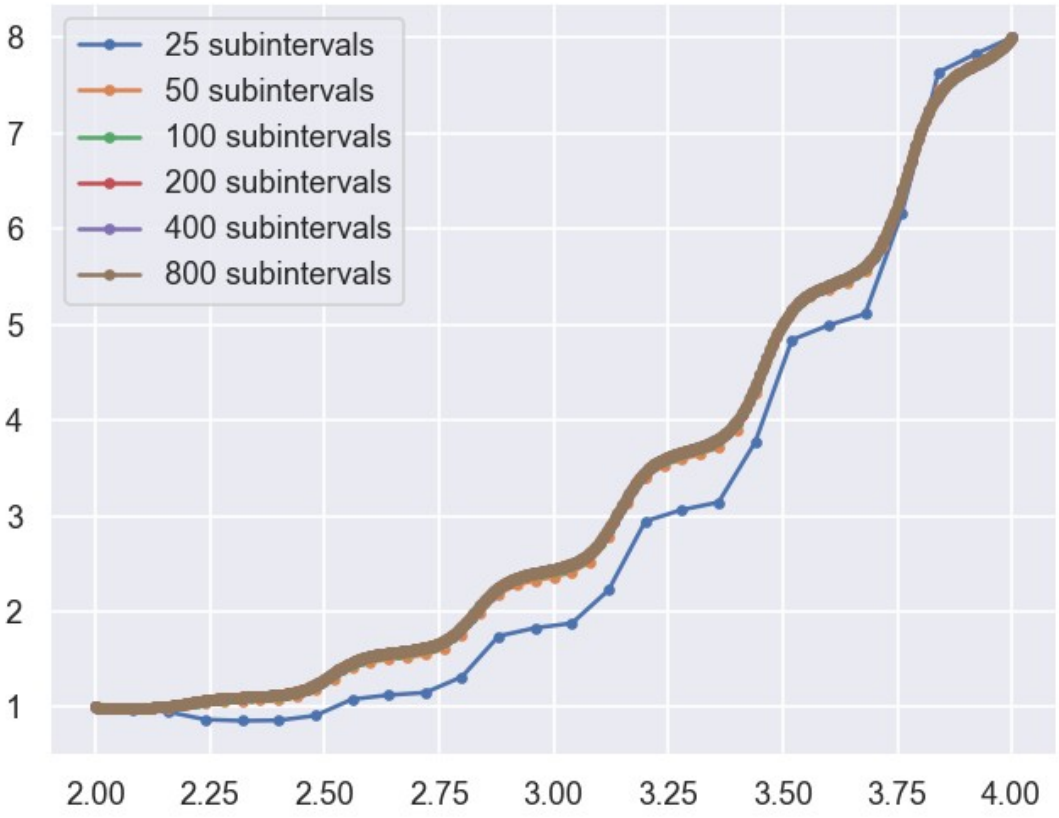
538 Spring 2022 Homework 5 Solutions

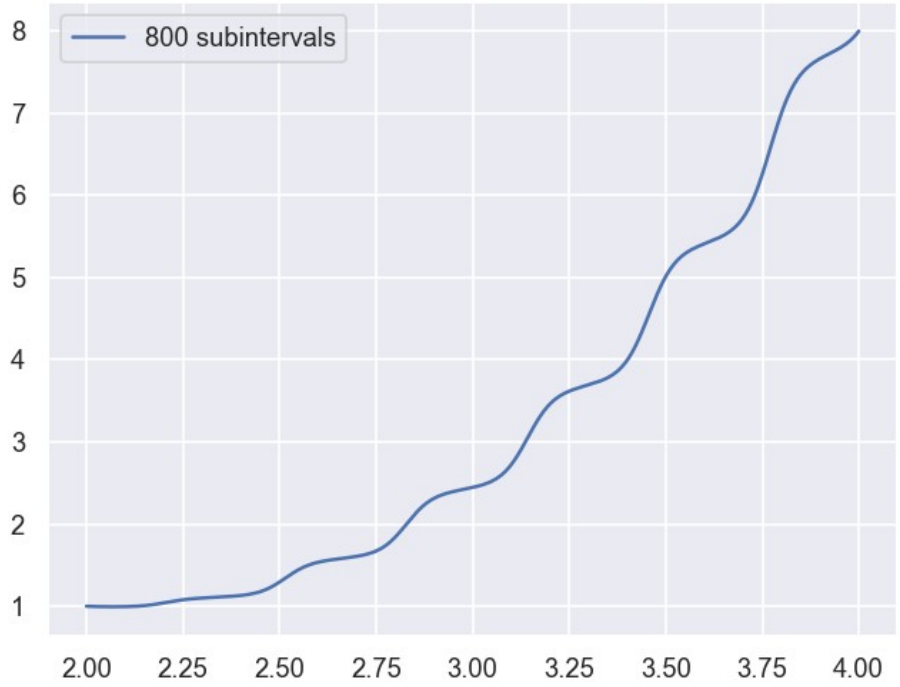
1. BVP by finite differences
Code:

```
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 notebook
#%matplotlib inline
sns.set()
import numpy as np

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

# coefficiencts of some random example 2nd order linear ODE
def p(x):   return 20*np.sin(20*x)
def q(x):   return -6/x**2
def phi(x): return 0*x + 1

x0 = 2  # a
x1 = 4  # b
y0 = 1  # alpha, the starting value
y1 = 8  # beta, the target ending value

#plt.figure(figsize=(10,6))
plt.plot(x0,y0,'o')
plt.plot(x1,y1,'o')

%matplotlib notebook
midpoint_values = []

for N in [24,49,99,199,399,799]:
    h = (x1-x0)/(N+1)
    x = np.linspace(x0,x1,N+2)
    pv = p(x)
    qv = q(x)
    phiv = phi(x)

    # form the matrix A
    A = np.zeros((N,N))
    i = np.arange(N+2)
    A[i[ :-2],i[ :-2]] = -2 + h**2*qv[1:-1]  # main diagonal
    A[i[1:-2],i[ :-3]] =  1 - h/2 *pv[2:-1]  # subdiagonal
    A[i[ :-3],i[1:-2]] =  1 + h/2 *pv[1:-2]  # superdiagonal
    A

    # rhs
    rhs = h**2*phiv[1:-1]
    rhs
    rhs[0 ] -= (1-h/2*pv[1])*y0
    rhs[-1] -= (1+h/2*pv[N])*y1
    rhs

    u = np.zeros_like(x)
    u[ 0] = y0
    u[-1] = y1
    u[1:-1] = np.linalg.solve(A,rhs)
    u
    plt.plot(x,u,'.-',label=f'{N+1} subintervals')
    print(N+1,u[(N+1)//2])  # value at midpoint
    midpoint_values.append(u[(N+1)//2])
plt.legend()
```

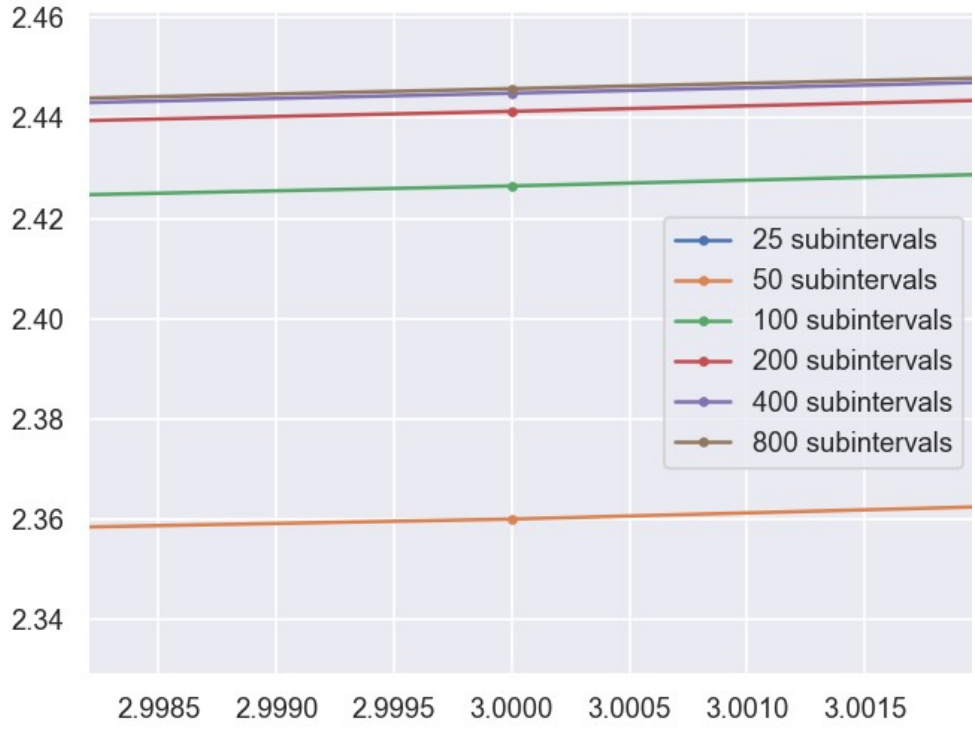
Results for various choices of number of gridpoints:



Appears to be converging well.
Best approximation:



Zooming in on x=3:
By eyeball, each doubling of the number of subintervals takes care of at least 3/4 of the error, indicating the error is O(h^2), or even better.
The best estimate of y(3) is approximately 2.446.



```
25 1.8282434586058591
50 2.359954018224484
100 2.426466898570565
200 2.4413598076637095
400 2.444986299125836
800 2.4458870396249535
```

2. Variational equations in Lorenz system¶

```
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/5 + x*y ])

def Df(t,Y):
    x,y,z = Y
    return np.array([[-10, 10, 0],
                     [28-z, -1, -x],
                     [y, x, -8/5]])

ODEF = f
ODEDF = Df

Y = [-4, -8, 15]
YV = np.append(Y,np.eye(3).reshape(9))
F(0,YV)
t1 = 10
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');
```

```
'Singular values at t =10: [1.66961456e+03 4.23324431e-01 9.43744719e-14]'
```

```
print('Final values:',yva[:3,-1])
Final values: [ 1.78029292  2.88998527 15.34319187]
```

(b) The singular values are approximately 1700, 0.4, and 10^{-13}.
The large one, 1700, corresponds to rapid exponential separation of nearby trajectories.
The middle one, 0.4, is close to 1, and denotes little or no exponential separation of trajectories along some direction, which is likely to be in the direction along the trajectory. (Consider cars on a one-lane circular track: they can separate and bunch a bit, but cannot separate or converge exponentially.) The very small one indicates there is a direction along which trajectories converge very strongly. This is reflected in the long-term trajectory of the system being confined to a very thin sheet.

3. Dependence on a parameter.

(a)

IVP $\dfrac{\partial y(t,p)}{\partial t} = f(y(t,p),p)$, $y(0,p) = y_0$

Let $v(t,p) \equiv \dfrac{\partial y(t,p)}{\partial p}$.

Then $\boxed{v(0,p) = 0}$ and

$$\frac{\partial v(t,p)}{\partial t} = \frac{\partial}{\partial t}\left(\frac{\partial y(t,p)}{\partial p}\right)$$

$$= \frac{\partial}{\partial p}\left(\frac{\partial y(t,p)}{\partial t}\right)$$ assuming 2nd derivatives are continuous

$$= \frac{\partial}{\partial p} f(y(t,p),p)$$

$$= \partial_1 f(y(t,p),p)\cdot \frac{\partial}{\partial p} y(t,p) + \partial_2 f(y(t,p),p)$$

That is,

$$\boxed{\frac{\partial v(t,p)}{\partial t} = \partial_1 f(y(t,p),p)\cdot v(t,p) + \partial_2 f(y(t,p),p)}$$

which we can co-solve with the IVP for y.

(b) For $f(y,p) = py^3$

$\partial_1 f(y,p) = 3py^2$, $\partial_2 f(y,p) = y^3$

So $\dfrac{\partial v(t,p)}{\partial t} = 3p\,y(t,p)^2 \cdot v(t,p) + y(t,p)^3$

and $v(0,p) = 0$.

That is, we solve

$$\boxed{\begin{aligned} y' &= py^3 \quad , \quad y(0) = y_0 \\ v' &= 3py^2 v + y^3 \quad , \quad v(0) = 0 \end{aligned}}$$