Table of Contents

Solutions to ODE Worksheet #1

```
clc
clear variables
close all
```

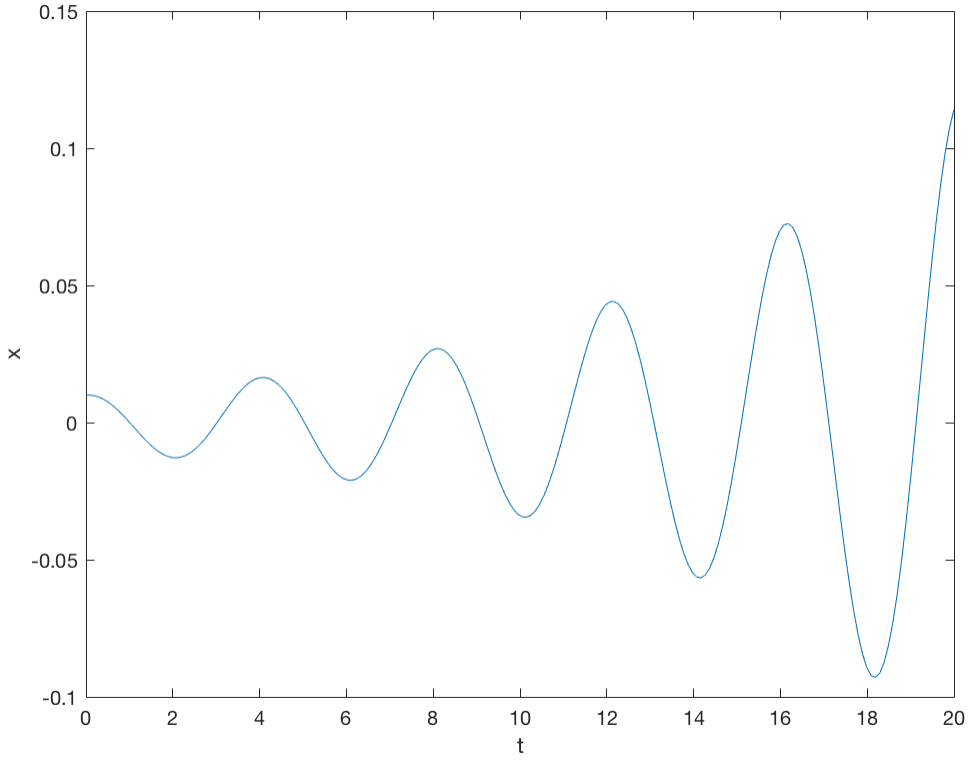
Q1

Find y(t)=[x(t),v(t)] with y(0)=[0.01;0] satisfying dy/dt=[y(2);-k/m*y(1)]

Q2

odeEuler is attached at the end of this file

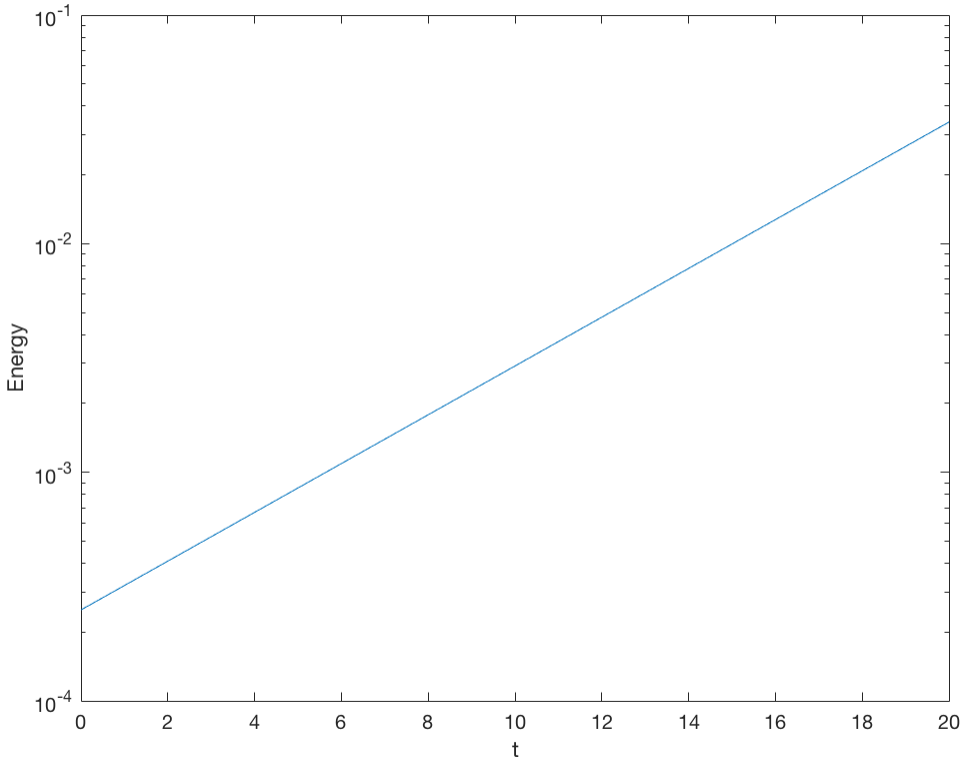
```
T=[0 20];
dt = 0.1;
k = 5;
m = 2;
x0 = 0.01;
v0 = 0;
y0 = [x0;v0];
odeSpring = @(t,y) [y(2);-k/m*y(1)];
[t,y] = odeEuler(odeSpring,T,y0,dt);
x = y(:,1);
v = y(:,2);
plot(t,x)
xlabel('t')
ylabel('x')
```



The amplitude increases, suggesting an increase in energy; this is not physical as no energy is inserted into the system.

Q3

```
E = 0.5*m*v.^2 + 0.5*k*x.^2;
semilogy(t,E)
xlabel('t')
ylabel('Energy')
set(legend,'Location','NorthWest')
```



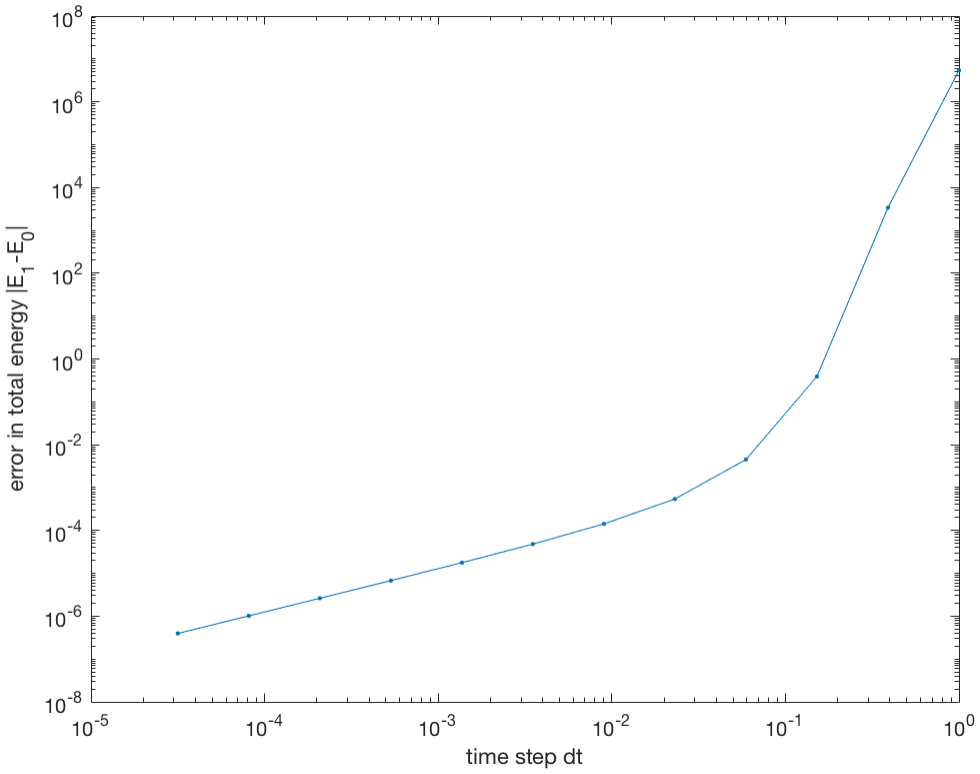
Total energy is increasing, which is unphysical.

Q4

E.g. energy conservation is a good criterium. The code does not pass this test. Another is that the amplitude should be limited, so you could look at max(x).

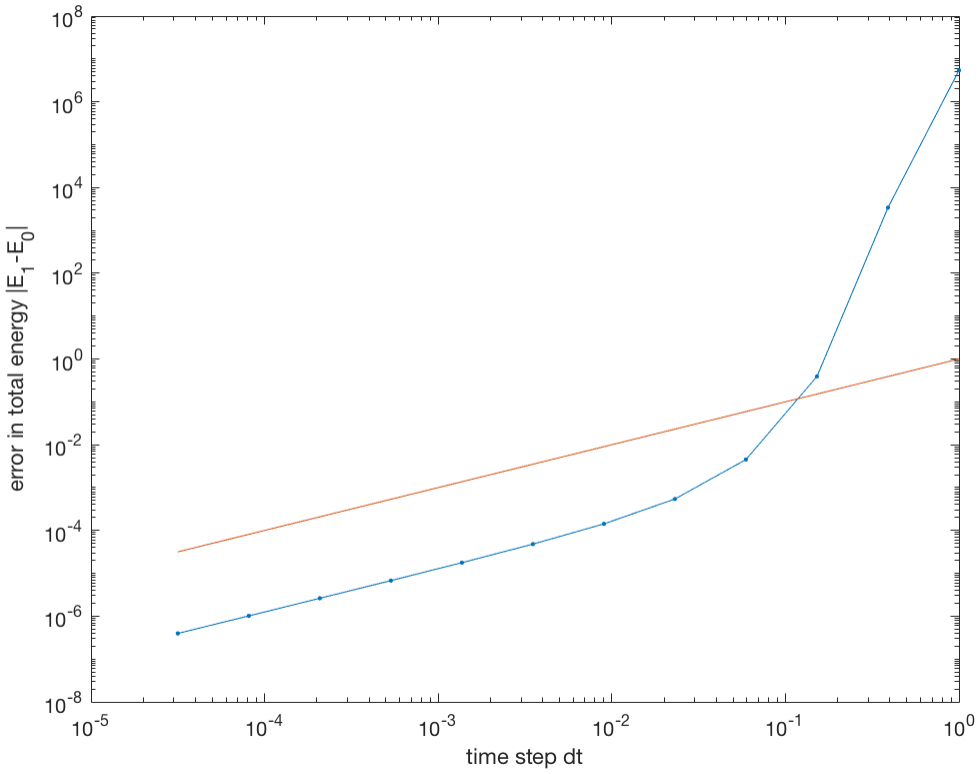
Q5

```
dt = logspace(-4.5,0,12);
E0 = 0.5*m*v0^2 + 0.5*k*x0^2;
for i = 1:length(dt)
    [t,y] = odeEuler(odeSpring,T,y0,dt(i));
    x1 = y(end,1);
    v1 = y(end,2);
    E1(i) = 0.5*m*v1^2 + 0.5*k*x1^2;
end
loglog(dt,abs(E1-E0),'.-')
xlabel('time step dt')
ylabel('error in total energy |E_1-E_0|')
```



The figure shows that the error decreases with the time step. Euler is first-order accurate, so we expect `E1-E0`=O(dt). Test this by plotting in log-log scale and comparing the slope of `E1-E0` and dt.

```
hold on
plot(dt,dt)
hold off
```

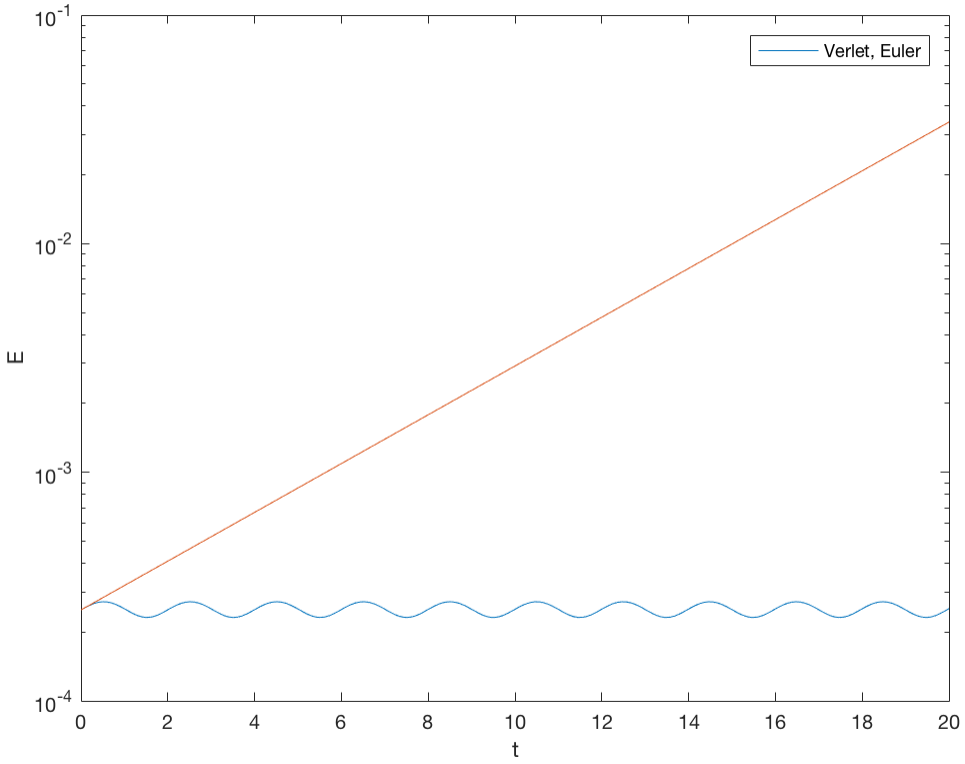


Q6

I implemented this in a separate function to keep the format similar to odeEuler, see below.

Q7

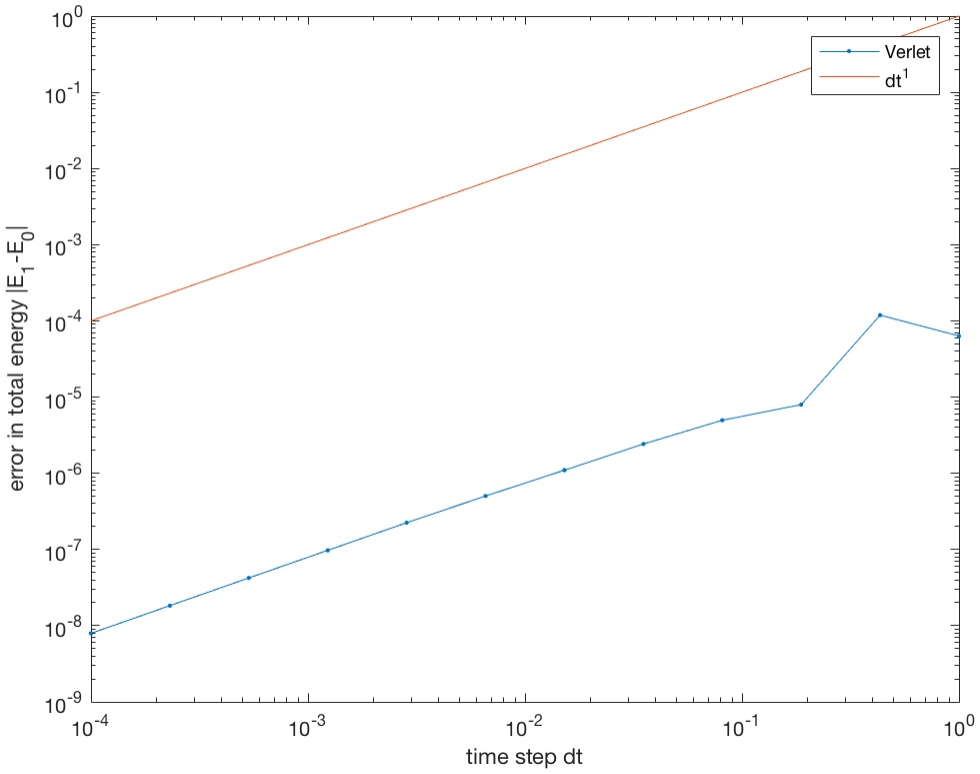
```
dt=0.1;
[tV,yV]=odeVerlet(T,y0,dt,k,m);
[tE,yE]=odeEuler(odeSpring,T,y0,dt);
xV = yV(:,1);
vV = yV(:,2);
xE = yE(:,1);
vE = yE(:,2);
EV = 0.5*m*vV.^2 + 0.5*k*xV.^2;
EE = 0.5*m*vE.^2 + 0.5*k*xE.^2;
semilogy(tV,EV,tE,EE)
xlabel('t')
ylabel('E')
legend('Verlet, Euler')
```



Verlet conserves energy better (N.B. the oscillation is actually due to a wrong interpretation of the velocity).

Q7b

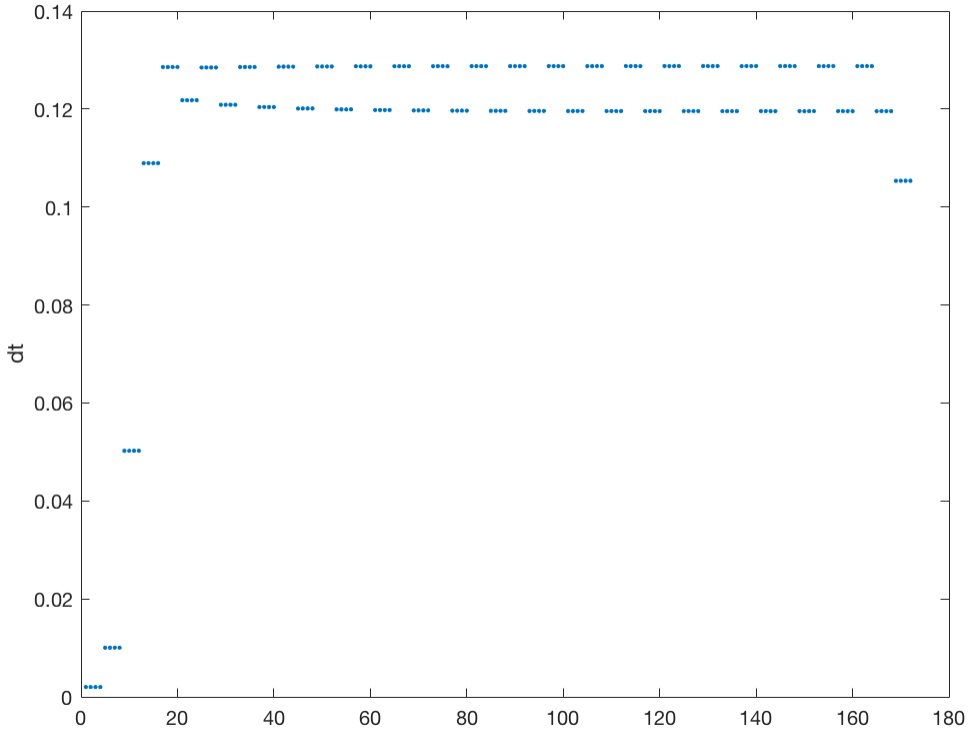
```
dt = logspace(-4,0,12);
E0 = 0.5*m*v0^2 + 0.5*k*x0^2;
for i = 1:length(dt)
    [t,y] = odeVerlet(T,y0,dt(i),k,m);
    x1 = y(end,1);
    v1 = y(end,2);
    E1(i) = 0.5*m*v1^2 + 0.5*k*x1^2;
end
loglog(dt,abs(E1-E0),'.-',dt,dt)
xlabel('time step dt')
ylabel('error in total energy |E_1-E_0|')
legend('Verlet','dt^1')
```



Verlet is also first order, like Euler

Q8

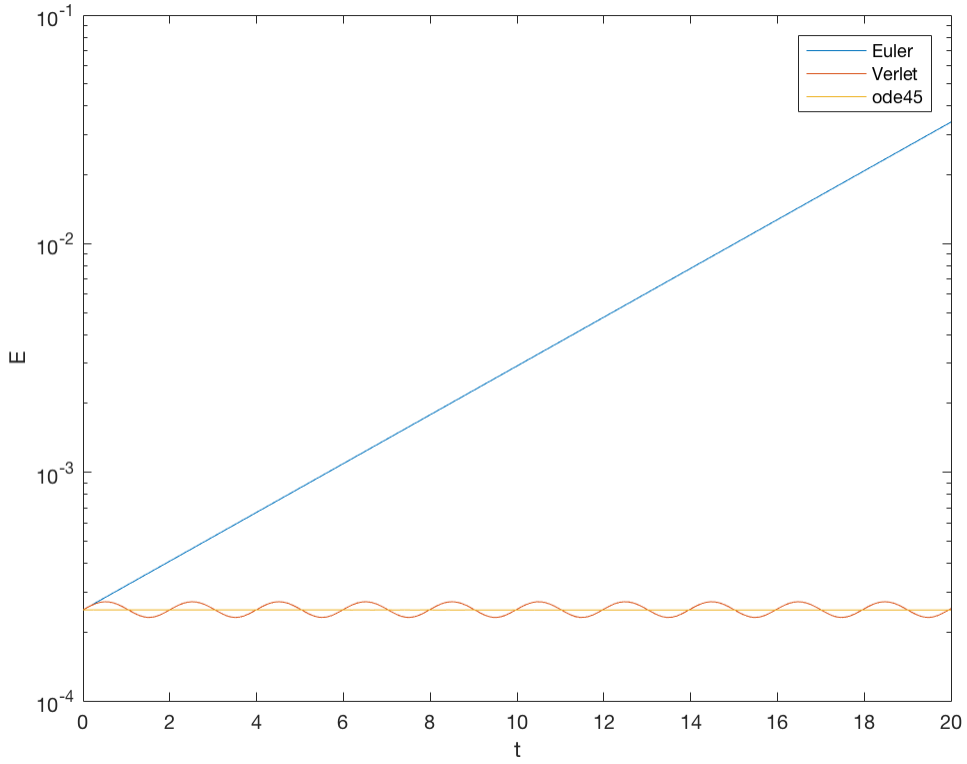
```
[t,y]=ode45(odeSpring,T,y0);
plot(diff(t),'.')
ylabel('dt')
```



The time step is not constant (you see four repeating values, because ode45 plots four values per timestep it takes).

Q9

```
x = y(:,1);
v = y(:,2);
E = 0.5*m*v.^2 + 0.5*k*x.^2;
semilogy(tE,EE,tV,EV,t,E)
legend('Euler','Verlet','ode45')
xlabel('t')
ylabel('E')
```



Energy is well conserved in ode45 ode45 is forth-order accurate (not shown) ode45 uses less time steps than Euler, Verlet, produces higher accuracy

odeEuler function

could be in a separate file

```
function [t,y]=odeEuler(odefun,T,y0,dt)
% Uses forward Euler to solve the IVP,
% dy/dt=f(t,y) for T(1)<t<T(2), y(t0)=y0.
% compute the number of time steps
n = ceil((T(2)-T(1))/dt);
% List of time values
t=linspace(T(1),T(2),n)';
% preallocate vector of function values
y=zeros(length(t),length(y0));
% Set initial value (colon notation to allow vectors)
y(1,:)=y0;
% now iterate over time
for i = 2:n
    y(i,:)=y(i-1,:)+dt*odefun(t(i-1),y(i-1,:))';
end
end
```

odeVerlet function

could be in a separate file

```
function [t,y]=odeVerlet(T,y0,dt,k,m)
% Uses forward Euler to solve the IVP,
% dy/dt=[y(2);-k/m*y(1)] for T(1)<t<T(2), y(t0)=y0.
% compute the number of time steps
n = ceil((T(2)-T(1))/dt);
% List of time values
t=linspace(T(1),T(2),n)';
% preallocate vector of function values
y=zeros(length(t),length(y0));
% Set initial value (colon notation to allow vectors)
y(1,:)=y0;
% now iterate over time
for i = 2:n
    y(i,1)=y(i-1,1)+dt*y(i-1,2);
    y(i,2)=y(i-1,2)-dt*k/m*y(i,1);
end
end
```

Published with MATLAB® R2016b