538 Spring 2022 Homework 3 solutions

(a)

Taylor series of solutions of Lorenz system

First I build formulas for the Taylor expansions of the solution components.

For 4th order, this takes a couple of seconds to execute.

```
import sympy as sp
sp.init_printing()
t = sp.symbols('t')
h = sp.symbols('h')
x = sp.Function('x')
y = sp.Function('y')
z = sp.Function('z')

de = {x(t).diff(t): 10*y(t) - 10*x(t),
      y(t).diff(t): 28*x(t) - y(t) - x(t)*z(t),
      z(t).diff(t): -8*z(t)/5 + x(t)*y(t) }       # changed from 8/3 to 8/5 for s22

xders = [x(t),de[x(t).diff(t)]]
yders = [y(t),de[y(t).diff(t)]]
zders = [z(t),de[z(t).diff(t)]]

for i in range(4):
    for item in [xders, yders, zders]:
        item.append(item[-1].diff(t))
        item[-1] = item[-1].subs( de )

tx = sp.simplify(sum([ xders[i]*h**i/sp.factorial(i) for i,xder in enumerate(xders) ] ))
ty = sp.simplify(sum([ yders[i]*h**i/sp.factorial(i) for i,yder in enumerate(yders) ] ))
tz = sp.simplify(sum([ zders[i]*h**i/sp.factorial(i) for i,zder in enumerate(zders) ] ))
nextY = sp.lambdify( (x(t),y(t),z(t),t,h), (tx,ty,tz), "numpy" )

# numerical version of the DE
def f(t,Y):
    x,y,z = Y
    return np.array([ 10*y - 10*x, 28*x -y -x*z, -8*z/5 + x*y ])
```

Sanity check of the function I made to deliver the result of taking one step of size h:

```
nextY(-5.        ,  -8.        ,  15.       ,0,.02)
```

$(-5.653747374453333, \ -9.189686442674136, \ 15.42831540097754)$

plotting and numpy 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
```

function to generate Euler or Taylor approximate numerical solution and plot it

```
def numerical_solution(m,Y0,T,method,color='k',plot_type='time'):

    global nexty,f,ax

    t = 0
    y = np.array(Y0)

    h = (T-t)/m

    ta = np.empty(m+1)
    ta[0] = t

    ya = np.empty((len(Y0),m+1))  # a 2d array
    ya[:,0] = y
    for k in range(m): # 0, 1, 2, ...
        if method=='euler':
            y = y + h*f(t,y)  # scalar mult and addition done element-wise
        elif method=='taylor':
            y = nextY(y[0],y[1],y[2],t,h)
        else:
            assert(False) # exit if unimplemented method requested
        t += h
        ta[k+1] = t
        ya[:,k+1] = y

    if plot_type == 'time':
        d = len(Y0)
        for i in range(d):
            ax.plot(ta,ya[i,:],'-',color=color) # plot ith component vs t
    elif type(plot_type) == list:
            i,j = plot_type
            ax.plot(ya[i,:],ya[j,:],'-',color=color) # plot ith component vs jth

    return t,y  # return the final point
```

Generate the numerical approximate solutions

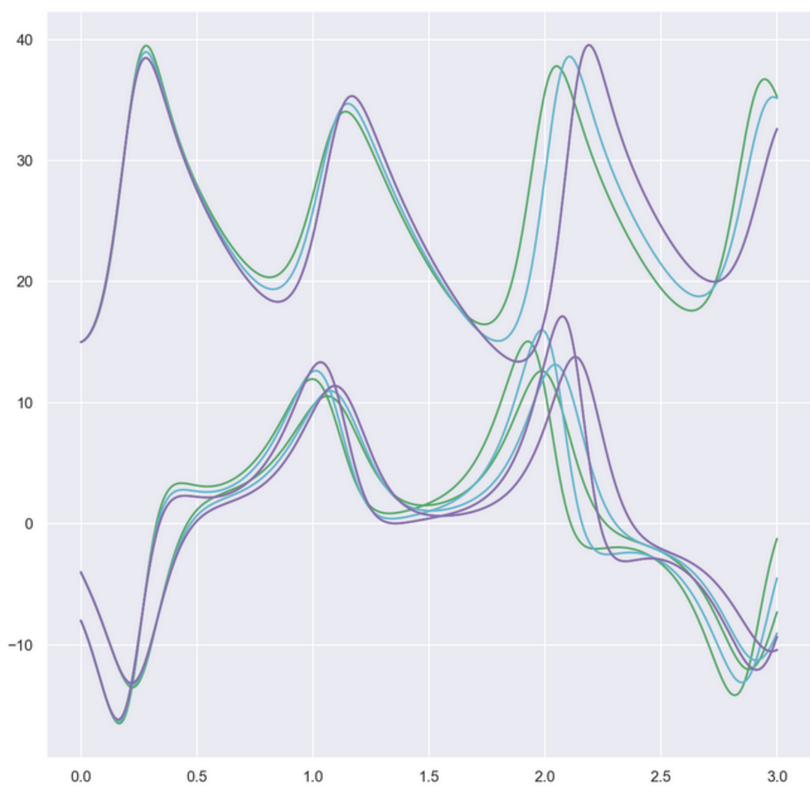
I am going to do two step sizes for each of Euler (green, cyan) and Taylor (red, purple).

```
Y0 = np.array([-4.,-8.,15.])
T = 1.
doplot = True

fig = plt.figure(figsize=(10,10))
ax = plt.subplot(111)

colors = 'gcrm'
nc = 0
for method in ['euler','taylor']:
    for m in [600,1200]:
        print(m)
        t1,y1 = numerical_solution(m,Y0,3.0,method,color=colors[nc])
        nc += 1
        print(y1)  # to see final value of y
```

```
600
[-7.29821011 -1.23697072 35.29220732]
1200
[-9.0372689  -4.50379722 35.13665865]
600
(-10.416387751599878, -9.339072685430786, 32.61181927320872)
1200
(-10.416387749870921, -9.339073794231501, 32.61181788493598)
```



We see that the two Euler approximations are greatly different, while the two Taylor approximations are concident on the plot, and in fact agree to 6 digits. Thus at this step size Taylor method has converged, while Euler is not even close.

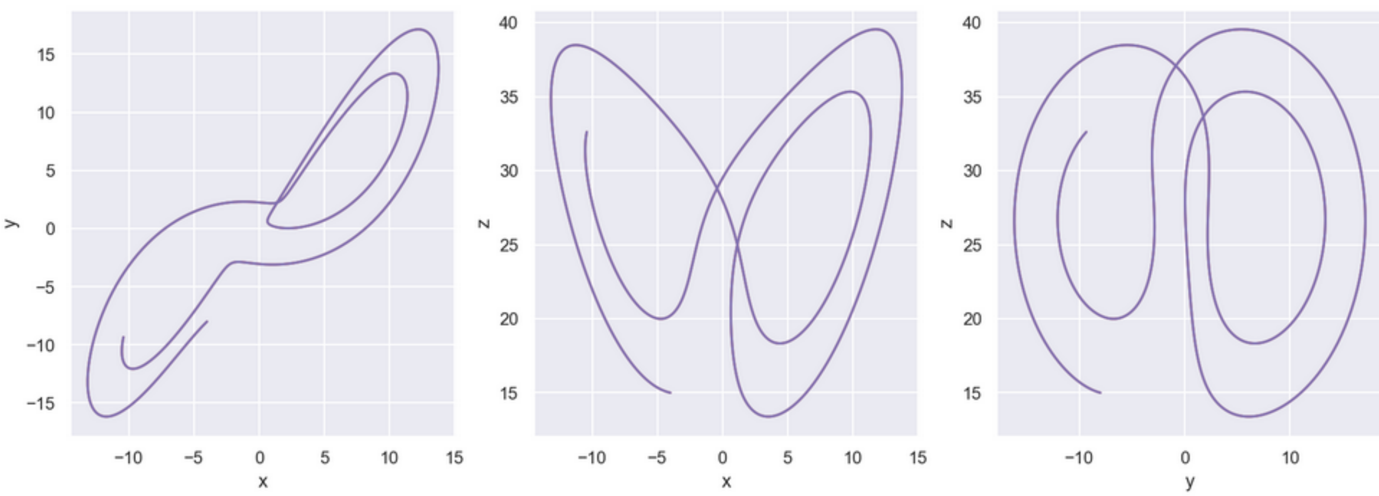
Now the xy , xz, and yz plots:

```
Y0 = np.array([-4.,-8.,15.])
T = 1.
doplot = True

#fig = plt.figure(figsize=(15,10))
fig,axes = plt.subplots(1,3, figsize=(15,5))

colors = 'rm'
nc = 0
for method in ['taylor']:
    for m in [600,1200]:
        print(m)
        ax = axes[0]; t1,y1 = numerical_solution(m,Y0,3.0,method,color=colors[nc],plot_type=[0,1])
        ax.set_xlabel('x'); ax.set_ylabel('y')
        ax = axes[1]; t1,y1 = numerical_solution(m,Y0,3.0,method,color=colors[nc],plot_type=[0,2])
        ax.set_xlabel('x'); ax.set_ylabel('z')
        ax = axes[2]; t1,y1 = numerical_solution(m,Y0,3.0,method,color=colors[nc],plot_type=[1,2])
        ax.set_xlabel('y'); ax.set_ylabel('z')
        nc += 1
        print(y1)  # to see final value of y
```

```
600
(-10.416387751599878, -9.339072685430786, 32.61181927320872)
1200
(-10.416387749870921, -9.339073794231501, 32.61181788493598)
```



(b)

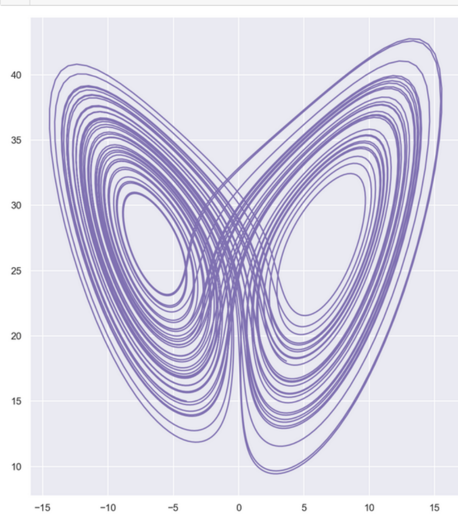
Taylor method for a longer stretch of time

```
Y0 = np.array([-4.,-8.,15.])
tfac = 20
T = 3.*tfac
doplot = True

fig = plt.figure(figsize=(10,10))
ax = plt.subplot(111,aspect=1)

t1,y1 = numerical_solution(300*tfac,Y0,T,method,color='m',plot_type=[0,2])

```



2. 3rd order 3-stage RK $\quad y^{(0)} = y^{(k)}$

(a)

$$K^{(1)} = f(y^{(0)}) \qquad y^{(1)} = y^{(k)} + hb_{21}K^{(1)}$$

$$K^{(2)} = f(y^{(1)}) \qquad y^{(2)} = y^{(k)} + hb_{31}K^{(1)} + hb_{32}K^{(2)}$$

$$K^{(3)} = f(y^{(2)}) \qquad y^{(3)} = y^{(k)} + hb_{41}K^{(1)} + hb_{42}K^{(2)} + hb_{43}K^{(3)} = y^{(k+1)}$$

RK formula:

$$\begin{aligned} y_i^{(k+1)} = y_i^{(k)} &+ hb_{41} f_i(y^{(k)}) \\ &+ hb_{42} f_i\big(y^{(k)} + \overbrace{hb_{21} f(y^{(k)})}^{"v"}\big) \\ &+ hb_{43} f_i\big(y^{(k)} + \underbrace{hb_{31} f(y^{(k)}) + hb_{32} f\big(y^{(k)} + \overbrace{hb_{21} f(y^{(k)})}^{"v"}\big)}_{"v"}\big) \end{aligned}$$

Taylor expansion for 2-variable function:

$$f_i(y^{(k)}+v) = f_i(y^{(k)}) + \partial_1 f_i \cdot v_1 + \partial_2 f_i \cdot v_2 + \frac{1}{2!}\left(\partial_1^2 f_i \cdot v_1^2 + 2\partial_1\partial_2 f_i \cdot v_1 v_2 + \partial_2^2 f_i \cdot v_2^2\right) + \ldots$$

where all the derivatives are evaluated at $y^{(k)}$.

$$\begin{aligned} y_i^{(k+1)} = y_i^{(k)} &+ hb_{41} f_i \\ &+ hb_{42}\Big[f_i + \partial_1 f_i \cdot hb_{21} f_1 + \partial_2 f_i \cdot hb_{21} f_2 \\ &\qquad + \frac{1}{2!}\big(\partial_1^2 f_i (hb_{21} f_1)^2 + 2\partial_1\partial_2 f_i \cdot (hb_{21} f_1)(hb_{21} f_2) + \partial_2^2 f_i \cdot (hb_{21} f_2)^2\big) + O(h^3)\Big] \\ &+ hb_{43}\Big[f_i + \partial_1 f_i \big\{ hb_{31} f_1 + hb_{32}\left[f_1 + \partial_1 f_1 \cdot hb_{21} f_1 + \partial_2 f_1 \cdot hb_{21} f_2 + O(h^2)\right]\big\} \\ &\qquad + \partial_2 f_i \big\{ hb_{31} f_2 + hb_{32}\left[f_2 + \partial_1 f_2 \cdot hb_{21} f_1 + \partial_2 f_2 \cdot hb_{21} f_2 + O(h^2)\right]\big\} \\ &\qquad + \frac{1}{2!}\Big(\partial_1^2 f_i \left(hb_{31} f_1 + hb_{32} f_1 + O(h^2)\right)^2 \\ &\qquad\qquad + 2\partial_1\partial_2 f_i \left(hb_{31} f_1 + hb_{32} f_1 + O(h^2)\right)\left(hb_{31} f_2 + hb_{32} f_2 + O(h^2)\right) \\ &\qquad\qquad + \partial_2^2 f_i \left(hb_{31} f_2 + hb_{32} f_2 + O(h^2)\right)^2 \Big) \\ &\qquad + O(h^3)\Big] \end{aligned}$$

$$\begin{aligned} = y_i^{(k)} &+ h(b_{41} + b_{42} + b_{43}) f_i \\ &+ h^2\left[b_{42} b_{21} (\partial_1 f_i \cdot f_1 + \partial_2 f_i \cdot f_2) + b_{43}(b_{31} + b_{32})(\partial_1 f_i \cdot f_1 + \partial_2 f_i \cdot f_2)\right] \\ &+ h^3 \Big[b_{42} \cdot \tfrac{1}{2} b_{21}^2 \left(\partial_1^2 f_i \cdot f_1^2 + 2\partial_1\partial_2 f_i \cdot f_1 f_2 + \partial_2^2 f_i \, f_2^2\right) \\ &\qquad + b_{43}\, b_{32}\, b_{21} \left\{ \partial_1 f_i \left(\partial_1 f_1 \cdot f_1 + \partial_2 f_1 \cdot f_2\right) + \partial_2 f_i \left(\partial_1 f_2 \cdot f_1 + \partial_2 f_2 \cdot f_2\right)\right\} \\ &\qquad + b_{43} \tfrac{1}{2}(b_{31} + b_{32})^2 \left(\partial_1^2 f_i \cdot f_1^2 + 2\partial_1\partial_2 f_i \cdot f_1 f_2 + \partial_2^2 f_i \cdot f_2^2\right)\Big] + O(h^4) \end{aligned}$$

On the other hand,

$$\begin{aligned} y_i(t_k + h) &= y_i(t_k) + h y_i'(t_k) + \frac{h^2}{2!} y_i''(t_k) + \frac{h^3}{3!} y_i'''(t_k) + O(h^4) \\ &= y_i^{(k)} + h f_i + \frac{h^2}{2!}(\partial_1 f_i \cdot f_1 + \partial_2 f_i \cdot f_2) \\ &\quad + \frac{h^3}{3!}\left[\begin{aligned} &\partial_1^2 f_i \, f_1 f_1 + \partial_2\partial_1 f_i \cdot f_1 f_2 + \partial_1 f_i \cdot (\partial_1 f_1 + \partial_2 f_1) \\ &+ \partial_1\partial_2 f_i \cdot f_2 f_1 + \partial_2^2 f_i \cdot f_2^2 + \partial_2 f_i \cdot (\partial_1 f_2 + \partial_2 f_2) \end{aligned}\right] + O(h^4) \end{aligned}$$

	RK	solution
$O(1)$	$y_i^{(k)}$	$y_i^{(k)}$
$O(h)$	$b_{43} + b_{42} + b_{41}$	1
$O(h^2)$	$\{b_{42} b_{21} + b_{43}(b_{31} + b_{32})\}(\partial_1 f_i \cdot f_1 + \partial_2 f_i \cdot f_2)$	$\frac{1}{2!}(\partial_1 f_i \cdot f_1 + \partial_2 f_i \cdot f_2)$
$O(h^3)$	$\{b_{42} \frac{1}{2} b_{21}^2 + b_{43} \frac{1}{2}(b_{31} + b_{32})^2\}(\partial_1^2 f_i \cdot f_1^2 + 2\partial_1\partial_2 f_i \cdot f_1 f_2 + \partial_2^2 f_i \, f_2^2)$ $+ b_{43}\, b_{32}\, b_{21} \{\partial_1 f_i (\partial_1 f_1 \cdot f_1 + \partial_2 f_1 \cdot f_2) + \partial_2 f_i (\partial_1 f_2 \cdot f_1 + \partial_2 f_2 \cdot f_2)\}$	$\partial_1^2 f_i \, f_1 f_1 + 2\partial_2\partial_1 f_i \cdot f_1 f_2 + \partial_1 f_i \cdot (\partial_1 f_1 + \partial_2 f_1)$ $+ \partial_2^2 f_i \cdot f_2^2 + \partial_2 f_i \cdot (\partial_1 f_2 + \partial_2 f_2)$

So error $= O(h^4)$ iff

$$\begin{aligned} b_{43} + b_{42} + b_{41} &= 1 \\ b_{42} b_{21} + b_{43}(b_{31} + b_{32}) &= \tfrac{1}{2} \\ b_{42} \cancel{\tfrac{1}{2}} b_{21}^2 + b_{43} \cancel{\tfrac{1}{2}} (b_{31} + b_{32})^2 &= \cancel{\tfrac{1}{6}}\ \tfrac{1}{3} \\ b_{43}\, b_{32}\, b_{21} &= \tfrac{1}{6} \end{aligned}$$

This is 4 equations in 6 degrees of freedom.
We would expect a 2-parameter family of methods.
The Wikipedia page presents a 1-parameter family called "generic third-order method". I checked that the values of that method satisfy my 4 equations for all values of their parameter.
But it seems there's a whole other dimension of methods.

(b) There are many solutions of the above 4 equations in 6 unknowns.
Sympy solve finds the following 4 options if we set b41=1/4 and b42=3/4.
My examples have irrational coefficients.
There seems to be a preference for methods with rational coefficients.

```python
import sympy as sp
sp.init_printing()
```

```python
b43,b42,b41,b32,b31,b21 = sp.symbols('b43,b42,b41,b32,b31,b21')
```

```python
eqs6 = [b43+b42+b41,
        b42*b21 + b43*(b31+b32) - sp.Rational(1,2),
        b42*b21**2 + b43*(b31+b32)**2 - sp.Rational(1,3),
        b43*b32*b31 - sp.Rational(1,6)]
```

```python
eqs6
```

$$\left[b_{41} + b_{42} + b_{43},\ b_{21} b_{42} + b_{43}\left(b_{31} + b_{32}\right) - \frac{1}{2},\ b_{21}^{2} b_{42} + b_{43}(b_{31} + b_{32})^{2} - \frac{1}{3},\ b_{31} b_{32} b_{43} - \frac{1}{6}\right]$$

```python
o2 = sp.Rational(1,2)
o3 = sp.Rational(1,3)
o4 = sp.Rational(1,4)

eqs = [eq.subs({b41:o4,b42:3*o4}) for eq in eqs6]
sp.solve(eqs)
```

$$\left[\left\{b_{21} : -\frac{14}{3},\ b_{31} : -2 + \frac{5\sqrt{6}}{6},\ b_{32} : -\frac{5\sqrt{6}}{6} - 2,\ b_{43} : -1\right\},\right.$$
$$\left\{b_{21} : -\frac{14}{3},\ b_{31} : -\frac{5\sqrt{6}}{6} - 2,\ b_{32} : -2 + \frac{5\sqrt{6}}{6},\ b_{43} : -1\right\},$$
$$\left.\left\{b_{21} : \frac{2}{3},\ b_{31} : -\frac{\sqrt{6}}{6},\ b_{32} : \frac{\sqrt{6}}{6},\ b_{43} : -1\right\},\ \left\{b_{21} : \frac{2}{3},\ b_{31} : \frac{\sqrt{6}}{6},\ b_{32} : -\frac{\sqrt{6}}{6},\ b_{43} : -1\right\}\right]$$