Bungee Jump Problem with Runge-Kutta Method

2016039034 박준형
2018021794 염태은

CONTENTS

Introduction

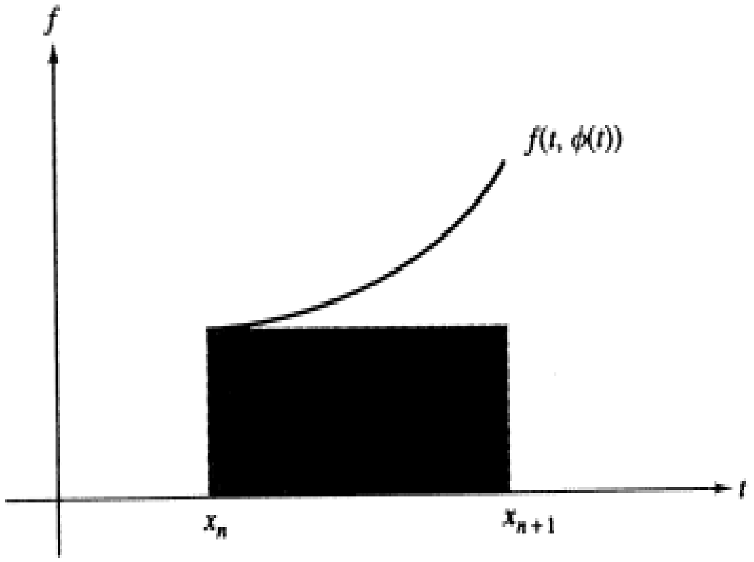


FIG. 1. *Approximation by a rectangle*

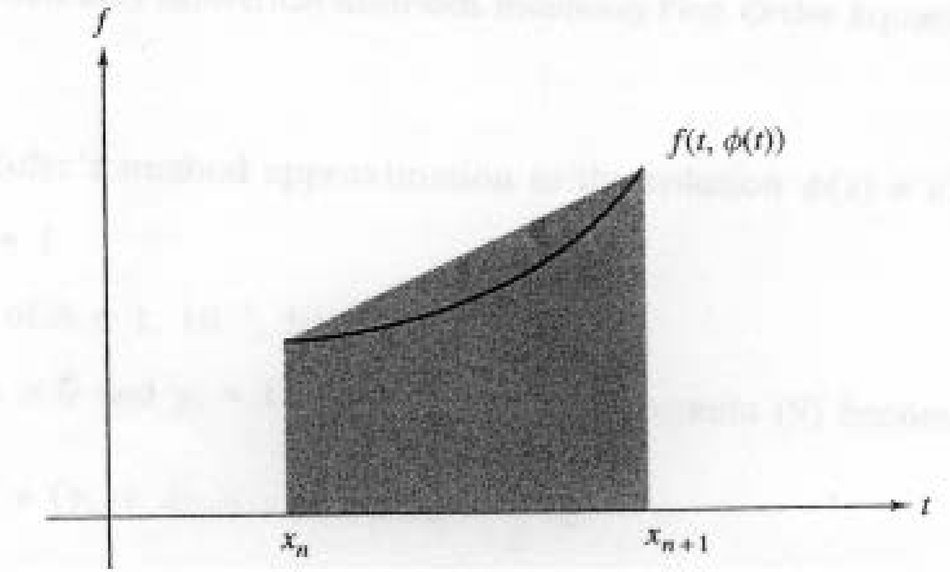


FIG. 2. *Approximation by a trapezoid*

$$y_{n+1} = y_n + hf(x_n, y_n)$$

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_n + h, y_n + hf(x_n, y_n))]$$

Introduction

$$y_{i+1} \simeq y_i + f(t_i, y_i)h + \frac{1}{2!}f'(t_i, y_i)h^2 + \frac{1}{3!}f''(t_i, y_i)h^3 + \frac{1}{4!}f'''(t_i, y_i)h^3$$

$$\int_a^b f(x)\,dx \approx \int_a^b P(x)\,dx = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$: Simpson’s Rule

$$\int_{t_n}^{t_n+h} f\left(t, \varphi(t)\right)\,dt \approx \frac{h}{6}\left[f\left(t_n, \varphi(t_n)\right) + 4f\left(t_n + \frac{h}{2}, \varphi(t_n + \frac{h}{2})\right) + f\left(t_n + h, \varphi(t_n + h)\right)\right]$$

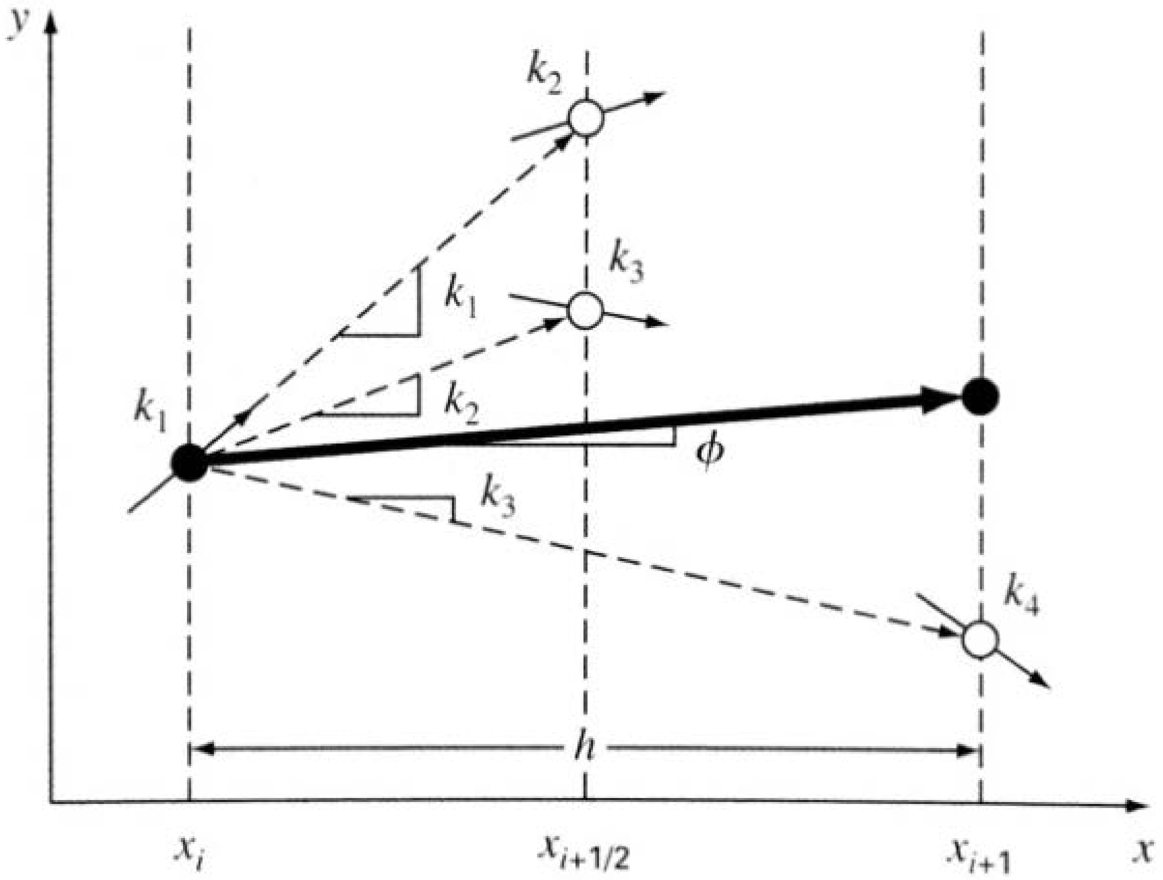
Runge-Kutta Method

$$k_1 = f(t_i, y_i)$$

$$k_2 = f(t_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}k_1 h)$$

$$k_3 = f(t_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}k_2 h)$$

$$k_4 = f(t_i + h, y_i + k_3 h)$$



$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$

Bungee-Jump Problem

$x =$ position (m) ,
$v =$ downward vertical velocity (m/s),
$t =$ time (s),
$g =$ the acceleration due to gravity $(\simeq 9.81\ m/s^2)$,
$c_d =$ a lumped drag coefficient (kg/m),
$m =$ the jumper's mass (kg)

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v \quad , \quad \frac{\mathrm{d}v}{\mathrm{d}t} = g - \frac{\mathrm{c_d}}{m} v^2$$

$x = 0\ (m)$, $v = 0\ (m/s)$,
$t = 0\ (s)$ to $t = 10\ (s)$, $h = 2$,
$g \simeq 9.81\ m/s^2$, $c_d = 0.25\ (kg/m)$, $m = 68.1\ (kg)$

Analytical Solution

$$v(t) = \sqrt{\frac{gm}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}}t\right)$$

$$x(t) = \frac{m}{c_d} \ln\left[\cosh\left(\sqrt{\frac{gc_d}{m}}t\right)\right]$$

Matlab Code

```
h=2;                                          % step size
t = 0:h:10;
v = zeros(1,length(t));
x = zeros(1,length(t));

F_v = @(t,x,v) 9.81-(0.25/68.1)*v^2;          % dv/dt function
F_x = @(t,x,v) v;                             % dx/dt function

for i=1:(length(t)-1)                         % calculation loop

    %For dv/dt function
    k_1 = F_v(t(i),x(i),v(i));
    k_2 = F_v(t(i)+0.5*h,x(i)+0.5*h*k_1,v(i)+0.5*h*k_1);
    k_3 = F_v(t(i)+0.5*h,x(i)+0.5*h*k_2,v(i)+0.5*h*k_2);
    k_4 = F_v((t(i)+h),(x(i)+k_3*h),(v(i)+k_3*h));

    % For dx/dt function
    kk_1 = F_x(t(i),x(i),v(i));
    kk_2 = F_x(t(i)+0.5*h,x(i)+0.5*h*kk_1,v(i)+0.5*h*k_1);
    kk_3 = F_x(t(i)+0.5*h,x(i)+0.5*h*kk_2,v(i)+0.5*h*k_2);
    kk_4 = F_x((t(i)+h),(x(i)+kk_3*h),(v(i)+k_3*h));

    % main equation
    v(i+1) = v(i) + (1/6)*(k_1+2*k_2+2*k_3+k_4)*h;
    x(i+1) = x(i) + (1/6)*(kk_1+2*kk_2+2*kk_3+kk_4)*h;

end
```

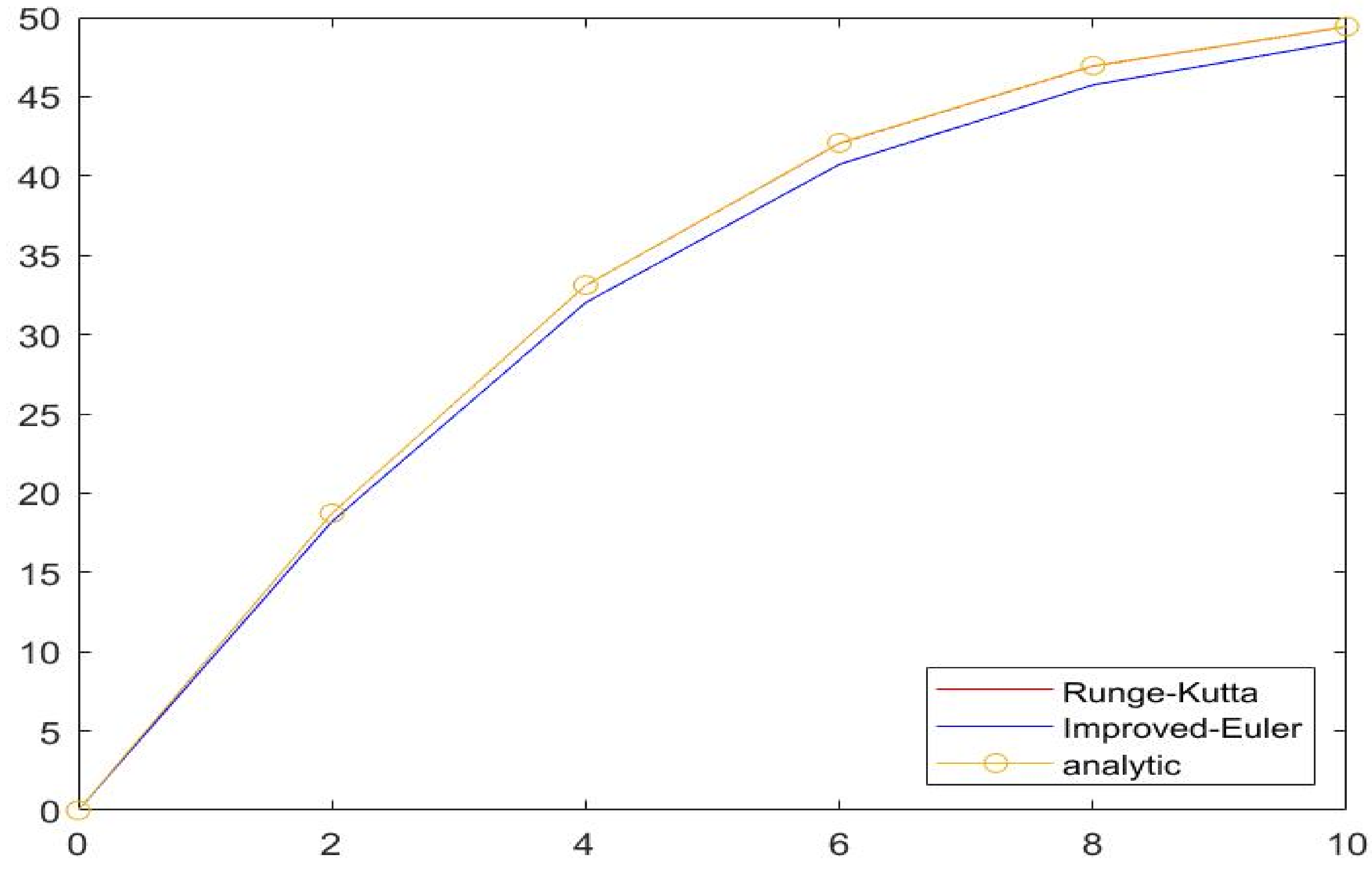
Matlab Code

```
%improved Euler Method
v_E = zeros(1,length(t));
x_E = zeros(1,length(t));

for i=1:(length(t)-1)                               % calculation loop

    v_E(i+1) = v_E(i) + (1/2)*(F_v(t(i),x_E(i),v_E(i))+F_v(t(i+1),x_E(i)+F_x(t(i),x_E(i),v_E(i))*h,v_E(i)+F_v(t(i),x_E(i),v_E(i))*h))*h;
    x_E(i+1) = x_E(i) + (1/2)*(F_x(t(i),x_E(i),v_E(i))+F_x(t(i+1),x_E(i)+F_x(t(i),x_E(i),v_E(i))*h,v_E(i)+F_v(t(i),x_E(i),v_E(i))*h))*h;

end

%plot(t,x,'o-')

V = sqrt(9.81*68.1/0.25)*tanh(sqrt(9.81*0.25/68.1)*t);     % 시간 속도 그래프 식
X = 68.1/0.25*log(cosh(sqrt(9.81*0.25/68.1)*t));            % 시간 위치 그래프 식
plot(t,v,'b');
hold on;
plot(t,v_E,'m');
hold on;
plot(t,V,'k');
legend({'Runge-Kutta','Improved-Euler','analytic'},'Location','southeast')

% 시간 위치
plot(t,x,'b');
hold on;
plot(t,x_E,'m');
hold on;
plot(t,X,'k');
legend({'Runge-Kutta','Improved-Euler','analytic'},'Location','southeast')
```

Result_velocity

t	$v_{analytic}$	v_{RK4}	$v_{Improved.Euler}$
0	0	0	0
2	18.7291888456973	18.7255538969182	18.2068414096916
4	33.1118250352482	33.0994736464891	32.0112996015369
6	42.0762270564987	42.0547480252525	40.7274572790186
8	46.9574951289851	46.9344762262423	45.7403931632060
10	49.4213669186913	49.4027393473134	48.5024210593411

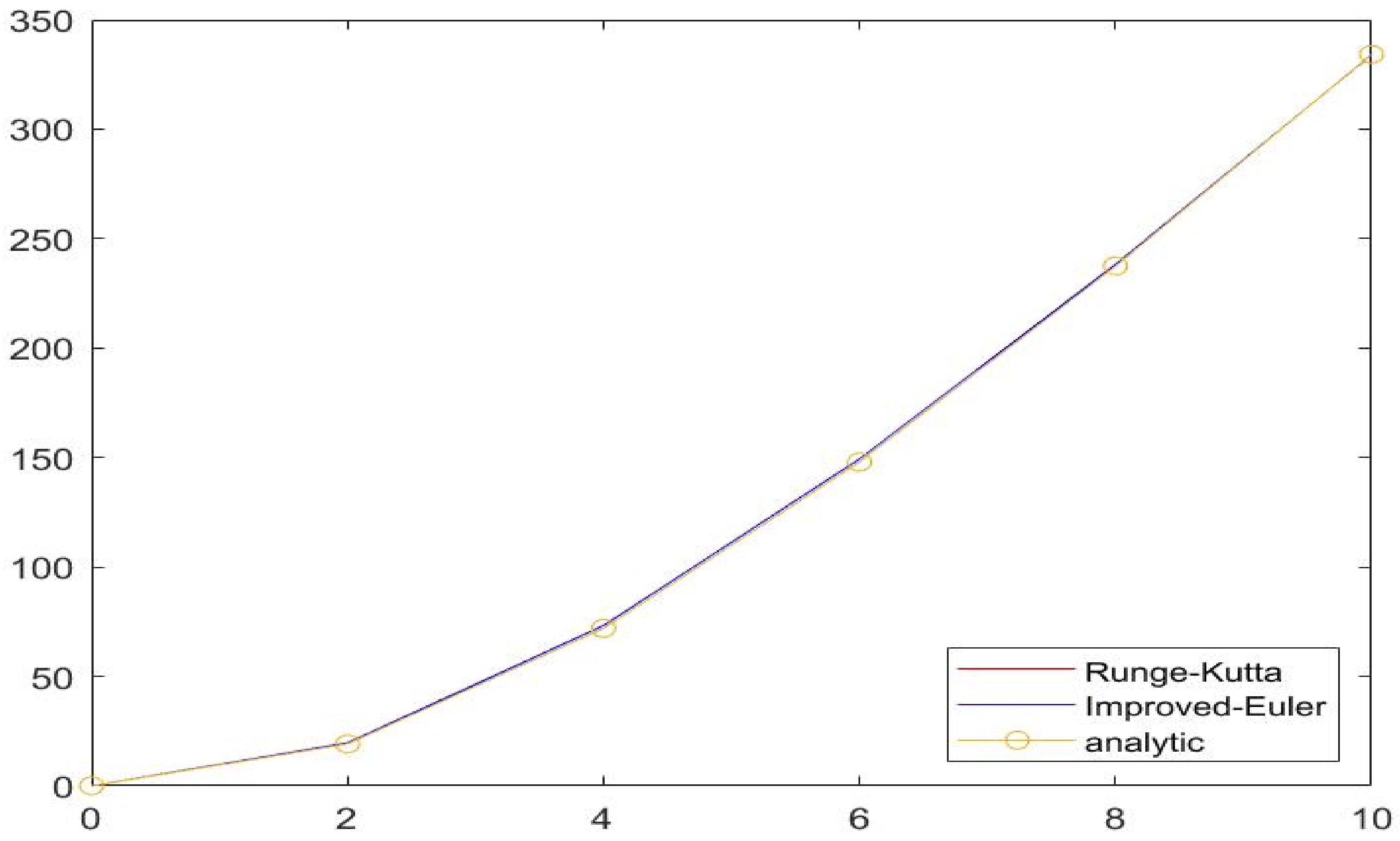
Graph_velocity



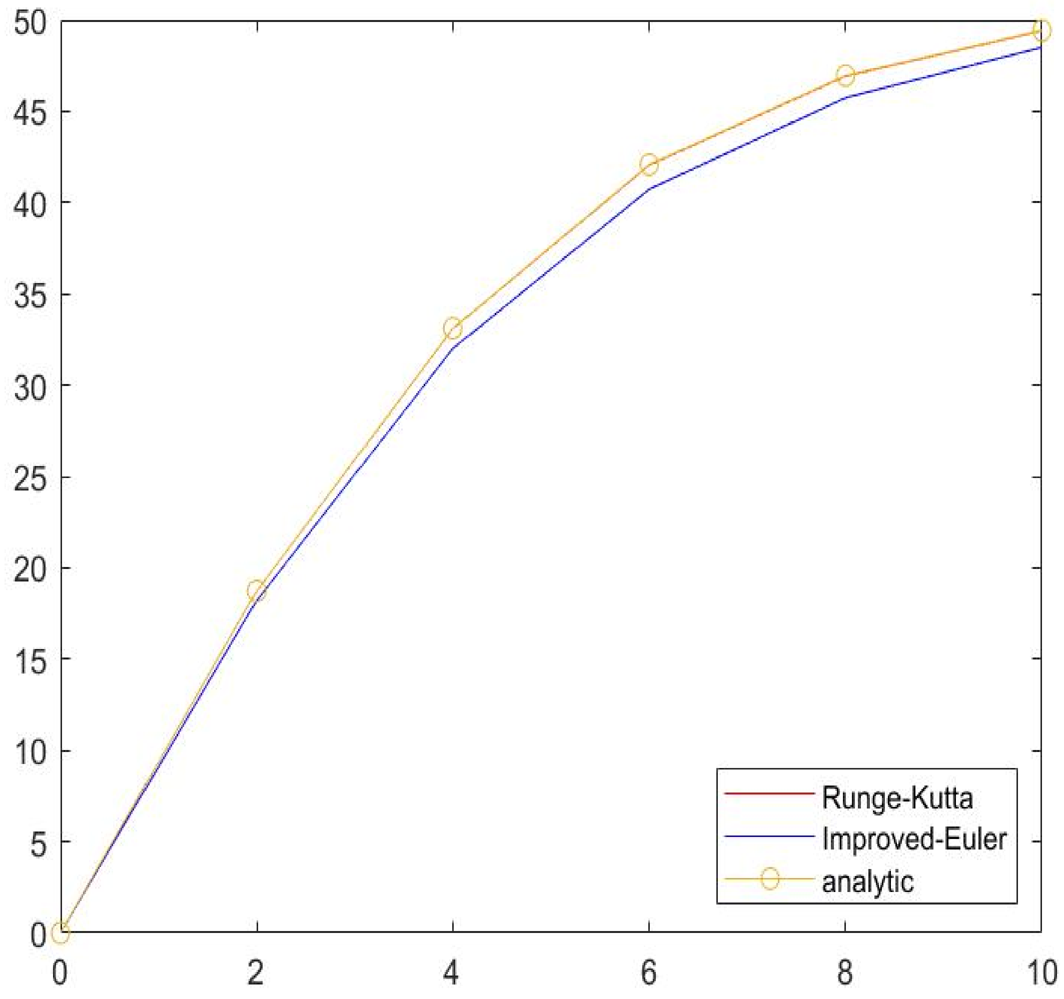
Result_position

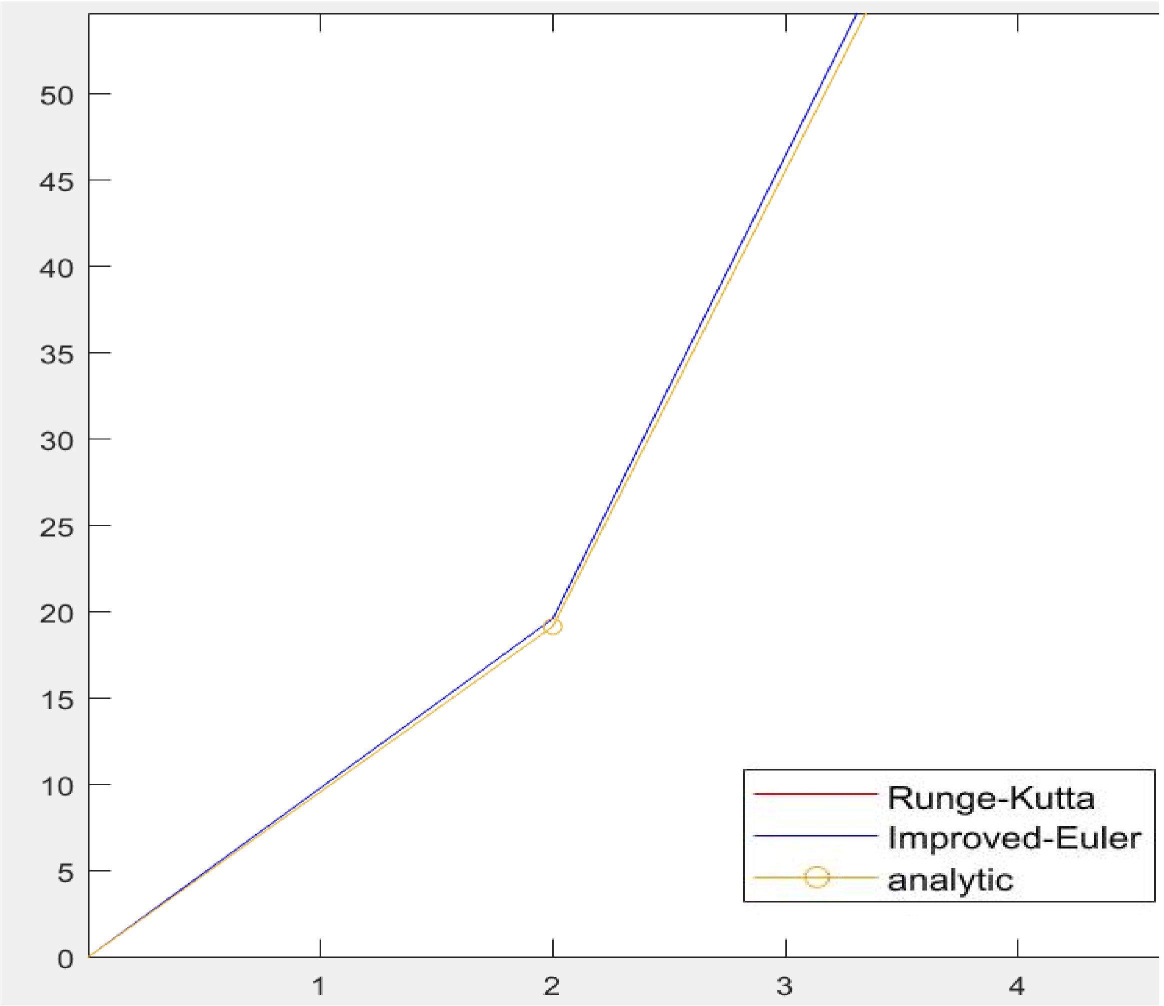
t	$x_{analytic}$	x_{RK4}	$x_{Improved.Euler}$
0	0	0	0
2	19.1662861446833	19.1656057986059	19.6200000000000
4	71.9303659297974	71.9311162115826	73.2198423339379
6	147.946179735052	147.952142086551	149.338775588558
8	237.510440116875	237.510389811099	238.235086795536
10	334.178167247408	334.162579770609	333.974760297244

Graph_position



Graph_position





Error same step size

Error : analytic sol – numerical sol

t	$Imporved\ Euler's\ method$	$Runge-Kutta\ method$
X	0.918945859350210	0.018627571377948 0
V	0.203406950163412	0.015587476798373 2

4th order Runge-Kutta Method is more accurate

Error_v_different step size

Step Size	Number of Steps	$v_{analytic}$ - $v_{Improved.Euler}$	$v_{analytic}$ - v_{RK4}
2	5	0.918945859350210	0.018627571377948
2/10	50	0.00529396853441000	1.17660835741162e-06
2/100	500	5.06429714590695e-05	1.12542863917042e-10
2/1000	5000	5.04235423193222e-07	7.10542735760100e-15

Improved Euler Method : $O(h^2)$, $4\text{th Runge} - \text{Kutta Method}$: $O(h^4)$

Error_x_different step size

Step Size	Number of Steps	$x_{analytic}$ - $x_{Improved.Euler}$	$x_{analytic}$ - x_{RK4}
2	5	0.203406950163412	0.0155874767983732
2/10	50	0.00440631136120828	3.46563643915943e-06
2/100	500	4.26764880785413e-05	3.63002072845120e-10
2/1000	5000	4.25090888711566e-07	3.41060513164848e-13

Improved Euler Method : $O(h^2)$, $\mathrm{4th\ Runge-Kutta\ Method}$: $O(h^4)$