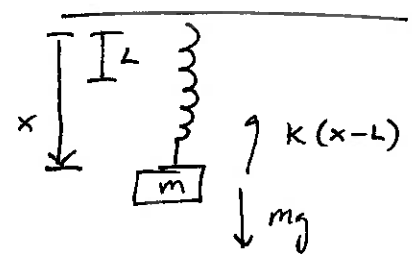
09-25 2nd order DEO Example

Bouncing Verlet

```python
#!/usr/bin/env python
# coding: utf-8


import math
import numpy as np
import matplotlib.pyplot as plt


# working with mg spring system  with following knowns
# DOWN is + x
# F_net = + mg - k (x-L)
m = 2 # kg
k = 100 # N/m
g = 9.8 # N/kg
L = 0.50 # meters relaxed spring length from mount point
x_eq = L + m*g / k # equilibrium stretch (F_net=0) from mount point
omega = (k/m)**0.5 # natural oscillation frequency

# solving this set of equations
# m dv_dt = + m*g - k*(x-L)
# dx_dt = v

period = 2.0*math.pi/omega
time_limit = 2.5*period
N=100
dt = period/N

# the VERLET integrator needs two initial state points
t1 = 0
x1 = L + 1.1*x_eq
v1 = 0.0
a1 = (m*g - k*(x1-L))/m
# using kinematics to get the second starting point - source of error!
x2 = x1 + v1*dt + 0.5*a1*dt*dt
v2 = v1 + a1*dt
t2 = t1 + dt

print("period (s) = %f"%period)
print("L + mg/k = %.3f + %.3f = %.3f (meters)"%(L,m*g/k,x_eq))
print("(x1, x2) = (%.3f, %.3f)"%(x1,x2))
print("dt = %.2e (seconds)"%dt)

# storage arrays
v_2s=[]
x_2s=[]
t_2s=[]
KE_2s=[]
Ug_2s=[]
Uspring_2s=[]
Etot_2s=[]

# repeat DiffEq solver procedure many many times to get an approximate model for the motion
while (t2<time_limit) :

    F2 = m*g - k*(x2-L)
    a2 = F2/m
    # apprimately solve differential equation over a very short time interval
    # using the "Velocity Verlet" DEQ integrator
    # x3 = new position, i+1
    # x2 = current position, i
    # x1 = old position, i-1
    x3 = 2*x2 - x1 + a2*dt*dt
    v2 = (x3-x1)/(2*dt)
```

Velocity Verlet Integrator



```
    t2 = t2 + dt

    # store the current dynamic values
    v_2s.append(v2)
    x_2s.append(x2)
    t_2s.append(t2)

    KE = 0.5*m*v2**2
    Ug = -m*g*x2
    Uspring = 0.5*k*(x2-L)**2
    Etot = KE+Ug+Uspring

    KE_2s.append(KE)
    Ug_2s.append(Ug)
    Uspring_2s.append(Uspring)
    Etot_2s.append(Etot)

    # update/recycle values for the next loop
    x1 = x2
    x2 = x3

plt.plot(t_2s,x_2s,label="verlet", linestyle='--', marker='o', color='b')
plt.grid()
plt.legend()
plt.title("spring-mass system, Velocity-Verlet, dt = %.2e(sec)"%dt)
plt.ylabel("position (m) down is +, so lower numbers are less stretch")
plt.xlabel("time (seconds)")

plt.plot(t_2s,KE_2s,label="KE")
plt.plot(t_2s,Ug_2s,label="U_gravity")
plt.plot(t_2s,Uspring_2s,label="U_spring")
plt.plot(t_2s,Etot_2s,label="E_total")
plt.grid()
plt.legend()
plt.title("spring-mass system, Velocity-Verlet, dt = %.2e(sec)"%dt)
plt.ylabel("Energy (J)")
plt.xlabel("time (seconds)")

plt.plot(x_2s,v_2s,label="verlet")
plt.grid()
plt.legend()
plt.title("Phase plot for oscillator, using Velocity Verlet")
plt.xlabel("position (m)")
plt.ylabel("velocity (m/s)")

# compute and plot change in energy
print(Etot_2s[0],Etot_2s[len(Etot_2s)-1])

fractional_gain = []
for E in Etot_2s:
    fractional_gain.append((E-Etot_2s[0])/Etot_2s[0])
plt.plot(t_2s,fractional_gain)
plt.xlabel("time (s)")
plt.ylabel("($E_{total} - E_{total 0})/E_{total 0}$")
```

09-25 2nd Order DEQ Example

Bouncing solve-ivp

$$\frac{dv}{dt} = mg - k(x-L)$$

$$\frac{dx}{dt} = v$$

```python
#!/usr/bin/env python
# coding: utf-8


# working with mg spring system  with following knowns
# down is +
# F_net = +mg - k (x-x0)
m = 2 # kg
k = 100 # N/m
g = 9.8 # N/kg
L = 0.50 # meters relaxed spring length from mount point
x_eq = m*g / k # equilibrium stretch from mount point/relaxed length
omega = (k/m)**0.5 # natural oscillation frequency

import math
import numpy as np
from scipy.integrate import solve_ivp

# for solve_ivp, this needs to be of the form:
# name(time, variables, args)
def dstate_dt(t, state, k, m, g, L):
    x, v = state

    dx_dt = v
    dv_dt = (m*g - k*(x-L))/m # reminder, a = F/m

    return [dx_dt, dv_dt]

period = 2.0*math.pi/omega
tspan = [0,2*period]
x0 = L + 1.1*x_eq
v0 = 0.0

solution = solve_ivp(
    dstate_dt, # derivative as function
    tspan,  # time interval to solve for
    [1.1*x_eq, 0], # initial values
    args=(k,m,g,L,) # why does this have to have a comma?  needs to be a "tuple"????
    , method="RK45"
)

# this was helpful for syntax
# https://simulationbased.com/2021/02/16/differential-equations-with-scipy-odeint-or-solve_ivp/comment-page-1/

print(solution)

print(solution.t)
print(solution.y[0])
print(solution.y[1])

import matplotlib.pyplot as plt
plt.plot(solution.t,solution.y[0],label="x solve_ivp", linestyle='--', marker='o', color='b')
tvals=np.arange(0,period,period/30)
plt.plot(tvals,L+(x0-L)*np.cos(omega*tvals), label="L+A*sin(omega t)")
plt.grid()
plt.legend()
plt.title("using solve_ivp (RK45)")
plt.ylabel("position")
plt.xlabel("time (seconds)")
```

09-25 2nd order DEQ Example

decays

```python
#!/usr/bin/env python
# coding: utf-8

# In[51]:


# working with radioactive decay CHAIN with following knowns
# Na -> Nb -> stable
# N(t) = N0 Exp[-t/tau]
# dN/dt = -(1/tau) N(t)
# tau = t_half / ln(2)

import math
import numpy as np
from scipy.integrate import solve_ivp

# for solve_ivp, this needs to be of the form:
# name(time, variables, args)
def dN_dt(t, state, tau_a, tau_b):
    Na, Nb, Nstable = state

    dNa = -Na/tau_a
    dNb = Na/tau_a + -Nb/tau_b
    dNstable = Nb/tau_b

    return [dNa, dNb, dNstable]

ta_half = 10 # seconds
tau_a = ta_half/math.log(2.0)
tb_half = 15 # seconds
tau_b = tb_half/math.log(2.0)
N0a = 100 # number of intial atoms
N0b = 0 # number of intial atoms
N0stable = 0 # number of stable atoms at the end of the decay chain

tspan = [0,5*ta_half]
solution = solve_ivp(
    dN_dt, # derivative as function
    tspan,  # time interval to solve for
    [N0a, N0b, N0stable], # initial values
    args=(tau_a,tau_b,) # why does this have to have a comma?  needs to be a "tuple"????
    , method="RK45"
)

# this was helpful for syntax
# https://simulationbased.com/2021/02/16/differential-equations-with-scipy-odeint-or-solve_ivp/comment-page-1/


# In[53]:


print(solution)


# In[55]:


print(solution.t)
print(solution.y[0])
print(solution.y[1])
print(solution.y[2])
```

```
# In[61]:


import matplotlib.pyplot as plt
plt.plot(solution.t,solution.y[0],label="Na solve_ivp t_half=%.1f"%ta_half, linestyle='--', marker='o', color='b')
plt.plot(solution.t,solution.y[1],label="Nb solve_ivp t_half=%.1f"%tb_half, linestyle='--', marker='o', color='r')
plt.plot(solution.t,solution.y[2],label="Nstable solve_ivp ", linestyle='--', marker='o', color='violet')
tvals=np.arange(0,3*ta_half)
plt.plot(tvals,N0a*np.exp(-tvals/tau_a), label="N0a*Exp(-t/tau)")
plt.grid()
plt.legend()
plt.title("solve_ivp using solve_ivp (RK45)")
plt.ylabel("number of atoms")
plt.xlabel("time (seconds), ta_1/2 = %.2f"%ta_half)


# In[ ]:
```