Homework #4 Implicit trapezoid method for a stiff system.

I made methods for an Euler step and a Trapezoid step that assume there's a function that returns f and its jacobian Df at a supplied (t,Y).

The Trapezoid code is general except that the number Newton iterations is hard-coded, currently at 1. I used the current Y value as the initial guess at the future value - other choices are possible.

```
def euler_step(t,yj,fDf,h):
    y = yj
    fj = fDf(t,yj)[0]
    return y + h*fj

def trapezoid_step(t,yj,fDf,h):
    y = yj  # use current y value as initial guess for future value
    fj = fDf(t,yj)[0]
    newtonsteps = 1
    for k in range(newtonsteps):
        f,Df = fDf(t,y)
        g = y - yj - h/2*(fDf(t,y)[0] + fj)
        Dg = np.eye(len(y)) - h/2*Df
        s = -np.linalg.solve(Dg,g)
        y = y + s
    return y
```

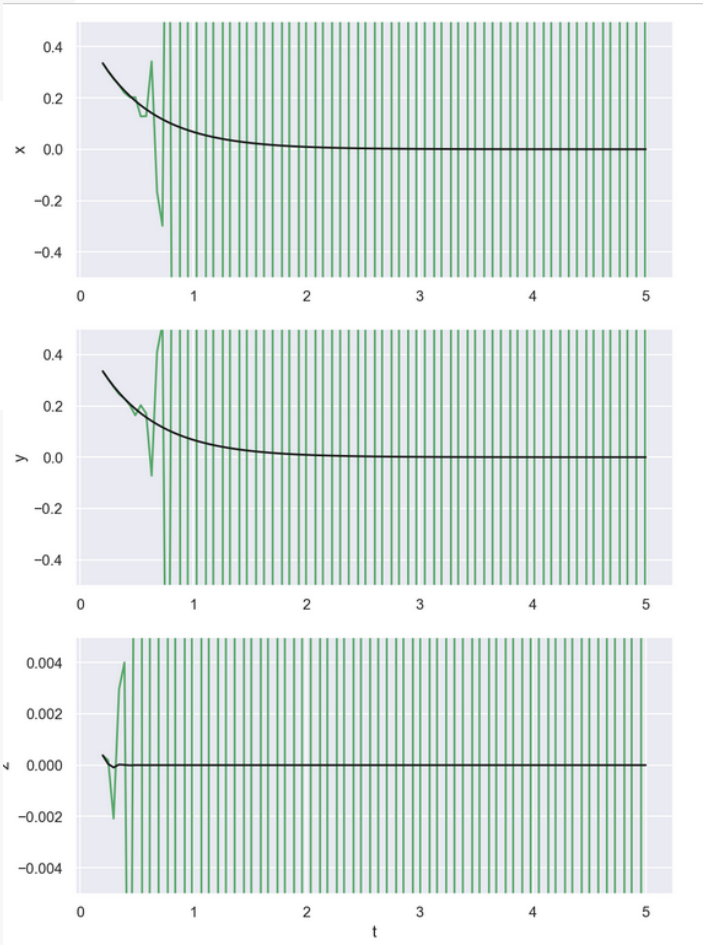
Results:
Euler (green) blows up.
Trapezoid goes to zero as expected, though z(t) is not very accurate in a relative sense:

Original problem from Ackleh text:

```
import numpy as np
A = np.array([[-21, 19,-20],
              [ 19,-21, 20],
              [ 40,-40,-40]],dtype=float)
def fDf(t,Y):
    global A
    return A@Y,A
```

```
t0 = 0.2
t1 = 5
M = 100
h = (t1-t0)/M
ya = np.empty((3,M+1))

fig,ax = plt.subplots(3,1,figsize=(8,12))
for method,color in zip([euler_step,trapezoid_step],'gk'):
    y = [0.33530156, 0.33501848, 0.0003807 ] # ic(t0)
    ya[:,0] = y
    for j in range(M):
        t = t0 + h*j
        y = method(t,y,fDf,h)
        ya[:,j+1] = y
        #break
    ta = np.linspace(t0,t1,M+1)
    for i in range(3):
        ax[i].plot(ta,ya[i,:],'-',color=color)
ax[0].set_ylim(-0.5,.5)   ; ax[0].set_ylabel('x')
ax[1].set_ylim(-0.5,0.5)  ; ax[1].set_ylabel('y')
ax[2].set_ylim(-.005,.005); ax[2].set_ylabel('z'); ax[2].set_xlabel('t');
```



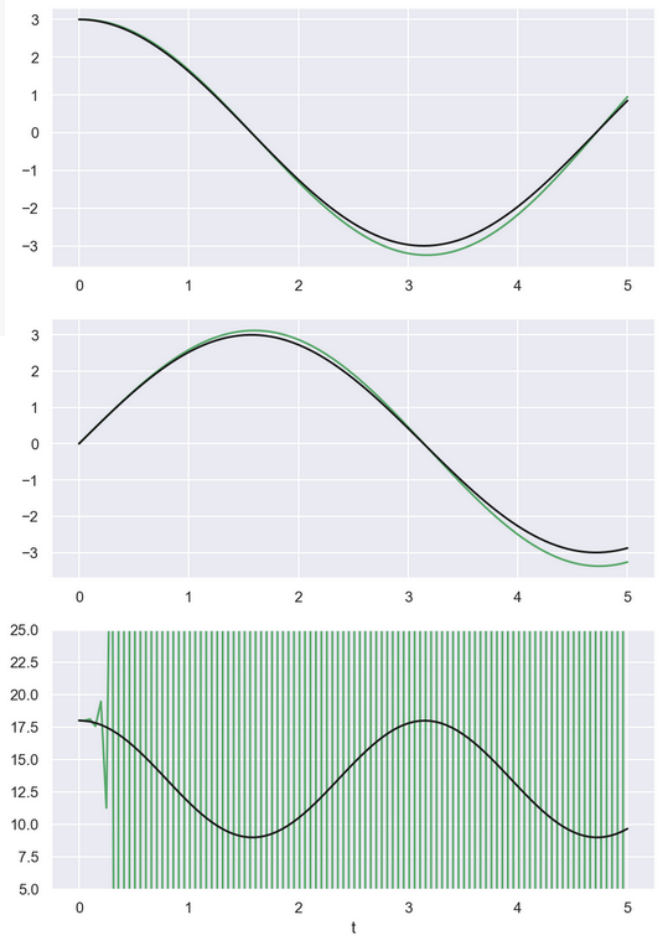
Alternate problem: a nonlinear stiff system

```
def fDf(t,Y):
    x,y,z = Y
    a = 100.
    f = np.array([-y,x, -a*(z-2*x**2-y**2) ])
    Df = np.array([[0,-1,0],
                   [1,0,0],
                   [2*2*a*x,2*a*y,-a]])
    return f,Df


def ic(t):
    return np.array([3,0,18.])
```

```
t0 = 0.0;
t1 = 5
M = 100
h = (t1-t0)/M
print(h)
ya = np.empty((3,M+1))

fig,ax = plt.subplots(3,1,figsize=(8,12))
for method,color in zip([euler_step,trapezoid_step],'gk'):
    y = ic(t0)
    ya[:,0] = y
    for j in range(M):
        t = t0 + h*j
        y = method(t,y,fDf,h)
        ya[:,j+1] = y
        #break
    ta = np.linspace(t0,t1,M+1)
    for i in range(3):
        ax[i].plot(ta,ya[i,:],'-',color=color)
ax[2].set_ylim(5,25); ax[2].set_xlabel('t');
```



Results: Trapezoid method generates a good approximation to the oscillatory solution, while Euler blows up.

Using a method whose A.S. region includes the entire negative real axis is effective here!