538 Spring 2025 Homework 4 Solutions

1. 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

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

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

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

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

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 $v(0,p) = 0$.

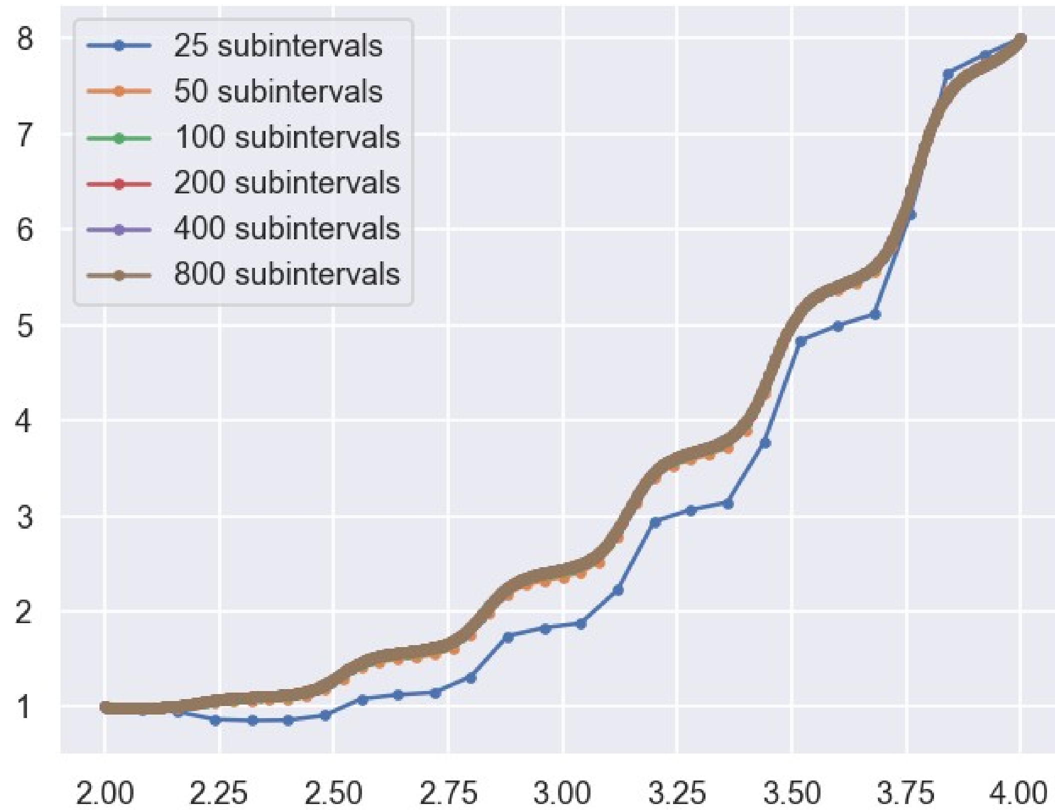
That is, we solve

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

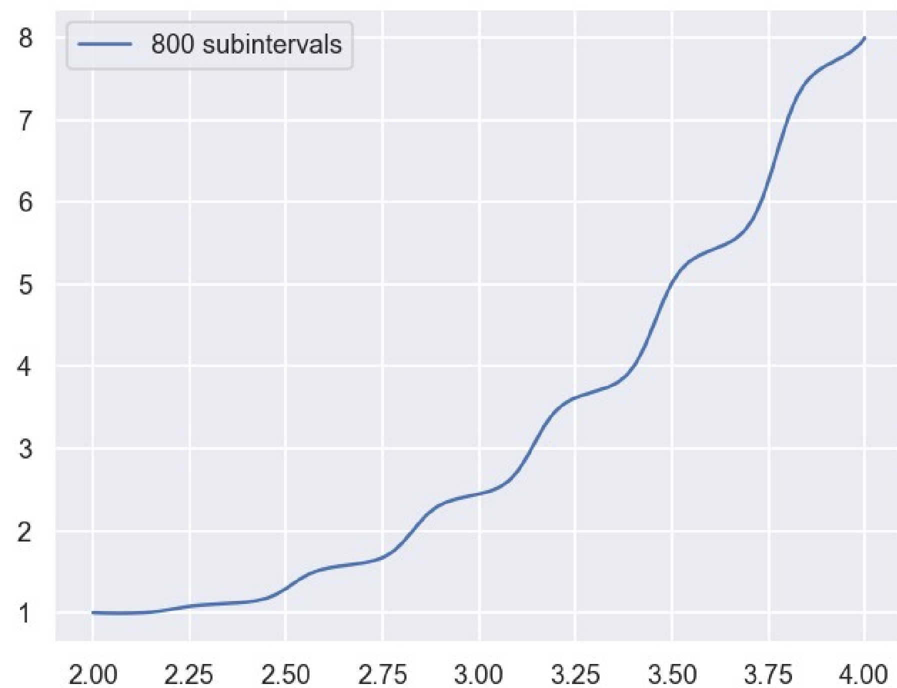
2. BVP by finite differences
Code:

```
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()
```

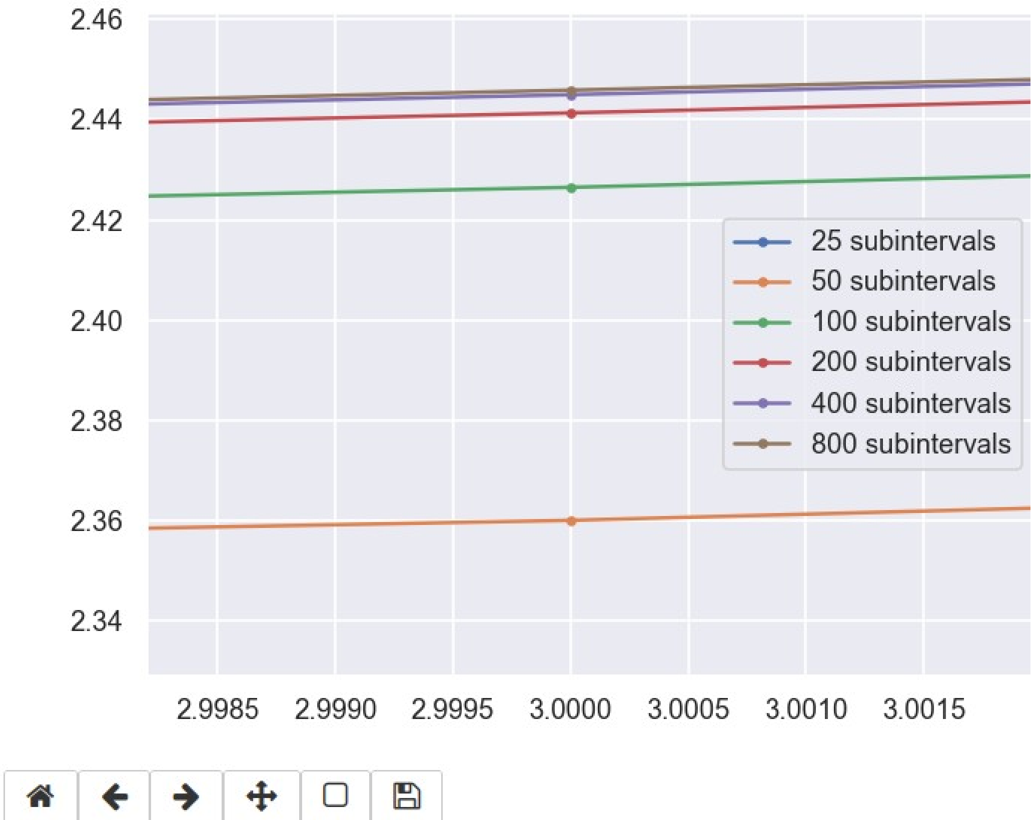
Results for various choices of number of gridpoints:



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
```

3. Off-center finite-difference approximation to y''

(a) I'm doing this problem with the computer. It may be a better learning experience for you to do it with pencil and paper. I haven't come to a firm conclusion about this.

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

```
y = sp.symbols('y0:4')  # y and its derivatives
y
```

$(y_0,\ y_1,\ y_2,\ y_3)$

```
y = sp.symbols('y0:4')
a = sp.symbols('a0:4')
h = sp.symbols('h')
def tay(h):
    return sum( [ h**j/sp.factorial(j)*y[j] for j in range(len(y))] )
tay(h)
```

$$\frac{h^3 y_3}{6} + \frac{h^2 y_2}{2} + h y_1 + y_0$$

```
expr = sp.expand((a[0]*tay(0) + a[1]*tay(h) + a[2]*tay(2*h)))
expr
```

Here is the start of Taylor expansion of an arbitrary linear combination of our data.

$$a_0 y_0 + \frac{a_1 h^3 y_3}{6} + \frac{a_1 h^2 y_2}{2} + a_1 h y_1 + a_1 y_0 + \frac{4 a_2 h^3 y_3}{3} + 2 a_2 h^2 y_2 + 2 a_2 h y_1 + a_2 y_0$$

```
coeffs = [sp.simplify(expr.coeff(yj)) for j,yj in enumerate(y)]
coeffs
```

Here are the coefficients of each of the 0th, 1st, 2nd, 3rd derivatives at 0:

$$\left[a_0 + a_1 + a_2,\ h\left(a_1 + 2a_2\right),\ \frac{h^2\left(a_1 + 4a_2\right)}{2},\ \frac{h^3\left(a_1 + 8a_2\right)}{6}\right]$$

```
eqns = [coeffs[0],coeffs[1],coeffs[2]-1]
eqns

```

Here are the 3 things we want to be zero, in order to aproximate y''(0):

$$\left[a_0 + a_1 + a_2,\ h\left(a_1 + 2a_2\right),\ \frac{h^2\left(a_1 + 4a_2\right)}{2} - 1\right]$$

coeff of y(0) = 0
coeff of y'(0) = 0
coeff of y''(0) = 1

```
sol = sp.solve(eqns,a[0:3])
sol
```

Choices of the a's that make the above all 0s:

$$\left\{a_0 : \frac{1}{h^2},\ a_1 : -\frac{2}{h^2},\ a_2 : \frac{1}{h^2}\right\}$$

These values are required in order to get approximation of y''.

```
sp.expand(expr.subs(sol))
```

$h y_3 + y_2$ With these coefficients the leading error term is hy''', so of order h, not h^2.

Here is the start of the Taylor expansion of our linear combination of the data with the determined values of the a's.
We have y_2 as desired, but we also have an error whose Taylor expansion starts at order h.
Hence no combination of the given data provides a formula for y''(0) with error = O(h^2).

.b A 4-point O(h^2) approx to y'' at the boundary

```python
y = sp.symbols('y0:5')
a = sp.symbols('a0:5')
h = sp.symbols('h')
def tay(h):
    return sum( [ h**j/sp.factorial(j)*y[j] for j in range(len(y))] )
tay(h)
expr = sp.expand( sum([a[j]*tay(j*h) for j in range(4)]) )
display(expr)
coeffs = [sp.simplify(expr.coeff(yj)) for j,yj in enumerate(y)]
display(coeffs)
eqns = [coeffs[0],coeffs[1],coeffs[2]-1,coeffs[3]]
display(eqns)
sol = sp.solve(eqns,a[0:4])
display(sol)
sp.expand(expr.subs(sol))
```

$$a_0 y_0 + \frac{a_1 h^4 y_4}{24} + \frac{a_1 h^3 y_3}{6} + \frac{a_1 h^2 y_2}{2} + a_1 h y_1 + a_1 y_0 + \frac{2a_2 h^4 y_4}{3} + \frac{4a_2 h^3 y_3}{3} + 2a_2 h^2 y_2 + 2a_2 h y_1$$
$$+ a_2 y_0 + \frac{27a_3 h^4 y_4}{8} + \frac{9a_3 h^3 y_3}{2} + \frac{9a_3 h^2 y_2}{2} + 3a_3 h y_1 + a_3 y_0$$

$$\left[a_0 + a_1 + a_2 + a_3,\ h(a_1 + 2a_2 + 3a_3),\ \frac{h^2(a_1 + 4a_2 + 9a_3)}{2},\ \frac{h^3(a_1 + 8a_2 + 27a_3)}{6},\ \frac{h^4(a_1 + 16a_2 + 81a_3)}{24}\right]$$

$$\left[a_0 + a_1 + a_2 + a_3,\ h(a_1 + 2a_2 + 3a_3),\ \frac{h^2(a_1 + 4a_2 + 9a_3)}{2} - 1,\ \frac{h^3(a_1 + 8a_2 + 27a_3)}{6}\right]$$

$$\left\{a_0 : \frac{2}{h^2},\ a_1 : -\frac{5}{h^2},\ a_2 : \frac{4}{h^2},\ a_3 : -\frac{1}{h^2}\right\}$$

$$-\frac{11h^2 y_4}{12} + y_2$$

Thus, with 4 pieces of data, we can find an approximation of y''(0) with error = O(h^2).

For the centered difference using y(-h), y(0), y(h), we have this:

```python
y = sp.symbols('y0:5')
a = sp.symbols('a0:5')
h = sp.symbols('h')
def tay(h):
    return sum( [ h**j/sp.factorial(j)*y[j] for j in range(len(y))] )
tay(h)
expr = sp.expand( sum([a[j]*tay((j-1)*h) for j in range(3)]) )
display(expr)
coeffs = [sp.simplify(expr.coeff(yj)) for j,yj in enumerate(y)]
display(coeffs)
eqns = [coeffs[0],coeffs[1],coeffs[2]-1,coeffs[3]]
display(eqns)
sol = sp.solve(eqns,a[0:4])
display(sol)
sp.expand(expr.subs(sol))
```

$$\frac{a_0 h^4 y_4}{24} - \frac{a_0 h^3 y_3}{6} + \frac{a_0 h^2 y_2}{2} - a_0 h y_1 + a_0 y_0 + a_1 y_0 + \frac{a_2 h^4 y_4}{24} + \frac{a_2 h^3 y_3}{6} + \frac{a_2 h^2 y_2}{2} + a_2 h y_1 + a_2 y_0$$

```
[a0 + a1 + a2,
 h*(-a0 + a2),
 h**2*(a0 + a2)/2,
 h**3*(-a0 + a2)/6,
 h**4*(a0 + a2)/24]

[a0 + a1 + a2, h*(-a0 + a2), h**2*(a0 + a2)/2 - 1, h**3*(-a0 + a2)/6]

{a0: h**(-2), a1: -2/h**2, a2: h**(-2)}
```

$$\frac{h^2 y_4}{12} + y_2$$

and we see that the 4-point off-center formula is **11 times worse** than the 3-point centered difference.

4. Galerkin method with a 2D basis

Ackleh et al. Exercise 10.3.6 (small Galerkin)

Extra credit: Extend the basis to $\{\sin(\pi x),\ \ \sin(2\pi x), \ldots, \sin(N\pi x)\}$ and explore how the error depends on N.

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

import numpy as np
import sympy as sp
sp.init_printing()
from matplotlib import rcdefaults
rcdefaults()
%config InlineBackend.figure_format='retina'
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set()
```

Exact solution

This is a constant-coefficient 2nd order linear ODE, whose general solution we can find using the methods of our undergrad ODE course. Find roots of characteristic equation, etc.
I am using sympy to find the coefficients to satisfy the BCs.

```
x = sp.symbols('x')
c = sp.symbols('c0:2')
y = c[0]*sp.exp(sp.pi/2*x) + c[1]*sp.exp(-sp.pi/2*x) + x/(sp.pi/2)**2
y,y.subs({x:0}),y.subs({x:1})
sol = sp.solve( [y.subs({x:0}),y.subs({x:1})], c )
yexact = y.subs(sol)
display(yexact)
checkde = sp.simplify( -sp.diff( yexact, x, x) + (sp.pi/2)**2*yexact - x )
print( 'diff eq satisfied:',checkde == 0 )
yexactp = sp.diff(yexact,x)
yexactfunc = sp.lambdify( x, yexact, 'numpy' )  # for plotting it
```

$$\frac{4x}{\pi^2} - \frac{2e^{\frac{\pi x}{2}}}{\pi^2 \sinh\left(\frac{\pi}{2}\right)} + \frac{2e^{-\frac{\pi x}{2}}}{\pi^2 \sinh\left(\frac{\pi}{2}\right)}$$

```
diff eq satisfied: True
```

Code to generate Galerkin approximations:

```
def norm1( expr, x ):
    exprp = sp.diff(expr,x)
    integrand = exprp**2 + expr**2
    inp = sp.lambdify( x, integrand, 'numpy' )
    Q,E = quadrature(inp,0,1) # Gaussian quadrature
    return np.sqrt(Q)

fig,(ax0,ax1,ax2) = plt.subplots(3,1,figsize=(15,15))
Ns = [2,4,8,16,32,64]#,128]
norm1errors = []
for N in Ns:
    phis = [ sp.sin(i*sp.pi*x) for i in range(1,N+1) ]
    phips = [ sp.diff( phi, x) for phi in phis ]
    phipps = [ sp.diff( phip, x) for phip in phips ]
    phis,phips,phipps
    def L(Y): return -sp.diff( Y, x, x) + (sp.pi/2)**2*Y
    f = x
    A = ([[0]*N])*N
    for i in range(N):
        #for j in range(N):  basis functions are orthogogonal - hence all off-diagonal elements are zero
        j=i
        A[i][j] =  sp.integrate( L(phis[i])*phis[j], (x,0,1) )

    A = np.array(A).reshape((N,N))
    b = np.array([ sp.integrate(phi*f, (x,0,1)) for phi in phis ])
    exacta = [b[i]/A[i,i] for i in range(N)]  # only true because this A is diagonal
    Y = sum( [ ai*phi for ai,phi in zip(exacta,phis) ] )

    exacterror = sp.expand( Y - yexact )
    exacterrorp = sp.diff(exacterror, x)
    exacterrorpfunc = sp.lambdify( x, exacterrorp, 'numpy')
    display(Y)
    Yfunc = sp.lambdify( x, Y, 'numpy' )

    xa = np.linspace(0,1,2000)
    ax0.plot(xa,Yfunc(xa),label=str(N));
    ax1.plot(xa,N**2*(Yfunc(xa)-yexactfunc(xa)),label=str(N))
    ax2.plot(xa,N*(exacterrorpfunc(xa)),label=str(N))
    Yp = sp.diff(Y,x)
    #norm1errors.append( np.sqrt(float(sp.integrate( (Yp-yexactp)**2 + (Y-yexact)**2, (x,0,1)))) )
    # doing the 1-norm integral exactly seems to take forever with sympy, so do numerically:
    norm1errors.append( norm1( exacterror, x ) )
ax0.plot(xa,yexactfunc(xa),'k');
ax0.legend()
ax1.legend();
ax1.set_title('$N^2$ times pointwise error')
ax2.set_title('$N$ times pointwise error of derivative');
```

I will be looking at this norm of the error in the extra-credit part the 1-norm on Ackleh p549.

Integral will be done by numerical quadrature.

Here is the answer to the problem as posed in Ackleh (N=2).

The Galerkin approximations.
The basis functions are orthogonal, so the coefficients don't change as we add more basis functions.

$$\frac{8\sin(\pi x)}{5\pi^3} - \frac{4\sin(2\pi x)}{17\pi^3}$$

$$\frac{8\sin(\pi x)}{5\pi^3} - \frac{4\sin(2\pi x)}{17\pi^3} + \frac{8\sin(3\pi x)}{111\pi^3} - \frac{2\sin(4\pi x)}{65\pi^3}$$

$$\frac{8\sin(\pi x)}{5\pi^3} - \frac{4\sin(2\pi x)}{17\pi^3} + \frac{8\sin(3\pi x)}{111\pi^3} - \frac{2\sin(4\pi x)}{65\pi^3} + \frac{8\sin(5\pi x)}{505\pi^3} - \frac{4\sin(6\pi x)}{435\pi^3} + \frac{8\sin(7\pi x)}{1379\pi^3} - \frac{\sin(8\pi x)}{257\pi^3}$$

$$\frac{8\sin(\pi x)}{5\pi^3} - \frac{4\sin(2\pi x)}{17\pi^3} + \frac{8\sin(3\pi x)}{111\pi^3} - \frac{2\sin(4\pi x)}{65\pi^3} + \frac{8\sin(5\pi x)}{505\pi^3} - \frac{4\sin(6\pi x)}{435\pi^3} + \frac{8\sin(7\pi x)}{1379\pi^3} - \frac{\sin(8\pi x)}{257\pi^3} + \frac{8\sin(9\pi x)}{2925\pi^3} - \frac{4\sin(10\pi x)}{2005\pi^3} + \frac{8\sin(11\pi x)}{5335\pi^3} - \frac{2\sin(12\pi x)}{1731\pi^3} + \frac{8\sin(13\pi x)}{8801\pi^3} - \frac{4\sin(14\pi x)}{5495\pi^3} + \frac{8\sin(15\pi x)}{13515\pi^3} - \frac{\sin(16\pi x)}{2050\pi^3}$$

$$\frac{8\sin(\pi x)}{5\pi^3} - \frac{4\sin(2\pi x)}{17\pi^3} + \frac{8\sin(3\pi x)}{111\pi^3} - \frac{2\sin(4\pi x)}{65\pi^3} + \frac{8\sin(5\pi x)}{505\pi^3} - \frac{4\sin(6\pi x)}{435\pi^3} + \frac{8\sin(7\pi x)}{1379\pi^3} - \frac{\sin(8\pi x)}{257\pi^3} + \frac{8\sin(9\pi x)}{2925\pi^3} - \frac{4\sin(10\pi x)}{2005\pi^3} + \frac{8\sin(11\pi x)}{5335\pi^3} - \frac{2\sin(12\pi x)}{1731\pi^3} + \frac{8\sin(13\pi x)}{8801\pi^3} - \frac{4\sin(14\pi x)}{5495\pi^3} + \frac{8\sin(15\pi x)}{13515\pi^3} - \frac{\sin(16\pi x)}{2050\pi^3} + \frac{8\sin(17\pi x)}{19669\pi^3} - \frac{4\sin(18\pi x)}{11673\pi^3} + \frac{8\sin(19\pi x)}{27455\pi^3} - \frac{2\sin(20\pi x)}{8005\pi^3} + \frac{8\sin(21\pi x)}{37065\pi^3} - \frac{4\sin(22\pi x)}{21307\pi^3} + \frac{8\sin(23\pi x)}{48691\pi^3} - \frac{\sin(24\pi x)}{6915\pi^3} + \frac{8\sin(25\pi x)}{62525\pi^3} - \frac{4\sin(26\pi x)}{35165\pi^3} + \frac{8\sin(27\pi x)}{78759\pi^3} - \frac{2\sin(28\pi x)}{21959\pi^3} + \frac{8\sin(29\pi x)}{97585\pi^3} - \frac{4\sin(30\pi x)}{54015\pi^3} + \frac{8\sin(31\pi x)}{119195\pi^3} - \frac{\sin(32\pi x)}{16388\pi^3}$$

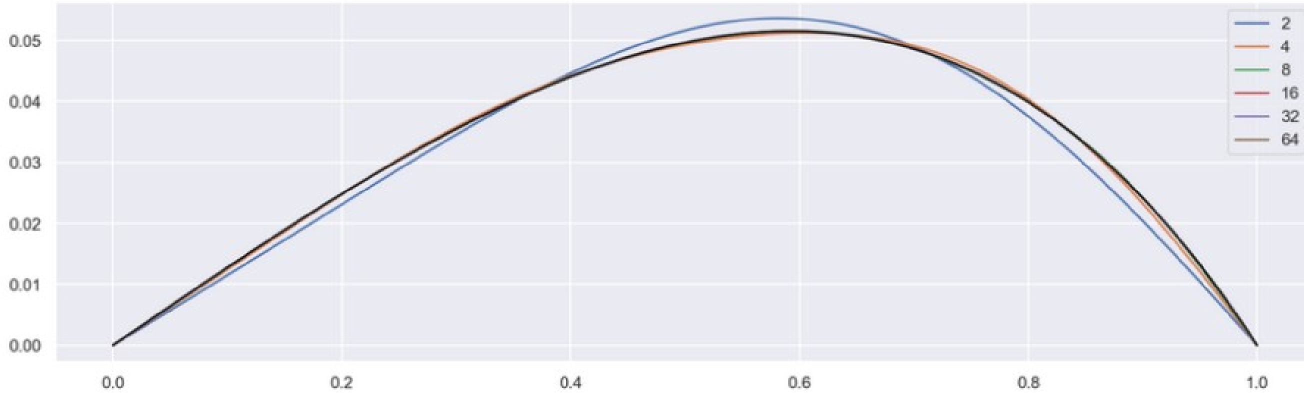
Here is the comparison requested by Ackleh of the N=2 Galerkin approximation and the exact solution at x=0.25, 0.5, 0.75:

```python
print()
for xval in [0.25, 0.5, 0.75]:
    print(f'{xval}\t{float(yexact.subs({x:xval}))}\t{Yfunc(xval)}\t{Yfunc(xval)-float(yexact.subs({x:xval}))}')
```

x	exact solution	Galerkin approx	error in Galerkin approx
0.25	0.030371158142647698	0.02889984958499546	-0.001471308557652238
0.5	0.049659597553640585	0.05160245509311919	0.0019428575394786068
0.75	0.045051428108221124	0.04407704225944229	-0.0009743858487788332

Below (extra credit) I explore the convergence of the Galerkin approximation as N is repeatedly doubled.

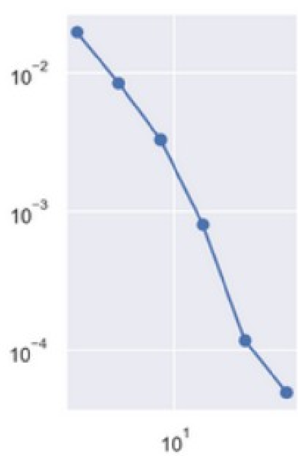
Plots of the Galerkin approximations and the exact solution (black):



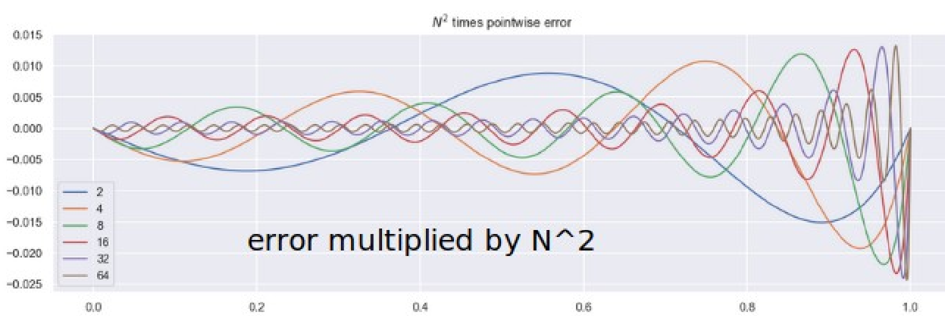
A log-log plot of the errors in the numerically estimated "1-norm" as defined in Ackleh p549 square root of integral of e'^2 + e^2:

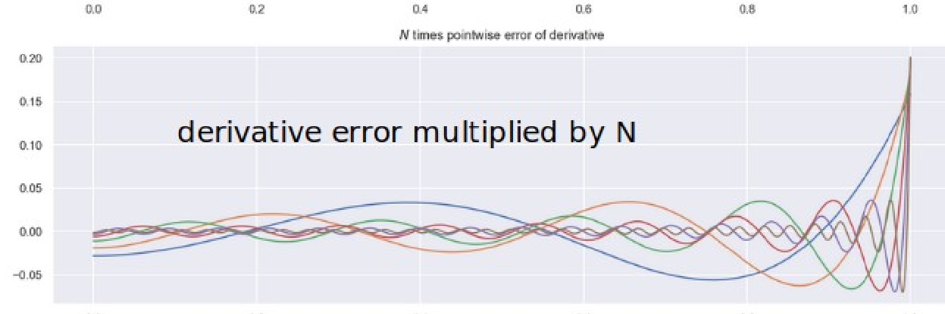
```
plt.subplot(111,aspect=1)
plt.loglog(Ns,norm1errors,'o-')
np.polyfit(np.log10(Ns),np.log10(norm1errors),1)[0]
```

−1.819563400336292



Slope is roughly −2 as we were told to expect. It's a bit wobbly, and I don't trust the numerical quadrature used to compute the error norm very much (because the integrand is getting increasingly spikey near x=1 as N increases - see plots below).

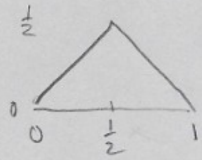




From these plots of the pointwise error and error in the derivative, we can see that the max norm of the value is going to zero as 1/N^2, but the Fourier series is having a bit of trouble with the derivative at the right endpoint, and the max norm of the derivative seems to be going to zero only as 1/N. Though its integral is evidently dropping as 1/N^2.

(a) BVP: $-\big(\underbrace{(1+x)}_{r(x)}y'\big)' = 100, \quad y(0)=y(1)=0.$

$s(x) \equiv 0.$

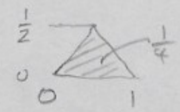
$N=1$ $\quad \phi =$  $, \quad \phi(x) = \begin{cases} x, & x\in[0,\tfrac12] \\ 1-x, & x\in[\tfrac12,1] \end{cases}$

Approx $Y = a\phi$.

Galerkin imposes $\int rY'\phi' + sY\phi = \int f\phi$

or in our case, with $Y' = a\phi'$

$$\int_0^{\frac12}(1+x)\,a\cdot(1)^2\,dx + \int_{\frac12}^1 (1+x)\,a\,(-1)^2\,dx = \int_0^{\frac12} 100x\,dx + \int_{\frac12}^1 100(1-x)\,dx$$



$$a\int_0^1 (1+x)\,dx = 100\cdot\frac14$$

$$a\left[x+\frac{x^2}{2}\right]_0^1 = 25$$

$$a\cdot\frac32 = 25$$

$$\boxed{a = \frac{50}{3}}$$

optimal $\boxed{Y_G(x) = \frac{50}{3}\phi(x)} = \frac{50}{3}\begin{cases} x \\ 1-x \end{cases}$

(b) $y(x) = -100x + \frac{100}{\log 2}\log(1+x), \quad y'(x) = -100 + \frac{100}{\log 2}\cdot\frac{1}{1+x}$

When I assigned this, I was thinking you could do it numerically, but it turned out to be easier to do it exactly by hand, so ...

(c) Exact solution is $y(x) = -100x + \frac{100}{\log 2}\log(1+x)$

Galerkin approx is $Y_G(x) = \frac{100}{6}\begin{cases} x \\ 1-x \end{cases}$

Error for $Y(x) = a\cdot\begin{cases} x \\ 1-x \end{cases}$ is

$$e = \begin{cases} ax + 100x - \frac{100}{\log 2}\log(1+x), & x\in[0,\frac{1}{2}] \\ a(1-x) + 100x - \frac{100}{\log 2}\log(1+x), & x\in(\frac{1}{2},1] \end{cases}$$

$$e' = \begin{cases} a + 100 - \frac{100}{\log 2}\frac{1}{1+x} \\ -a + 100 - \frac{100}{\log 2}\frac{1}{1+x} \end{cases}$$

$$\|e\|^2 = \int r e'^2 + s e^2 = \int_0^1 (1+x)\cdot\begin{Bmatrix} \left(a + 100 - \frac{100}{\log 2}\frac{1}{1+x}\right)^2 \\ \left(-a + 100 - \frac{100}{\log 2}\frac{1}{1+x}\right)^2 \end{Bmatrix} dx$$

$r(x) = 1+x$ $\quad$ $s(x) = 0$

Any validation of $\|e\|^2$ being minimized at $a = \frac{50}{3}$ is OK. I will do it by exact computation...

$$= \int_0^{\frac{1}{2}} (1+x)\cdot\left[a + 100 - \frac{100}{\log 2}\frac{1}{1+x}\right]^2 dx + \int_{\frac{1}{2}}^1 (1+x)\left[-a + 100 - \frac{100}{\log 2}\frac{1}{1+x}\right]^2 dx$$

Find optimal a by differentiating & set = 0!

$$\frac{d}{da}\|e\|^2 = \int_0^{\frac{1}{2}} (1+x)\cdot 2\left[a + 100 - \frac{100}{\log 2}\frac{1}{1+x}\right] dx - \int_{\frac{1}{2}}^1 (1+x)\cdot 2\left[-a + 100 - \frac{100}{\log 2}\frac{1}{1+x}\right] dx$$

$$= 2\int_0^1 (1+x)\cdot a\,dx + 2\int_0^{\frac{1}{2}} 100(1+x)dx - \int_0^{\frac{1}{2}}\frac{100}{\log 2}dx - 2\int_{\frac{1}{2}}^1 100(1+x)dx + \int_{\frac{1}{2}}^1 \frac{100}{\log 2}dx$$

$$= \left[(1+x)^2\cdot a\right]_0^1 + \left[100(1+x)^2\right]_0^{\frac{1}{2}} - \left[100(1+x)^2\right]_{\frac{1}{2}}^1$$

$$= 4a - a + 100\left(\tfrac{3}{2}\right)^2 - 100 - 100\cdot 4 + 100\left(\tfrac{3}{2}\right)^2 \overset{\text{set}}{=} 0 \text{ for extreme}$$

$$= 3a + 225 - 100 - (400 - 225) = 3a - 50$$

$$\Rightarrow a = \frac{50}{3}.$$

3(d) $\|e\|^2 = \int re'^2 + se^2$

This is *not* a norm for arbitrary integrable functions r, s.

For consider $r(x) = x - \frac{1}{2} = s(x)$. 

Then $\|e\| = 0$ for any symmetric (differentiable) function $e(1-x) = e(x)$,

such as $e(x) = x(1-x)$.

A norm of a non-zero function cannot be zero.

If $r(x) \geq c > 0$ and $s(x) \geq 0$, this forces

$\int re'^2 + se^2 > 0$ for any non-zero function e.