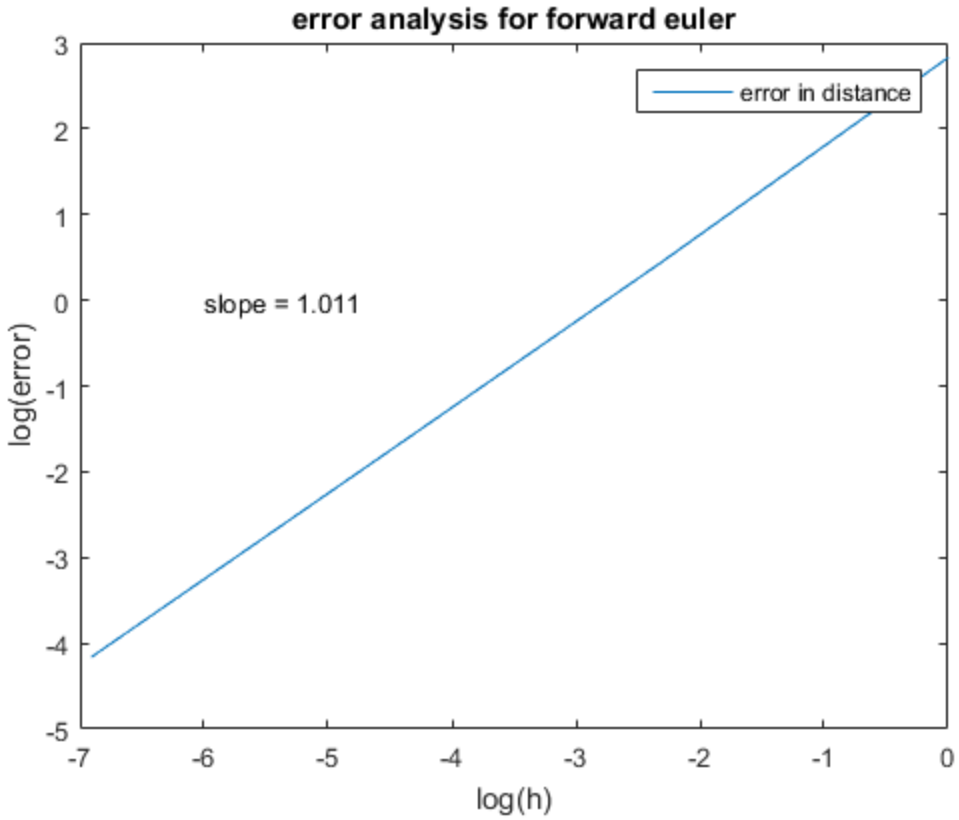

figure 2 error analysis for forward euler method

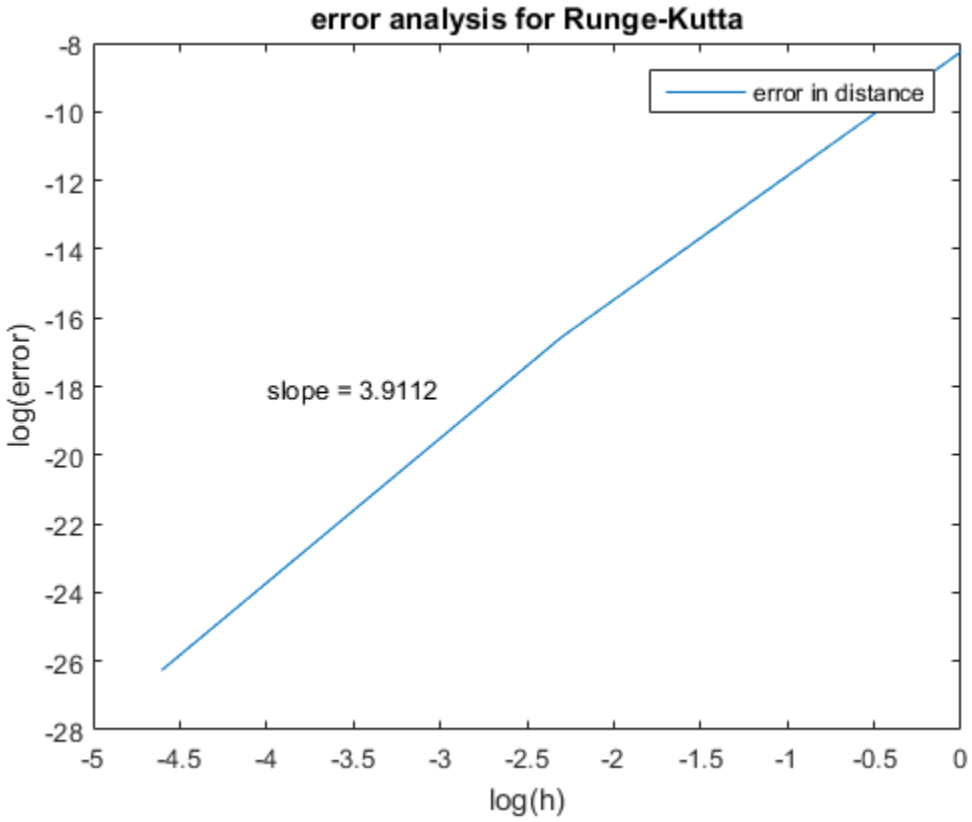

figure 3 error analysis for Runge-Kutta

3. Results

Using the formulas given in section 1, we can find the max acceleration. Max acceleration is found using runge kutta solver with h = 0.001. In figure 4 we can see that max acceleration happens at around 8s and its direction is upward, the acceleration is 32.84m/s². That is why bungee jumping is potential dangerous for you, as the rope pulls your harness upward when you are around 207m, these force will be transmitted to you via your harness, and since most harness is around your pelvis, it is your pelvis to handle the big acceleration upward, so it is hard on your spine, especially your lower back… so hopefully the harness is in good size with pads for the safety and comfort of the jumper. You will eventually stop at around 163 m after set final time to 1000s.

.

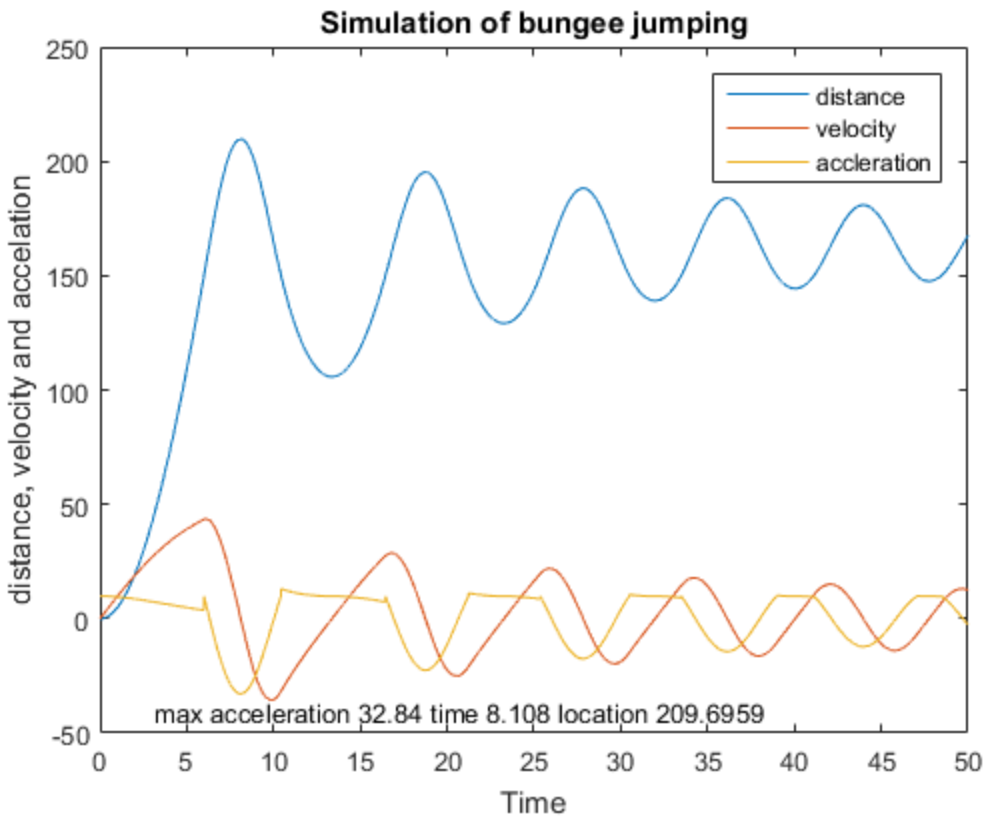

figure 4 max acceleration

4. Adjustment of rope length

Here, we need to adjust the rope to let the jumper reach not less than 20 meters from the water surface, that is y1 < 180.

We can simply set up a for loop in matlab using our runge kutta solver and let the rope length be from 1-200 at 1 m step and find out the max y1 then filter it with 180. We found the safety length is 122m.

5. Weight limit

Here we can use the similar idea above in find the best rope length to find our weight limit. By using a for loop setting testing weight from 1 to 200 at 1 kg step. The safety limit is 95 kg for a 5 m min distance from water and 104kg when the person just touches the water.

Or we can use the fzero by writing a function of m that returns ym as y1-195. And call fzero(@maxBungee, 70), we get 95.56 kg as our safety limit with 5m min distance from water and 104.15 kg when the person just touch the water. The result is similar to the looping method testing 200 different weights.

```
function ym = maxBungee(w)
  [t, y] = rungekutta(@(t,y)bungeeODE(t,y,w,122),[0 50], 0.01, [0;0]);
  ym =max(y(1,:))-195;
```

6. Correctness analysis

We take the effect of wind relative to speed. But not to surface presented to wind. If someone comes in trying to do a bungee jump with a high tech helmet and clothes that reduce the wind drag and try to dive in verticle as he jump off. I believe the model above will be with large error, But most people see bungee as a once in a life experience, so this is unlikely to happen. But still, it make sense to refine the model by adding the surface into equation. But it is difficult, some people may tend to dive in vertically, some people may tend to rotate at the platform when jumping and be more flat. This varies with different people.

In out model, we assumed the force from the rope is linear to the elongation and same for loading and unload. In reality, it might not be strict linear relationship and the loading path and unloading path may have small shift. But we can check this with the bungee rope manufacturer to refine our model.

We can also do some test jumps, using a speed meter with good accuracy to verify our model asking the jumper to leave the plat form with different body positions. We should choose a good day, with minimum wind to do these experiments.

And next important thing, choose a good harness for the safety and comfort of the jumper.

Appendix

1. Figures

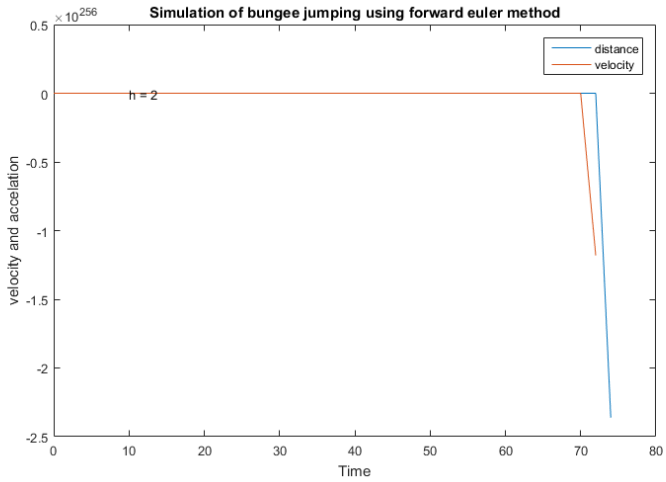

figure 5 forward euler method with h=2 (unstable result)

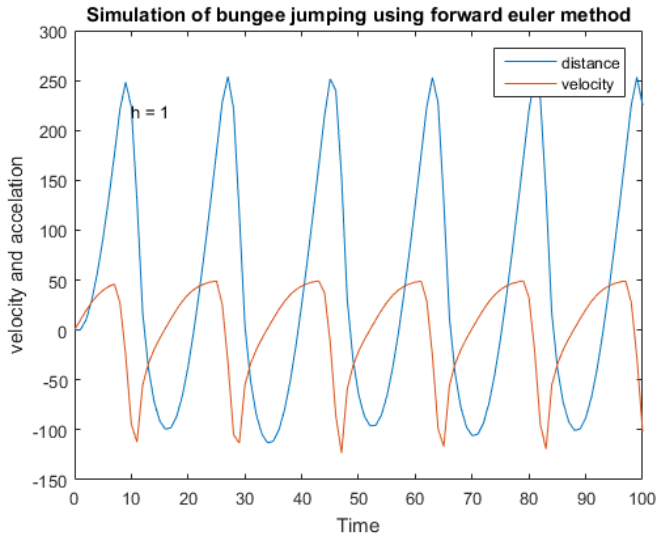

figure 6 forward euler method with h=1 (stable result but with large error)

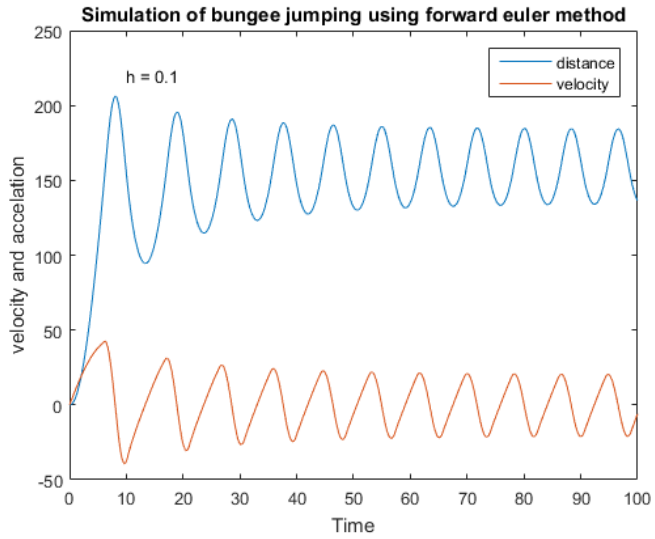

figure 7 forward euler method with h=0.1 (stable result)

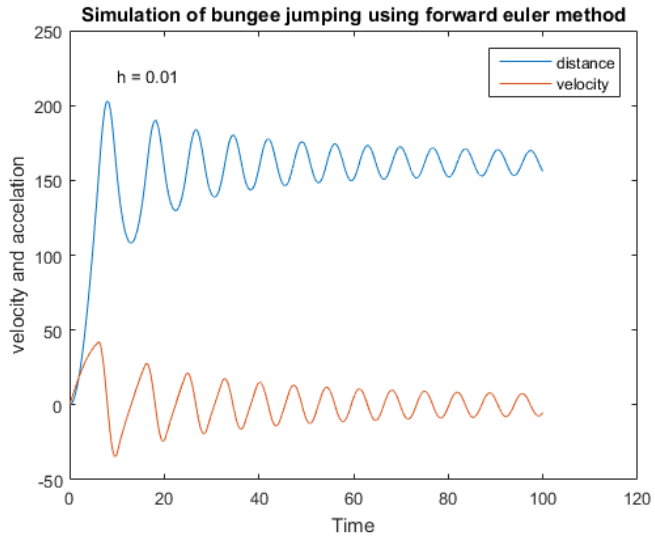

figure 8 forward euler method with h=0.01 (stable & good result)

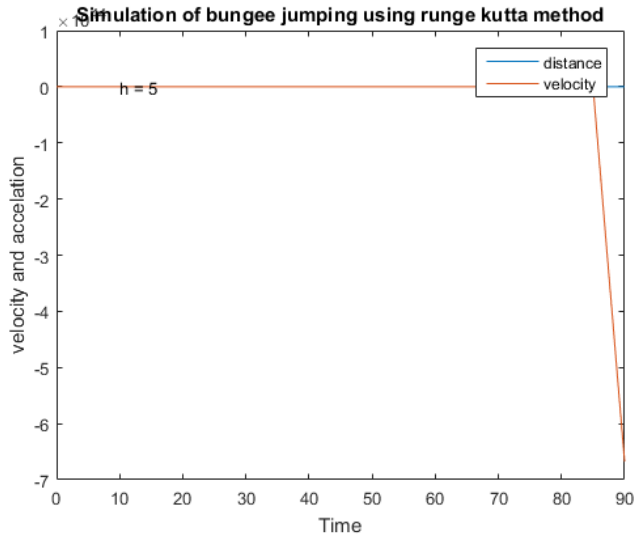

figure 9 Runge-Kutta method with h=5 (unstable result)

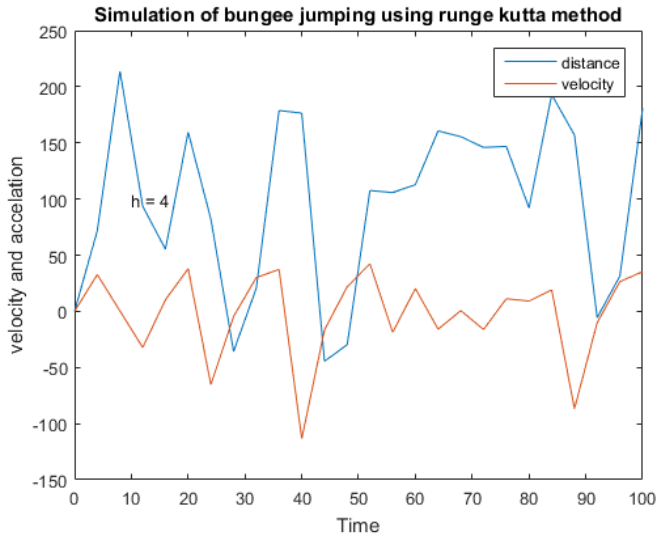

figure 10 Runge-Kutta method with h=4 (stable result but with large error)

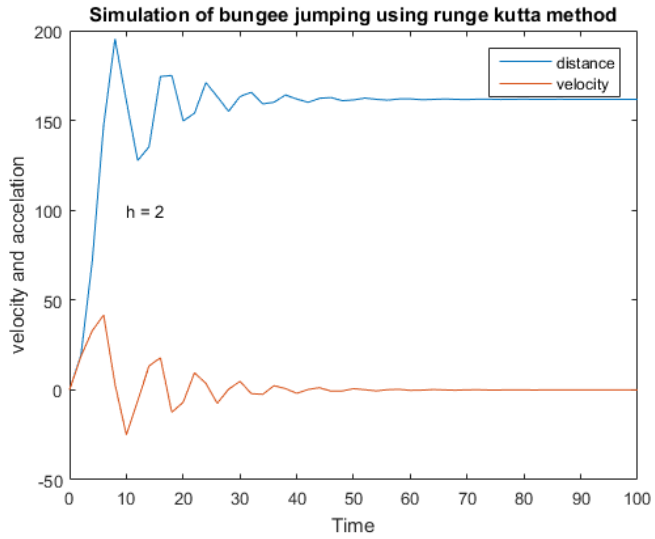

figure 11 Runge-Kutta method with h=2 (stable result)

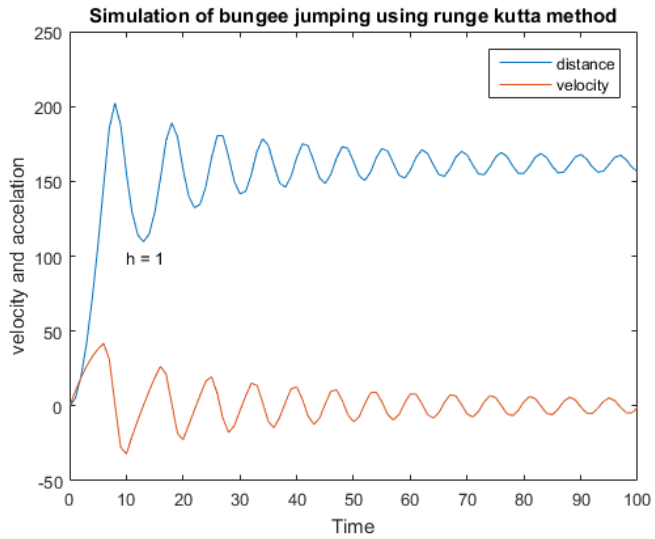

figure 12 Runge-Kutta method with h=1 (stable result)

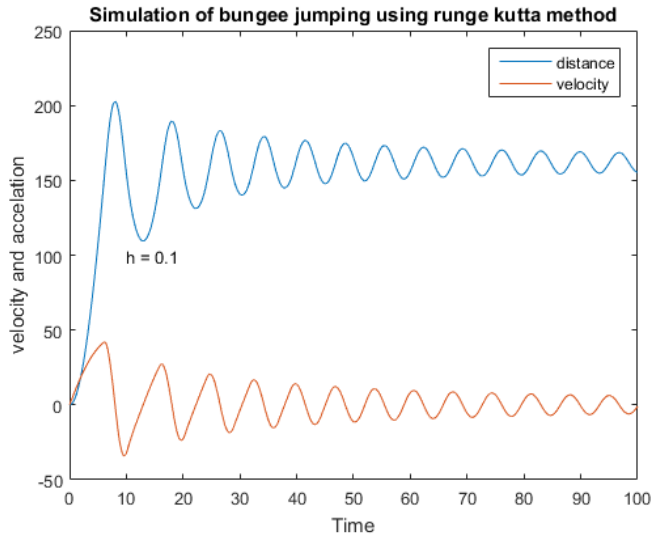

figure 13 Runge-Kutta method with h=0.1 (stable & good result)

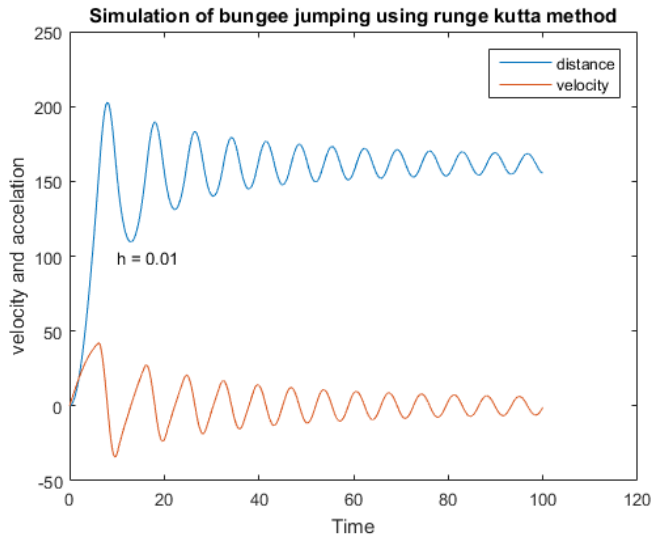

figure 14 Runge-Kutta method with h=0.01 (stable and good result)

2. Matlab files.

2.1 ODE equations

```
function y_out = bungeeODE(t, y, m, len)
% y_out = bungeeODE(t, y, m, len);
% call by @(t,y)@bungeeODE(t,y,m,len)
%
% Computes the right-hand side of the system of ODEs
%
% y is a vector with two components, y = [y1; y2]
% y1 is the distance and
% y2 is the velocity.
%
% m (weight of the person in kg) testing parameter
% g(gravity, 9.8)
% cv(coefficient for wind resistance 0.0227)
% len(lenth of the bungee rope) testing parameter

g = 9.8;
cv = 0.227;

if y(1) < len

y_out = [y(2); (m*g-sign(y(2))*cv*y(2)^2)/m];

else

y_out = [y(2); (m*g-sign(y(2))*cv*y(2)^2-50*(y(1)-len))/m];
end
```

2.2 forward euler solver

```
function [t, y_out] = forwardeuler(func, t, h, y_0)
 %return the solution of forward euler for solving ODE
 %call [t, y_out] = forwardeuler(func, t, h, y_0)
 %will return t and y_out

N = round(((t(2) - t(1))/h)); %number of iteration
y_out = [y_0(:), zeros(2,N)];
t = [t(1), zeros(1,N)];

  for i=1:N
    t(i+1)=t(i)+h;

    y_out(:,i+1) = y_out(:,i) + h*func(t(i), y_out(:,i));
  end

end
```

2.3 forward euler error analysis

```
clear all;

ti=0;
ops=odeset('RelTol',1e-9);
[t_true,y_true]=ode45(@(t,y)bungeeODE(t,y,70,150),[0 5],[0;0],ops);
h=1;

for i=1:4
   [t, y] = forwardeuler(@(t,y)bungeeODE(t,y,70,150),[0 5], h, [0;0]);
   y=y';
   err(i,:)= abs(y(end,:)-y_true(end,:));
   x(i)=h;
   h=h/10;

end
my_y=err(:,1)';

linearCoefficients = polyfit(log(x), log(my_y), 1)

plot(log(x),log(err(:,1)))
text(-6,0,['slope = ',num2str(linearCoefficients(1))]);
xlabel('log(h)')
ylabel('log(error)');
title ('error analysis for forward euler');
legend('error in distance');
```

2.4 Runge-Kutta solver

```
function [t, y_out] = rungekutta(func, t, h, y_0)
 %return the rungekutta results for solving ODE
 %call [t, y_out] = rungekutta(func, t, h, y_0)
 %will return t and y_out

N = round(((t(2) - t(1))/h)); %number of iteration
y_out = [y_0(:), zeros(2,N)];
t = [t(1), zeros(1,N)];

  for i=1:N
    t(i+1)=t(i)+h;
    k1 = h * func(t(i), y_out(:,i));
    k2 = h * func((t(i)+h/2), (y_out(:,i)+k1/2));
    k3 = h * func((t(i)+h/2), (y_out(:,i)+k2/2));
    k4 = h * func((t(i)+h), (y_out(:,i)+k3));

    y_out(:,i+1) = y_out(:,i) + k1/6 + k2/3 + k3/3 + k4/6;
  end

end
```

2.5 Runge-Kutta error analysis

```
clear all;

ti=0;

[t_true,y_true]=rungekutta(@(t,y)bungeeODE(t,y,70,150),[0 5], 0.0001,
[0;0]);
h=1;

for i=1:3
   [t, y] = rungekutta(@(t,y)bungeeODE(t,y,70,150),[0 5], h, [0;0]);

   err(:,i)= abs(y(:,end)-y_true(:,end));

   x(i)=h;
   h=h/10;

end


linearCoefficients = polyfit(log(x), log(err(1,:)), 1)

plot(log(x),log(err(1,:)))
text(-4,-18,['slope = ',num2str(linearCoefficients(1))])
xlabel('log(h)')
ylabel('log(error)');
title ('error analysis for Runge-Kutta');
legend('error in distance');
```

2.6 test script

You can choose your step size, time and continue play with forward euler and Runge-Kutta method in this script.

```
clear all;

cont = 0;

while cont==0

choosemethod = input('enter 1 for euler forward and enter 2 for RK 4:
')

h = input('enter h: ');
t1 = input('start time: ');
t2 = input('final time: ');

if choosemethod==1
    [t, y] = forwardeuler(@(t,y)bungeeODE(t,y,60,150),[t1 t2], h,
[0;0]);

elseif choosemethod==2
    [t, y] = rungekutta(@(t,y)bungeeODE(t,y,60,150),[t1 t2], h, [0;0]);

end


hdis = ['h = ', num2str(h)];
plot (t, y);
text(10,100, hdis);

   xlabel('Time')
   ylabel('velocity and accelation');
   title ('Simulation of bungee jumping');
   legend('distance','velocity');



disp('press 0 to continue')
cont = input('enter here ');
end
```

2.7 max acceleration

In this script, we compute max acceleration.

```
clear all;

m = 70;
g = 9.8;
cv = 0.227;
len = 150;

[t, y] = rungekutta(@(t,y)bungeeODE(t,y,70,150),[0 50], 0.001, [0;0]);

size= length(y);
for k=1:size
    if y(1,k)< 150
        acce(k) = (m*g-sign(y(2,k))*cv*(y(2,k))^2)/m;
    else
        acce(k) = (m*g-sign(y(2,k))*cv*(y(2))^2-50*(y(1,k)-len))/m;


    end

end

[max_acce,location]=max(abs(acce))
time=location*0.001;
[max_vel,abc]=max(y(2,:));
[max_dis,abcd]=max(y(1,:));

y=[y;acce];
y(1,location)
plot (t, y);
text(time-5,-40,['max acceleration ', num2str(max_acce), ' time ',
num2str(time),' location ',num2str(y(1,location))])
xlabel('Time')
ylabel('distance, velocity and accelation');
title ('Simulation of bungee jumping');
legend('distance','velocity','accleration');
```

2.8 adjust rope length

```
clear all;

for len=1:200

    [t, y] = rungekutta(@(t,y)bungeeODE(t,y,70,len),[0 50], 0.01,
[0;0]);

    r(len) = max(y(1,:));

    if r(len)>180
        r(len)=0;
    end

end

[M,length] = max(r);

length
```

2.9 safety weight

```
 clear all;


for a=1:200

    [t, y] = rungekutta(@(t,y)bungeeODE(t,y,a,122),[0 50], 0.01,
[0;0]);

    w(a) = max(y(1,:));

    if w(a)>200
        w(a)=0;
    end

end


[M,safe_weight] = max(w);

safe_weight
```