538 HW4 S22

1 (a) 2-step AB is $y_{j+2} = y_{j+1} + h\left(\frac{3}{2}f(y_{j+1}) - \frac{1}{2}f(y_j)\right)$ (for autonomous)

$$= y_{j+1} + h\lambda\left(\frac{3}{2}y_{j+1} - \frac{1}{2}y_j\right)$$

for $f(y) = \lambda y$.

(b) $$\begin{bmatrix} y_{j+1} \\ y_{j+2} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ -\frac{1}{2}h\lambda & 1+\frac{3}{2}h\lambda \end{bmatrix}}_{M} \begin{bmatrix} y_j \\ y_{j+1} \end{bmatrix}$$

$$\det(M-\lambda z) = \det\begin{bmatrix} -z & 1 \\ -\frac{1}{2}h\lambda & 1+\frac{3}{2}h\lambda - z \end{bmatrix} = \underbrace{(-z)\left(1+\frac{3}{2}h\lambda - z\right) + \frac{1}{2}h\lambda}_{\text{char poly}}$$

$$\boxed{\begin{gathered} z^2 - \left(1+\frac{3}{2}h\lambda\right)z + \frac{1}{2}h\lambda = 0 \\ z = \frac{\left(1+\frac{3}{2}h\lambda\right) \pm \sqrt{\left(1+\frac{3}{2}h\lambda\right)^2 - 2h\lambda}}{2} \end{gathered}}$$

(c) $$= \frac{\left(1+\frac{3}{2}h\lambda\right) \pm \sqrt{1 + \underbrace{3h\lambda - 2h\lambda}_{h\lambda} + \frac{9}{4}(h\lambda)^2}}{2}$$

$$= \frac{1}{2}\left[\left(1+\frac{3}{2}h\lambda\right) \pm \underbrace{\left\{1 + \frac{1}{2}\left(h\lambda + \frac{9}{4}(h\lambda)^2\right) - \frac{1}{8}\left(h\lambda + \frac{9}{4}(h\lambda)^2\right)^2 + O(h^3)\right\}}_{\text{Taylor expanded } \sqrt{1+\varepsilon}}\right]$$

$$= \begin{cases} \frac{1}{2}\left[1 + \frac{3}{2}h\lambda + 1 + \frac{1}{2}h\lambda + \frac{9}{8}(h\lambda)^2 - \frac{1}{8}(h\lambda)^2 + O(h^3)\right] \\ \frac{1}{2}\left[1 + \frac{3}{2}h\lambda - 1 - \frac{1}{2}h\lambda + O((h\lambda)^2)\right] \end{cases}$$

$$= \begin{cases} 1 + h\lambda + \frac{1}{2}(h\lambda)^2 + O(h^3) = e^{h\lambda} + O(h^3) \\ \frac{h\lambda}{2} + O((h\lambda)^2) \xrightarrow[(h\lambda)\to 0]{} 0 \end{cases}$$

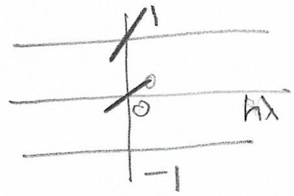
QED.

One eigenvalue approximates e^{h lambda}, the other is close to zero.

1(d) Char poly is $z^2 - \left(1 + \frac{3}{2}h\lambda\right)z + \frac{1}{2}h\lambda = 0$.

From part (c), we saw for small $h\lambda$

$$z_\pm \approx \begin{cases} 1 + h\lambda \\ \frac{h\lambda}{2} \end{cases}$$:



Want to know where $|z_+|$ or $|z_-|$ first attains value 1 as $h\lambda$ is decreased from 0.

If $z_\pm$ stay real, then must attain $|z|=1$ at $z = \pm 1$..

We can check for these easily:

$z = +1$: $1^2 - 1 - \frac{3}{2}h\lambda + \frac{1}{2}h\lambda = h\lambda = 0 \rightarrow h\lambda = 0$ only.

$z = -1$: $(-1)^2 + 1 + \frac{3}{2}h\lambda + \frac{1}{2}h\lambda = 2 + 2h\lambda = 0$

$\rightarrow \boxed{h\lambda = -1 \text{ only}}$

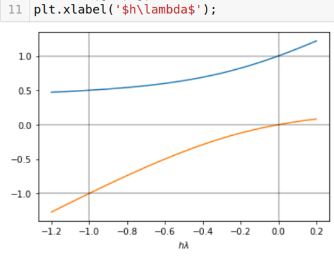
and can conclude stability lost at $\boxed{h\lambda = -1}$.

To guarantee $z_\pm$ do stay real, could show that discriminant does not change sign on $h\lambda \in (-1, 0)$.

I'll make a quick plot for sanity check.

```python
import numpy as np
import matplotlib.pyplot as plt
def axhlines(vals):
    for val in vals: plt.axhline(val,color='k',alpha=.3)
def axvlines(vals):
    for val in vals: plt.axvline(val,color='k',alpha=.3)

```

```python
hlambda = np.linspace(-1.2,0.2,40)
a = 1
b = -(1+3/2*hlambda)
c = 1/2*hlambda
zp = (-b + np.sqrt(b**2 - 4*a*c))/2/a
zm = (-b - np.sqrt(b**2 - 4*a*c))/2/a
plt.plot( hlambda,zp )
plt.plot( hlambda,zm )
axhlines([-1,0,1])
axvlines([-1,0])
plt.xlabel('$h\lambda$');
```



Ack 7.10.16

$$y_{j+1} = y_j + h\,\Phi(t_j, y_j, h)$$

$$\Phi(t_j, y_j, h) = \tfrac{1}{4} f(t_j, y_j) + \alpha f\left(t_j + \tfrac{h}{4}, y_j + \beta h f(t_j, y_j)\right)$$

(a) Consistent iff $\Phi(t_j, y_j, 0) = f(t_j, y_j)$

Here $\Phi(t_j, y_j, 0) = \tfrac{1}{4} f(t_j, y_j) + \alpha f(t_j\, y_j)$

$$= \left(\tfrac{1}{4} + \alpha\right) f(t_j, y_j)$$

Thus for consistency, $\boxed{\alpha = \tfrac{3}{4}}$.

(b)

$$y_{j+1} = y_j + h \cdot \left(\tfrac{1}{4} f + \tfrac{3}{4}\left[f + f_t \cdot \tfrac{h}{4} + f_y\, \beta h f + O(h^2) \right] \right)$$

$$= y_j + hf + h^2\left(\frac{3 f_t}{16} + \frac{3\beta}{4} f_y \cdot f \right) + O(h^3)$$

while

$$y(t_j + h) = y_j + hy' + \frac{h^2}{2} y'' + O(h^3)$$

$$= y_j + hf + \frac{h^2}{2}\left(f_t + f_y \cdot f \right) + O(h^3)$$

(i) The method is order $p = 1$ regardless of the value of β.

(ii) No value of β can make the method of order $p = 2$ (for general f) because $\frac{3}{16} \neq \frac{1}{2}$ regardless of β.

(For an <u>autonomus</u> system, $f_t \equiv 0$, and the method is of order 2 if $\frac{3}{4}\beta = \frac{1}{2}$, ie. $\beta = \frac{2}{3}$.)

HW4

3 Examples among many others

(a) Consistent and A-stable :

backward Euler $\quad y_{j+1} = y_j + hf(t_{j+1}, y_{j+1})$.

trapezoid

(b) consistent & stable, but not A-stable

Euler $\quad y_{j+1} = y_j + hf(t_j, y_j)$

and all Adams-Bashforth methods

(c) consistent but not stable

"full" 2-step $\quad y_{j+2} + 4y_{j+1} - 5y_j = h(4f_{j+1} + 2f_j)$

(7.85) in textbook

(d) stable but not consistent

$$y_{j+1} = \frac{1}{10} y_j$$

$$y_{j+1} = y_j + 2hf_j$$

(e) neither consistent nor stable

$$y_{j+1} = 10 y_j$$

4. A nonlinear stiff system causes trouble for Euler, not for Trapezoid.

imports

```
from matplotlib import rcdefaults
rcdefaults()  # restore default matplotlib rc parameters
%config InlineBackend.figure_format='retina'
import seaborn as sns  # wrapper for matplotlib that provides prettier styles and more
import matplotlib.pyplot as plt  # use matplotlib functionality directly
%matplotlib inline
sns.set()

import numpy as np
```

functions to execute Euler and trapezoid steps

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

def trapezoid_step(t,yj,fDf,h):
    y = np.array(yj)  # use current y value as initial guess for future value
    fj = fDf(t,yj)[0] # function of diff eq
    newtonsteps = 1
    for k in range(newtonsteps):
        f,Df = fDf(t,y) # function and jacobian of diff eq
        g = y - yj - h/2*(fDf(t,y)[0] + fj)  # value of function g that's supposed to be zero
        Dg = np.eye(len(y)) - h/2*Df         # jacobian of g
        s = -np.linalg.solve(Dg,g)  # Newton step
        y = y + s
    return y
```

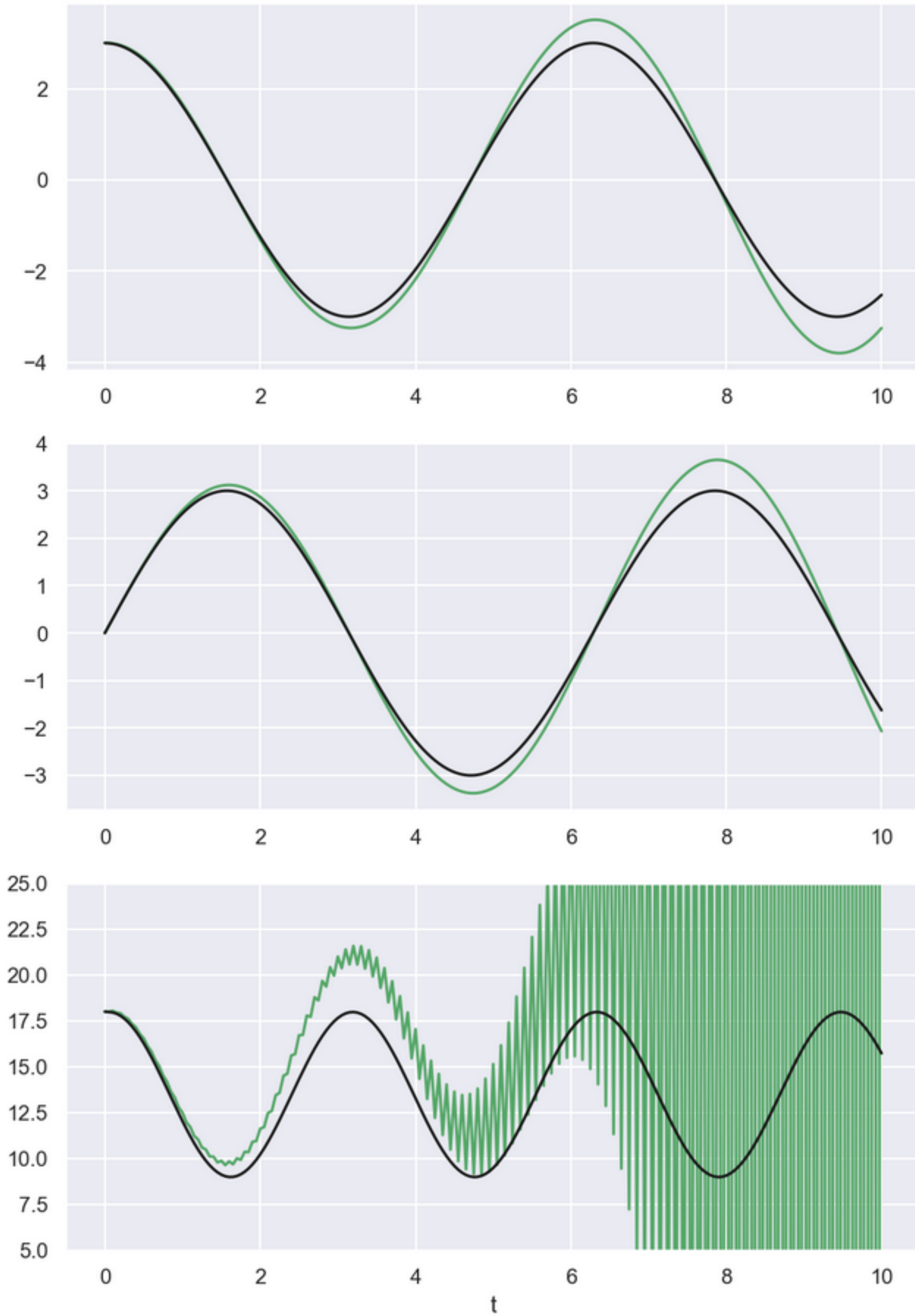
just taking 1 Newton step per timestep

A nonlinear stiff system

```
# another stiff example
import numpy as np

def fDf(t,Y):
    x,y,z = Y
    a = 41.
    b = 2
    f = np.array([-y,x, -a*(z-b*x**2-y**2) ])
    Df = np.array([[0,-1,0],
                   [1,0,0],
                   [2*a*b*x,2*a*b*y,-a]])
    return f,Df

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

t0 = 0.0;
t1 = 10
M = 200
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');
```

a function to compute both f and Df

Results: Euler in green, trapezoid in black.



Trapezoid looks good. Euler is unstable.