Homework #5

0. K4 was incorrect.
Is there a word for copying and pasting something simliar to what you want in order to save typing and then neglecting to make needed changes?

```python
def rk4(f,t0,y0,h,m):

    b21 = 1/2
    b31 = 0
    b32 = 1/2
    b41 = 0
    b42 = 0
    b43 = 1
    b51 = 1/6
    b52 = 1/3
    b53 = 1/3
    b54 = 1/6

    t = t0
    y = np.array(y0)

    ya = np.empty((len(y0),m+1))
    ya[:,0] = y
    ta = np.linspace(t0,t0+m*h,m+1)

    for k in range(m):
        t = t0 + k*h
        K1 = f(t,y)
        y1 = y + h*b21*K1
        K2 = f(t+h*b21,y1)
        y2 = y + h*(b31*K1 + b32*K2)
        K3 = f(t+h*(b31    + b32   ), y2)
        y3 = y + h*(b41*K1 + b42*K2 + b43*K3)
        #K4 = f(t+h*(b31    + b32    + b43   ), y3)  wrong
        K4 = f(t+h*(b41    + b42    + b43   ), y3)  # corrected
        y4 = y + h*(b51*K1 + b52*K2 + b53*K3 + b54*K4)
        y = y4
        ya[:,k+1] = y

    return ta,ya
```

numpy

Homework #5

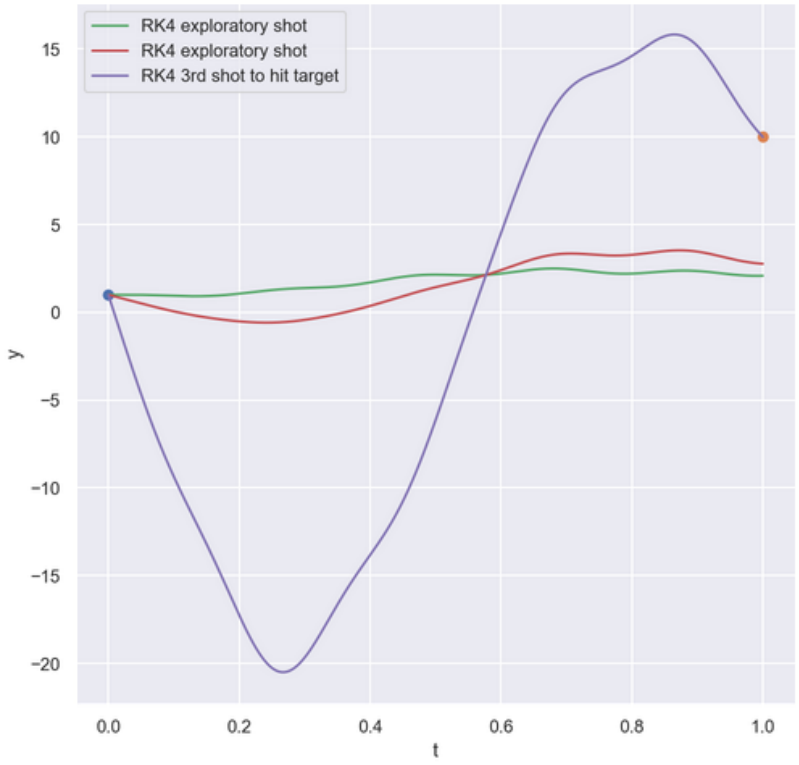
1. A second-order linear scalar BVP solved by shooting.

```
# general scalar linear 2nd order

def f(x,Y):
    global p,q,phi
    y,yp = Y
    return np.array([ yp, phi(x) -p(x)*yp - q(x)*y ])

# wild

def p(x):   return 5*x**2
def q(x):   return 30+50*np.sin(30*x)
def phi(x): return 50*np.cosh(x)
x0,x1 = 0,1
alpha = 1
beta = 10
```

```
plt.figure(figsize=(8,8))
plt.plot(0,alpha,'o')
plt.plot(1, beta,'o')
sigmas = [0,-10]
endvals = []
m = 640   # chosen on basis of convergence check shown below
h = (x1-x0)/m
for sigma in sigmas:
    ta,Ya = rk4(f,x0,[alpha,sigma],h,m)
    plt.plot(ta,Ya[0,:],label='RK4 exploratory shot')
    endvals.append(Ya[0,-1])
sigma = sigmas[0] + (beta - endvals[0])/(endvals[1] - endvals[0])*(sigmas[1]-sigmas[0])
sigma
ta,Ya = rk4(f,x0,[alpha,sigma],h,m)
plt.plot(ta,Ya[0,:],label='RK4 3rd shot to hit target')
plt.legend(); plt.xlabel('t'); plt.ylabel('y')
print('Hit target?',Ya[0,-1],'should be',beta)
print('Value of solution at t=0.2?',Ya[0,ta==0.2])
```

```
Hit target? 10.000000000000009 should be 10
Value of solution at t=0.2? [-17.25819506]
```

Hit is perfect.

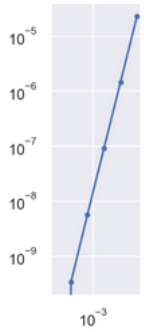


1. continued

Convergence check

```
sigmas = [0,-10]
midway_vals= []
mvals = [10*2**i for i in range(4,10)]
hvals = [(x1-x0)/m for m in mvals]
for m in mvals:
    endvals = []
    h = (x1-x0)/m
    for sigma in sigmas:
        ta,Ya = rk4(f,x0,[alpha,sigma],h,m)
        endvals.append(Ya[0,-1])
    sigma = sigmas[0] + (beta - endvals[0])/(endvals[1] - endvals[0])*(sigmas[1]-sigmas[0])
    ta,Ya = rk4(f,x0,[alpha,sigma],h,m)
    midway_vals.append(Ya[0,ta==0.2][0])
    print('m = ',m,' sigma =',sigma,' Hit target?',Ya[0,-1],'should be',beta,' Value of solution at t=0.2?',Ya[0,ta==0.2])

hvals,np.abs(np.array(midway_vals)-midway_vals[-1])
%matplotlib inline
plt.figure('convergence?')
plt.subplot(111,aspect=1)
plt.loglog(hvals,np.abs(np.array(midway_vals)-midway_vals[-1]),'.-');
```

```
m =  160  sigma = -116.03579275900853  Hit target? 9.999999999999993 should be 10  Value of solution at t=0.2? [-17.25817207]
m =  320  sigma = -116.03594200993655  Hit target? 10.000000000000018 should be 10  Value of solution at t=0.2? [-17.25819369]
m =  640  sigma = -116.03595147936198  Hit target? 10.000000000000009 should be 10  Value of solution at t=0.2? [-17.25819506]
m =  1280  sigma = -116.03595207490271  Hit target? 9.999999999999993 should be 10  Value of solution at t=0.2? [-17.25819515]
m =  2560  sigma = -116.0359521122293  Hit target? 10.000000000000068 should be 10  Value of solution at t=0.2? [-17.25819515]
m =  5120  sigma = -116.03595211456486  Hit target? 10.000000000000055 should be 10  Value of solution at t=0.2? [-17.25819515]
```



Convergence looks linear not O(h^4)! Did I code rk4 incorrectly? YES! Now fixed and converges as O(h^4).

2. Spreading and compression of nearby trajectories in Lorenz model

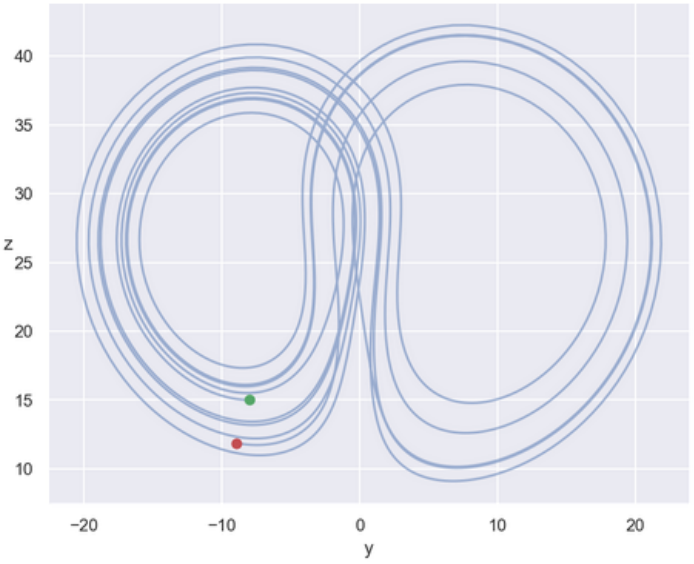
a.

Variational equations for Lorenz

```
from rk4 import rk4

def F(t,YV):
    global ODEF, ODEDF
    n = int((-1 + np.sqrt(1 + 4*len(YV)) )/2)
    #print(n)
    Y = YV[:n]
    V = YV[n:].reshape((n,n))
    #print(V)
    FY = ODEF(t,Y)
    FV = ODEDF(t,Y)@V
    return np.append( FY,FV.reshape(n*n) )

def f(t,Y):
    x,y,z = Y
    return np.array([ 10*y - 10*x,
                      28*x - y - x*z,
                     -8*z/3 + x*y ])

def Df(t,Y):
    x,y,z = Y
    return np.array([[-10, 10, 0],
                     [28-z, -1, -x],
                     [y, x, -8/3]])

ODEF = f
ODEDF = Df

Y = [-5, -8, 15]
YV = np.append(Y,np.eye(3).reshape(9))
F(0,YV)
t1 = 10.8
m = 2000
h = t1/m
ta,yva = rk4(F,0,YV,h,m)
yva
V1 = yva[3:,-1].reshape(3,3)
#display(V1)
display('Singular values at t =' + str(t1) + ': ' + str( np.linalg.svd(V1)[1]) )
plt.figure(figsize=(8,6))
plt.subplot(111,aspect=1)
plt.plot(yva[1,:],yva[2,:],'-',alpha=0.5)
plt.plot(yva[1, 0],yva[2, 0],'go')
plt.plot(yva[1,-1],yva[2,-1],'ro')
plt.xlabel('y')
plt.ylabel('z',rotation='horizontal');
```

I doubled the number of steps until converged to 7 or 8 digits.

```
'Singular values at t =10.8: [9.37006953e+04 4.78774395e-01 4.22395309e-12]'
```



b. One s.v. is huge - about 10^5, indicating very strong separation of nearby trajectories. (Butterfly effect - tiny changes in initial conditions get magnified to large differences at later times.

Another is tiny - practically 0, indicating balls of trajectories are squashed almost flat.

The other is about .5 indicating that there is a direction (approximately along the trajectory) along which trajectories do not separate or compress much at all.