Q1. Boundedness of total variation on an interval for two functions

(f_5) $f_5(x) = x - x^2$ on $[0, 1]$

(a) $f_5'(x) = 1 - 2x$ which is a polynomial is continuous.

(b) If a function f is monotonic on an interval [a,b], the total variation on that interval is $| f(b) - f(a) |$.

f_5 is monotonic on both [0,1/2] and [1/2,1]. Therefore its total variation on $[0,1]$ is

$$\left|f(0)-f(\tfrac{1}{2})\right| + \left|f(\tfrac{1}{2})-f(1)\right| = \left|0-\tfrac{1}{4}\right| + \left|\tfrac{1}{4}-0\right| = \tfrac{1}{2}.$$

(The variation wrt any partition that includes 1/2 is 1/4, and for any partition that does not include x=1/2 it will be less than that.)

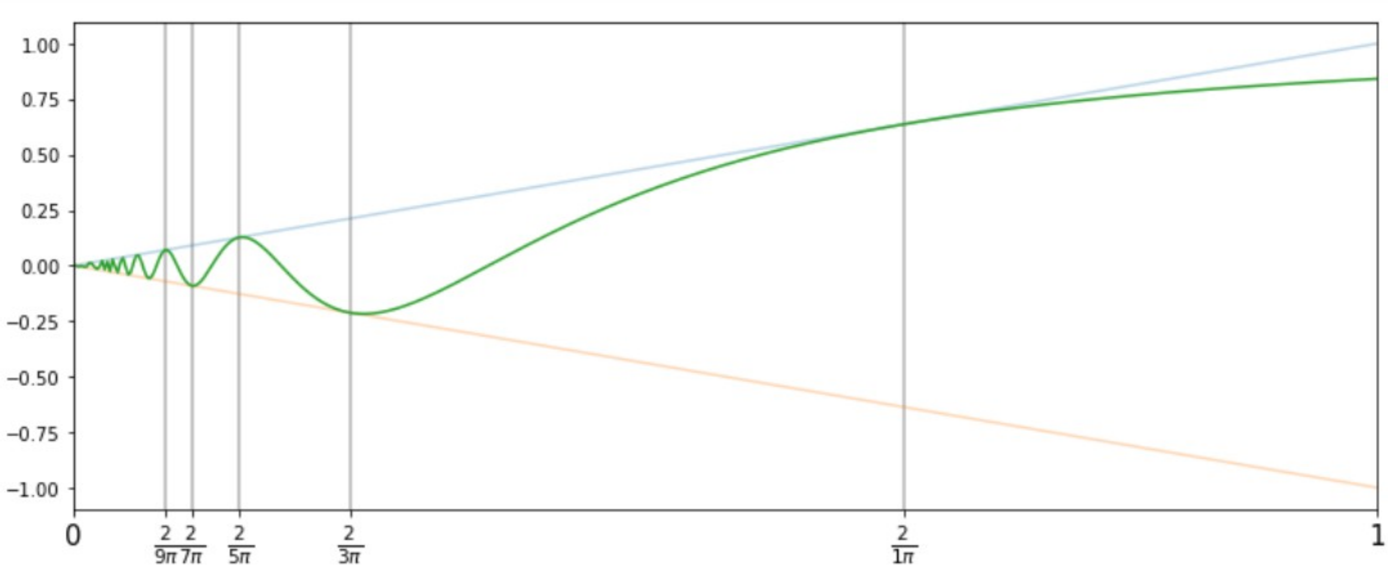
(f_6) $f_6(x) = \begin{cases} x \sin \frac{1}{x}, & x \neq 0 \\ 0, & x < 0 \end{cases}$ on $[0,1]$

(a) For x > 0, f6 as a combination and composition of C1 functions is C1. At x=0, we have

$$f_6'(0) = \lim_{h\to 0} \frac{h \sin \frac{1}{h} - 0}{h} = \lim_{h \to 0} \sin \frac{1}{h}$$

Since sin(1/h) takes on both the values +1 and -1 for arbitrarily small values of h, this limit does not exist. So f6 is not even differentiable at 0.

```
def f6(x): return x*sin(1/x)
plt.figure(figsize=(13,5))
x = np.linspace(1,0,500,endpoint=False)
sin = np.sin
k=5
[plt.axvline(2/(2*n+1)/np.pi,color='k',alpha=0.3) for n in range(k)]
plt.plot(x,x,alpha=0.3)
plt.plot(x,-x,alpha=0.3)
plt.plot(x,f6(x))
plt.xlim(0,1)
plt.xticks([0,1]+[2/(2*n+1)/np.pi for n in range(k)],
           [0,1]+['$\\frac{2}{'+str(2*n+1)+'\pi}$' for n in range(k)],fontsize=15);
```



Consider the partitions using the points $\left\{0, \frac{1}{(4k+1)\frac{\pi}{2}}, \dots, \frac{1}{\frac{3\pi}{2}}, \frac{1}{\frac{\pi}{2}}, 1\right\}$.
$k = 1, 2, \dots$

At these points, f_5 takes the values $\left\{0, +\frac{2}{(4k+1)\pi}, \dots, +\frac{2}{5\pi}, -\frac{2}{3\pi}, \frac{2}{\pi}, \sin 1\right\}$

So the variation wrt this partition is
$$\frac{2}{(4k+1)\pi} + \left|\frac{2}{(4k+1)\pi} + \frac{2}{(4k-1)\pi}\right| + \dots + \left|\frac{2}{5\pi} + \frac{2}{3\pi}\right| + \left|\frac{2}{3\pi} + \frac{2}{\pi}\right| + \left|\frac{2}{\pi} - \sin 1\right|$$

$$t_k = \frac{2}{(4k+1)\pi} + \left|\frac{2}{\pi} - \sin 1\right| + \frac{2}{\pi}\left(1 + \frac{1}{3} + \frac{1}{5} + \dots + \frac{1}{4k+1}\right)$$

We can show that t_k diverges as $k \to \infty$ by supposing $1 + \frac{1}{3} + \frac{1}{5} + \dots$ converges to S:

$$\begin{aligned} S &= 1 + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \frac{1}{9} + \cdots \\ &> 1 + \underbrace{\frac{1}{6} + \frac{1}{6} + \frac{1}{6}}_{\frac{1}{3}} + \underbrace{\frac{1}{10} + \frac{1}{10} + \cdots}_{\frac{1}{5}} \\ &= 1 + \frac{1}{3} + \frac{1}{5} + \cdots = S + \frac{1}{6} \end{aligned}$$

(A contradiction: $S > S + \frac{1}{6}$).

Thus f_6 does not have bounded total variation.

(Consistent with the remark at the very bottom of p250.)

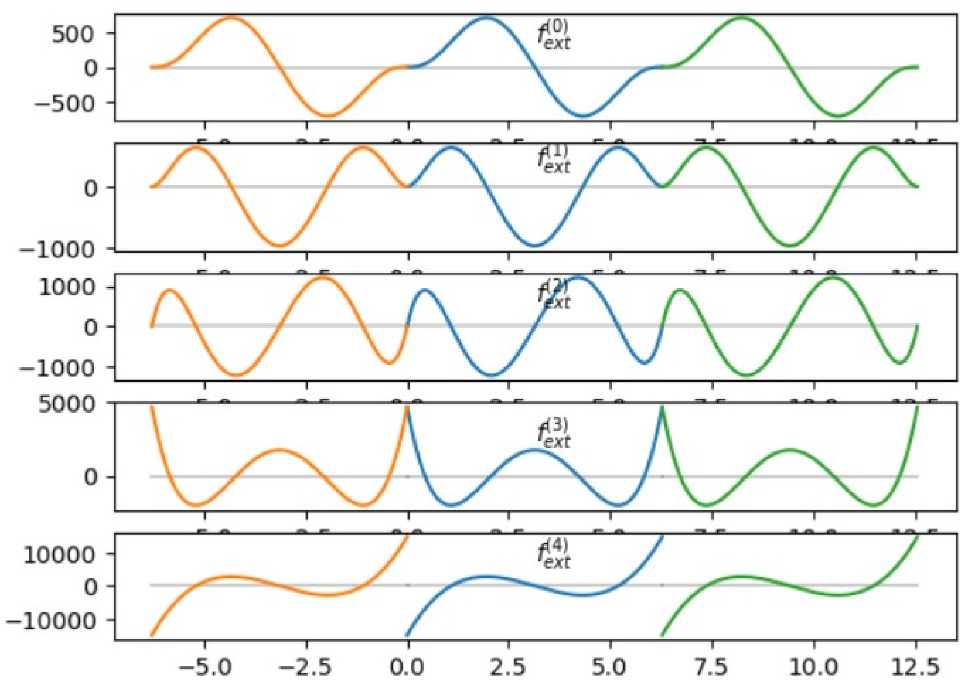
Q2

After a little experimentation, I came up with the function

$$f(x) = x^3(x-2\pi)^3(x-\pi).$$

Below I plot part of the periodic extension of f and its first 4 derivatives, where we see that the periodic extension is 3 times continuously differentiable, but not 4 times.

```python
from nsm import *

x = sp.symbols('x')
a,b = 0, 2*np.pi

f = ((x-a)*(x-b))**3 *(x-(a+b)/2) # (x-a)**4*(x-b)**5 #

xx = np.linspace(a,b,400)

n = 4
for j in range(n+1):
    fj = sp.lambdify(x,sp.diff(f,x,j),'numpy')  # jth derivative of f
    plt.subplot(n+1,1,j+1)
    for k in [-1,0,1]: plt.plot(xx+k*(b-a),0*xx,'k',alpha=0.2)
    plt.plot(xx,fj(xx))        # jth derivative of f
    plt.plot(xx-(b-a),fj(xx))  # part of its periodic extension
    plt.plot(xx+(b-a),fj(xx))  # another part of its periodic extension
    plt.text((a+b)/2,fj(xx).max()/2,'$f_{ext}^{('+str(j)+')}$')

```



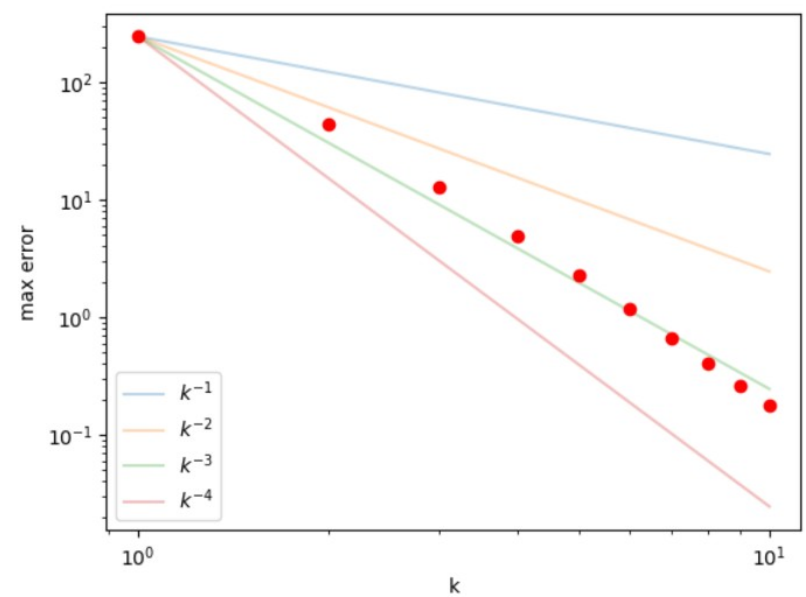
Q2 cont'd

Error in least-squares trigonometric polynomial approximations

as a function of dimension of approximating subspace

```
from scipy.integrate import quadrature

twopi = 2*np.pi
a,b = 0, twopi

def f(x): return ((x-a)*(x-b))**3 *(x-(a+b)/2)

i = 1j
x = np.linspace(0,twopi,1000)
kstop = 11
#plt.figure(figsize=(8,16))
kk = np.linspace(1,kstop-1,3)
for k in range(1,kstop):
    gk = np.zeros_like(x,dtype=complex)
    alpha = []
    for j in range(-k,k+1):
        wj = np.exp(i*j*x)
        def integrand(x): return f(x)*np.exp(-i*j*x)
        twopiaj,err = quadrature(integrand,0,twopi,rtol=1e-11,tol=1e-11)   # gaussian quadrature
        alphaj = twopiaj/twopi
        alpha.append(alphaj)
        gk += alphaj*wj

    maxerror = np.abs(f(x)-gk).max()
    print(k,maxerror)
    if k==1:
        for p in range(1,5):
            plt.loglog(kk,kk**(-p)*maxerror,label='$k^{-'+str(p)+'}$',alpha=0.3)

    plt.loglog(k,maxerror,'ro')
plt.xlabel('k'); plt.ylabel('max error')
plt.legend();
```

```
1 244.3744552822887
2 44.349514366548405
3 12.804661413783588
4 4.900059879692572
5 2.2471782423128257
6 1.1676070762817972
7 0.6643863804160324
8 0.4049384041775457
9 0.26062550911272964
10 0.175112201108280
```



Observations: For sufficiently large k ($k \geq 2$, perhaps), the slope of the measured errors (red dots) on the log-log plot is at least as steep as -3, as Thm 4.17 guarantees since "$n-1$" = 3.

In fact, and unexpectedly, it looks like the slope may be as steep as -4.

I think I once glimpsed what the proof of the theorem misses, but I can't find my notes on it and don't recall what it was. 😅

Q3

(a) $n=4, \quad \omega = e^{-i\frac{2\pi}{4}} = e^{-i\frac{\pi}{2}} = -i$

$$W_4 = \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & (-i)^2 & (-i)^3 \\ 1 & (-i)^2 & (-i)^4 & (-i)^6 \\ 1 & (-i)^3 & (-i)^6 & (-i)^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

So

$$F_4 = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

(b) $$F_4^{-1} = W^* = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

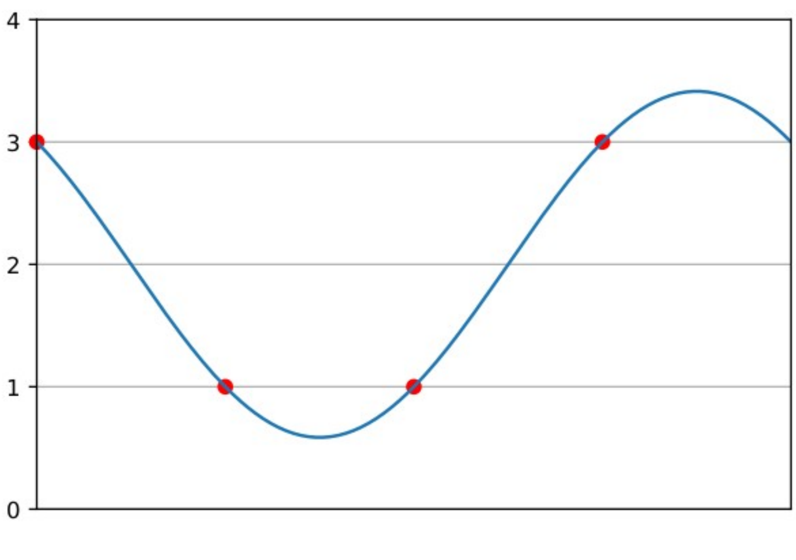
(c) $$F_4 y = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}\begin{bmatrix} 3 \\ 1 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ \frac{1}{2}+\frac{i}{2} \\ 0 \\ \frac{1}{2}-\frac{i}{2} \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

(d) $$p(x) = {\sum_{k=-\frac{n}{2}}^{\frac{n}{2}}}' c_{k \bmod n}\, e^{ik2\pi\frac{x}{L}}$$ where $\sum'$ means take only half of the 1st & last terms.

$$= \frac{1}{2}\,0\,e^{i(-2)2\pi\frac{x}{L}} + \left(\frac{1}{2}-\frac{i}{2}\right)e^{i(-1)2\pi\frac{x}{L}} + (2)\,e^0 + \left(\frac{1}{2}+\frac{i}{2}\right)e^{i(1)2\pi\frac{x}{L}} + \frac{1}{2}\,0\,e^{i(2)2\pi\frac{x}{L}}$$

$$= 0 + 2 + \frac{1}{2}\left(e^{2\pi i x/L} + e^{-2\pi i x/L}\right) + \frac{i}{2}\left(e^{2\pi i x/L} - e^{-2\pi i x/L}\right)$$

$$= 2 + \cos\frac{2\pi x}{L} - \sin\frac{2\pi x}{L}$$




```python
L = 7.7  # arbitrary range of x axis
y = np.array([3,1,1,3])
n = len(y)
x = np.linspace(0,L,n,endpoint=False)
xx = np.linspace(0,L,201)
twopi = 2*np.pi
p = 2 + np.cos(twopi*xx/L) - np.sin(twopi*xx/L)
plt.plot(x,y,'ro',clip_on=False)
plt.plot(xx,p)
plt.xlim(0,L); plt.ylim(0,4)
plt.xticks([]); plt.yticks([0,1,2,3,4]); plt.grid()
plt.savefig('temp.pdf');
```

Q4. Solve the recurrence relation $C_n = 2C_{n/2} + 3n + 1$, $C_1 = 0$.

Not-quite-right guess: $C_{2^j} = 3j2^j$.

Let's see how much it's off by:

j	2^j	actual C_{2^j} from r.r.	my guess $3j2^j$	my guess' deficit	$2^j - 1$ looks like
0	1	0	0	0	$1-1$
1	2	7	6	1	$2-1$
2	4	27	24	3	$4-1$
3	8	79	72	7	$8-1$
4	16	207	192	15	$16-1$
5	32	511	480	31	$32-1$

revised guess $\boxed{3j2^j + 2^j - 1} = G_{2^j}$ (G for guess)

This revised guess gets the first 6 values correct, but to prove that it gets <u>all</u> of them correct, we must show it satisfies the recurrence relation

$$C_n = 2C_{n/2} + 3n + 1$$

Let $m = 2^i$, $\frac{m}{2} = 2^{i-1}$.

$$G_{\frac{m}{2}} = G_{2^{i-1}} = 3(i-1)2^{i-1} + 2^{i-1} - 1$$

and

$$
\begin{aligned}
2G_{\frac{m}{2}} + 3m + 1 &= 2\left(3(i-1)2^{i-1} + 2^{i-1} - 1\right) + 3 \cdot 2^i + 1 \\
&= 3i2^i - \cancel{3 \cdot 2^i} + 2^i - 2 + \cancel{3 \cdot 2^i} + 1 \\
&= 3i2^i + 2^i - 1 \\
&= G_m .
\end{aligned}
$$

Thus by induction, G_{2^j} is correct for all $j = 0, 1, 2, \ldots$

Q4(b)

$$C_{2^j} = 3j2^j + 2^j - 1$$

With $2^j = n$, $j\log 2 = \log n$, $j = \frac{\log n}{\log 2}$, this becomes

$$C_n = 3\frac{\log n}{\log 2}\cdot n + n - 1 = \frac{3}{\log 2}\cdot n\log n + n - 1$$

$$= O(n\log n)$$

The leading-order term in the cost is $\sim n\log n$.
This grows slowly with n compared to n^2 which is the cost of brute-force matrix multiplication to perform the DFT.

```python
from nsm import *
plt.figure(figsize=(10,10))
nmax=500
n = np.arange(1,nmax)
plt.plot(n,2*n**2,lw=3,alpha=0.5,label='$2n^2$')
plt.plot(n,3*n*np.log2(n),lw=3,alpha=0.5,label='$3\ n\ \log_2\ n$')
plt.plot(n,10*n,lw=3,alpha=0.5,label=f'$10 n$')
plt.xlabel('$n$',fontsize=20)
plt.ylim(0,2*nmax**2)
plt.xlim(0,nmax)
plt.legend(fontsize=20);
```