AppliedW7

April 11, 2024

1 Question 1

Using the chain rule, we have

$$\frac{dy}{dt} = \frac{dy}{dx} \cdot \frac{dx}{dt}$$

or

$$-bxy = \frac{dy}{dx} \cdot (-ay)$$

Rearranging and solving

$$\frac{dy}{dx} = \frac{bx}{a},$$

we get

$$y = \frac{b}{2a}x^2 + c,$$

for some constant c. At $t = 0$, we take $x(0) = x_0$ and $y(0) = y_0$. Substituting this condition into the general solution, we obtain

$$c = y_0 - \frac{b}{2a}x_0^2.$$

We are interested in answering the questions "is it possible to send a specific number of enemy soldiers that will guarantee a target level of casualties in the enemy army?". Let the target number of causalities be p, and so when a steady state is reached, we require that $y = y_0 - p$. Assuming $y_0 \neq p$, we then require $x = 0$ to reach a steady state. Substituting this into the expression for y, we obtain:

$$\begin{aligned} y &= \frac{b}{2a}x^2 + y_0 - \frac{b}{2a}x_0^2 \\ \implies y_0 - p &= y_0 - \frac{b}{2a}x_0^2 \\ \implies p &= \frac{b}{2a}x_0^2 \end{aligned}$$

As this expression does not depend on y_0, we are unable to determine how many soldiers the enemy needs to send to result in p casualties, however we can tell how many soldiers the home team must send to obtain p casualties in the enemy team (This isn't something the enemy team has control over though).

From this analysis, we obtain an expression for the number of casualties the enemy team suffers when they win. Clearly, this results depends on a, b, and x_0.

On the other hand, if we let p be the number of casualties in the home army and we consider the steady state where they win, we have $x = x_0 - p$ when $y = 0$. Substituting this into the expression for y, and rearranging for p, we obtain:

$$p = x_0 - \sqrt{x_0^2 - \frac{2a}{b} y_0}$$

This implies that if the home team wins, then the number of casualties they will suffer is dependent on a, b, x_0, and y_0.

2 Question 2

The problem is given by:

$$y' = \frac{dy}{dx} = f(x, y)$$

with $y_0 = y(x_0)$.

Taylor expansion:

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2!} + \dots$$

For one iteration, we note $x_{i+1} = x_i + h$.

$$y_{i+1} = y(x_{i+1}) = y(x_i + h) = y(x_i) + hy'(x_i) + \frac{h^2}{2} y''(x_i) + \dots$$

Ignoring the quadratic and higher terms:

$$y_{i+1} \approx y(x_i) + hy'(x_i)$$

$$y_{i+1} \approx y_i + hy'(x_i)$$

which given the statement of the problem is:

$$y_{i+1} \approx y_i + hf(x_i, y_i)$$

$$x_{i+1} = x_i + h$$

i	x_i	y_i
0	0	1
1	1	1
2	2	2
3	3	6
4	4	24

One way to conceptualise this ODE solvers is to think of the estimation based on a slope s_1:

$$y_{i+1} = y_i + h * s_1$$

The Euler estimation uses $s_1 = f(x_i, y_i)$, which is the slope at (x_i, y_i).

We can also think of the slope at (x_{i+1}, y_{i+1}), s_2, given by:

$$s_2 = f(x_{i+1}, y_{i+1})$$

but noting $y_{i+1} = y_i + hf(x_i, y_i)$, and $x_{i+1} = h + x_i$. Thus we get:

$$s_2 = f(h + x_i, y_i + hf(x_i, y_i))$$

Heun's estimation uses $s = \frac{s_1+s_2}{2}$. So the difference equation is given by:

$$y_{i+1} = y_i + \frac{h}{2}\Big(f(x_i, y_i) + f(x_i + h, y_i + hf(x_i, y_i)\Big)$$

[57]:
```
import numpy as np
from matplotlib import pyplot as plt
```

2.1 Euler's 1D

[58]:
```
def euler_next(df, y, t, h):
    return y + h*df(t, y)
```

[76]:
```
def euler(df, y0, t0=0, h=0.1, max_iter=1000, verbose=False):
    t = np.zeros(max_iter+1)
    y = np.zeros(shape=max_iter+1)
    t[0], y[0] = t0, y0

    for i in range(max_iter):
        if verbose: print('iteration: {0}, t={1}, y={1}'.format(str(i),␣
 ↪str(t[i]), str(y[i])))
        t[i+1] = t[i] + h
        y[i+1] = euler_next(df=df, y=y[i], t=t[i], h=h)
    return t, y
```

2.2 RK2 1D

[60]:
```
def RK2_next(df, y, t, h, b):
    if b == 0:
        a, alpha, beta = 1, 1, 1
    else:
        a, alpha, beta = 1-b, 1/(2*b), 1/(2*b)

    k1 = df(t, y)
    k2 = df(t+alpha*h, y+beta*k1*h)
    return y + h*(a*k1 + b*k2)
```

[77]:
```
def RK2(df, y0, t0=0, h=0.1, max_iter=1000, b=0.5, verbose=False):
    t = np.zeros(max_iter+1)
    y = np.zeros(shape=max_iter+1)
```

```python
    t[0], y[0] = t0, y0
    for i in range(max_iter):
        if verbose: print('iteration: {0}, t={1}, y={1}'.format(str(i),␣
 ↪str(t[i]), str(y[i])))
        t[i+1] = t[i] + h
        y[i+1] = RK2_next(df=df, y=y[i], t=t[i], h=h, b=b)
    return t, y
```

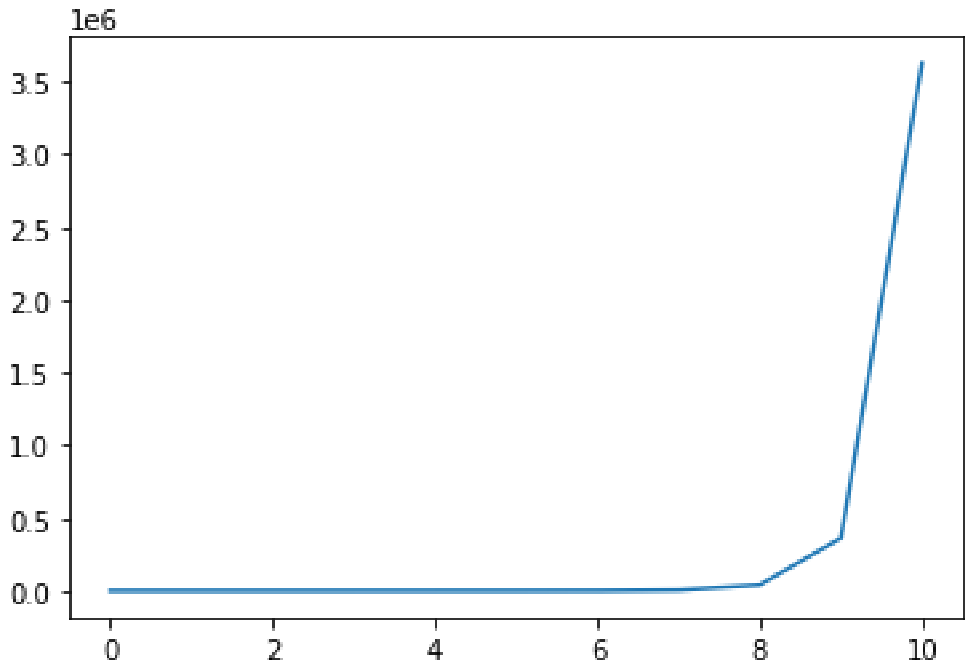
[78]:
```python
dydx = lambda x, y: x*y
y0 = 1
x0=0
h=1
max_iter=10
```

[79]:
```python
x, y = euler(df=dydx, y0=y0, t0=x0, max_iter=max_iter, h=h, verbose=True)
```

```
iteration: 0, t=0.0, y=0.0
iteration: 1, t=1.0, y=1.0
iteration: 2, t=2.0, y=2.0
iteration: 3, t=3.0, y=3.0
iteration: 4, t=4.0, y=4.0
iteration: 5, t=5.0, y=5.0
iteration: 6, t=6.0, y=6.0
iteration: 7, t=7.0, y=7.0
iteration: 8, t=8.0, y=8.0
iteration: 9, t=9.0, y=9.0
```

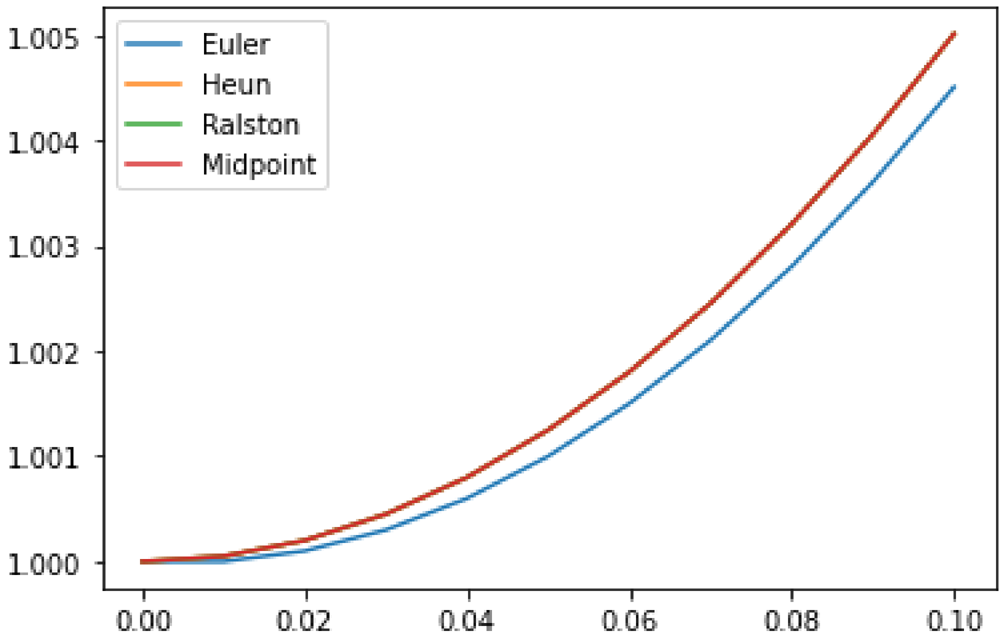
[80]:
```python
plt.plot(x, y)
```

[80]:
```
[<matplotlib.lines.Line2D at 0x7fe15c7b5ca0>]
```



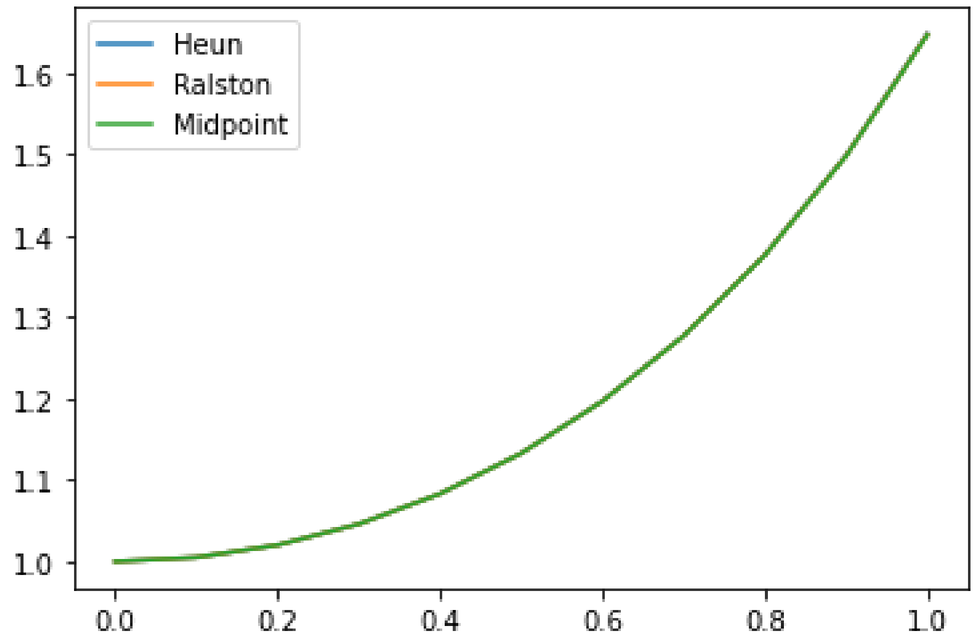
[81]:
```
B = [0, 0.5, 2/3, 1]
h=0.01
labels = ['Euler', 'Heun', 'Ralston', 'Midpoint']
for i, b in enumerate(B):
    x, y = RK2(df=dydx, y0=y0, t0=x0, max_iter=max_iter, h=h, verbose=False, b=b)
    plt.plot(x, y, label=labels[i])
plt.legend()
```

[81]: <matplotlib.legend.Legend at 0x7fe15c901130>



[82]:
```
B = [0.5, 2/3, 1]
h=0.1
labels = ['Heun', 'Ralston', 'Midpoint']
for i, b in enumerate(B):
    x, y = RK2(df=dydx, y0=y0, t0=x0, max_iter=max_iter, h=h, verbose=False, b=b)
    plt.plot(x, y, label=labels[i])
plt.legend()
```

[82]: <matplotlib.legend.Legend at 0x7fe15c849220>



3 Question 3

3.1 Dynamical Systems

[83]:
```
def euler_DS(df, y0, t0=0, h=0.1, max_iter=1000, verbose=False):
    t = np.zeros(max_iter+1)
    y = np.zeros(shape=(len(y0), max_iter+1))
    t[0], y[:, 0] = t0, y0

    for i in range(max_iter):
        if verbose: print('iteration: {0}, t={1}, y={1}'.format(str(i),␣
↪str(t[i])))
        t[i+1] = t[i] + h
        y[:, i+1] = euler_next(df=df, y=y[:, i], t=t[i], h=h)
    return t, y
```

[84]:
```
def RK2_DS(df, y0, t0=0, h=0.1, b=0.5, max_iter=1000, verbose=False):
    t = np.zeros(max_iter+1)
    y = np.zeros(shape=(len(y0), max_iter+1))
    t[0], y[:, 0] = t0, y0

    for i in range(max_iter):
        if verbose: print('iteration: {0}, t={1}, y={1}'.format(str(i),␣
↪str(t[i])))
        t[i+1] = t[i] + h
```

```
        y[:, i+1] = RK2_next(df=df, y=y[:, i], t=t[i], h=h, b=b)
    return t, y
```

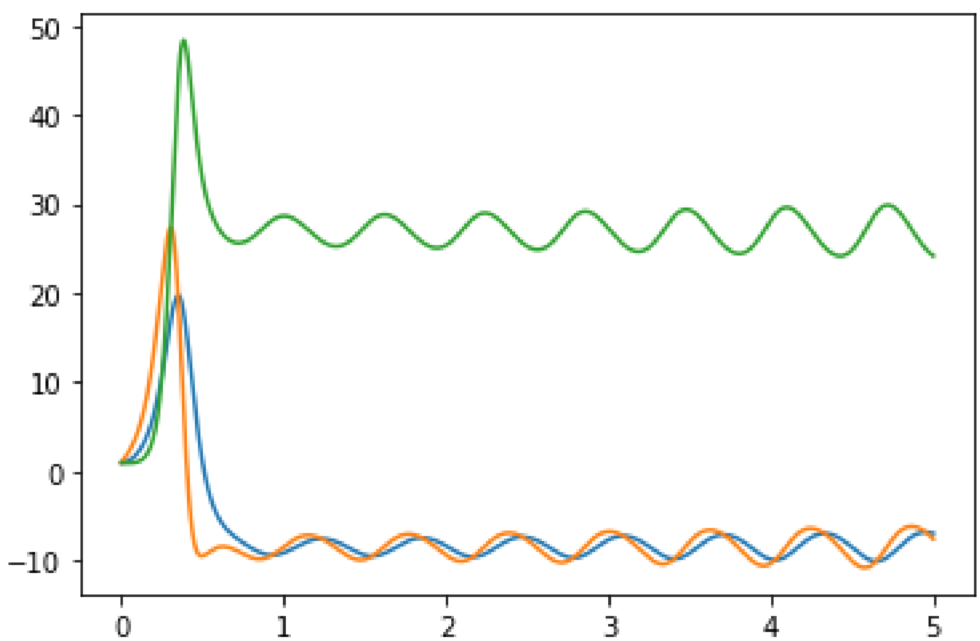
[85]:
```
sigma = 10
beta = 8/3
rho = 28
xyz0 = np.array([1, 1, 1])
t0 = 0
h=0.001
dxdt = lambda t, xyz: sigma*(xyz[1] - xyz[0])
dydt = lambda t, xyz: xyz[0]*(rho-xyz[2]) - xyz[1]
dzdt = lambda t, xyz: xyz[0]*xyz[1] - beta*xyz[2]

dxyzdt = lambda t, xyz: np.array([dxdt(t, xyz), dydt(t, xyz), dzdt(t, xyz)])
```

[87]:
```
t, (x,y,z) = euler_DS(df=dxyzdt, y0=xyz0, t0=t0, h=h, max_iter=5000,␣
 ↪verbose=False)
```

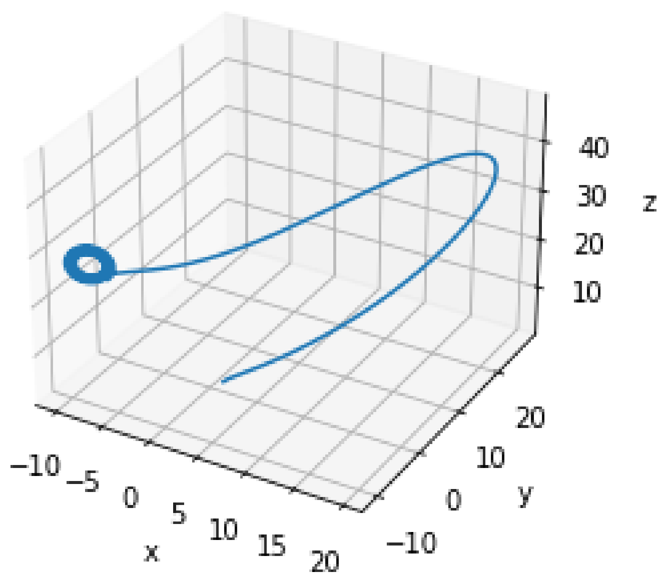
[88]:
```
plt.plot(t, x, label='x')
plt.plot(t, y, label='y')
plt.plot(t, z, label='z')
```

[88]: [<matplotlib.lines.Line2D at 0x7fe15c6a7cd0>]



[89]:
```
axs = plt.figure().add_subplot(projection='3d')
axs.plot(x, y, z)
axs.set_xlabel('x')
axs.set_ylabel('y')
axs.set_zlabel('z')
```

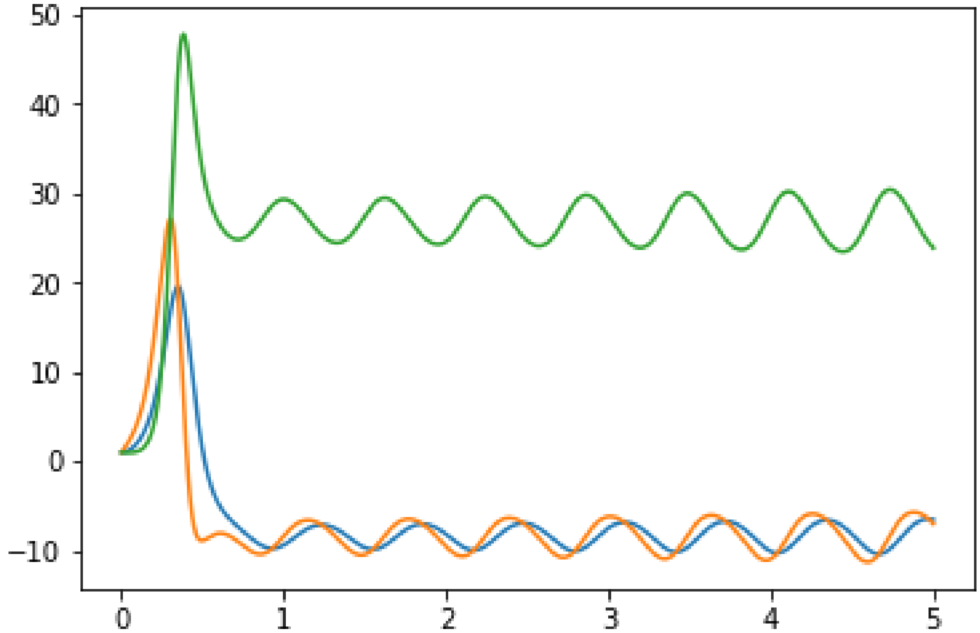
[89]: Text(0.5, 0, 'z')



[90]:
```
b=0.5
t, (x,y,z) = RK2_DS(df=dxyzdt, y0=xyz0, t0=t0, h=h, b=0.5, max_iter=5000,␣
↪verbose=False)
```

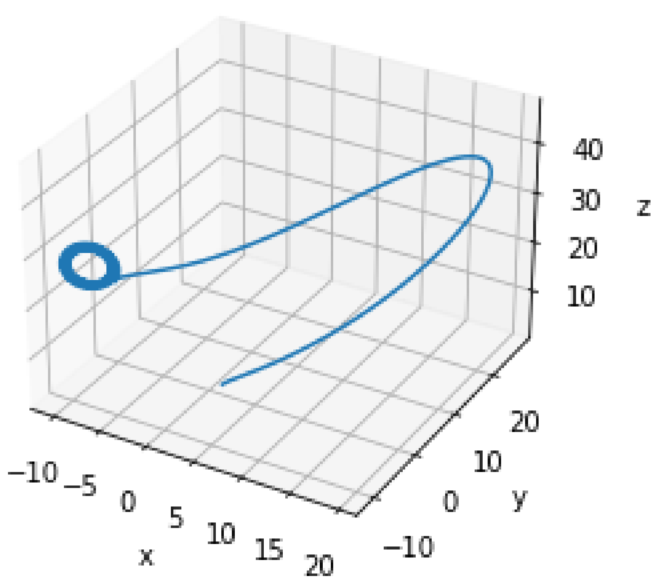
[91]:
```
plt.plot(t, x, label='x')
plt.plot(t, y, label='y')
plt.plot(t, z, label='z')
```

[91]: [<matplotlib.lines.Line2D at 0x7fe15c5c32b0>]



[92]:
```
axs = plt.figure().add_subplot(projection='3d')
axs.plot(x, y, z)
axs.set_xlabel('x')
axs.set_ylabel('y')
axs.set_zlabel('z')
```

[92]: Text(0.5, 0, 'z')



```
[ ]:
```

```
[ ]:
```

4 Question 4

The model, as stated above, is given by:

$$\frac{dx}{dt} = -ay$$

$$\frac{dy}{dt} = -bxy$$

Euler's Method schema:

$$\begin{aligned} x_{i+1} &= x_i - hay_i \\ y_{i+1} &= y_i - hbx_iy_i \\ t_{i+1} &= t_i + h \end{aligned}$$

Heun's method schema:

$$\begin{aligned} x_{i+1} &= x_i + \frac{h}{2}\left(-ay_i - a\left(y_i - hbx_iy_i\right)\right) \\ &= x_i - \frac{h}{2}\left(2ay_i - habx_iy_i\right) \\ y_{i+1} &= y_i + \frac{h}{2}\left(-bx_iy_i - b\left(x_i - hay_i\right)\left(y_i - hbx_iy_i\right)\right) \\ &= y_i - \frac{hb}{2}\left(x_iy_i - \left(x_i - hay_i\right)\left(y_i - hbx_iy_i\right)\right) \\ t_{i+1} &= t_i + h \end{aligned}$$

5 Question 5

The differential equation from question 3 is

$$\frac{dy}{dx} = xy,$$

and so the schema required is:

$$\begin{aligned} & y_{i+1} = y_i + hx_{i+1}y_{i+1} \\ \implies & y_{i+1} = \frac{y_i}{1 - hx_{i+1}} = \frac{y_i}{1 - h(x_i + h)} \\ & x_{i+1} = x_i + h \end{aligned}$$

Repeating for the new differential equation, we obtain:

$$\begin{aligned} y_{i+1} &= y_i + h\sin(y_{i+1}) \\ x_{i+1} &= x_i + h \end{aligned}$$

However, we can notice this time that we are not able to find an explicit expression for y_{i+1}. When calculating y_{i+1}, the value of y_i is known, and since h is known, the only unknown variable in the schema is y_{i+1}. This allows us to use a root finding technique (such as Bisection method, Newton's method or the Secant Method) to find our next value y_{i+1}. In terms of computation time to run this method compared to Euler's Forward method, this method would take a much greater amount of time since to calculate each new value, a root finding algorithm is required. This method is more stable than Euler's Forward Method - it will converge for some problems that Euler's Forward method will not.