Day 16 Approximation with trigonometric functions

1. "Best" approximation of f in vector space V = C[0,2pi] with inner product $(f,g) = \int f\bar{g} = \int_0^{2\pi} f(x)\bar{g}(x)\,dx$

2. Interpolation

1.

For f in V, approximate it by an element of

$W_{2k} = \text{span}\{w_{-k}, \dots, w_{-1}, w_0, w_1, \dots w_k\}$

where $w_j(x) = e^{ijx} = (e^{ix})^j$ where $i = \sqrt{-1}$, $j = -k, \dots, 0, \dots k$

{w_j} is a basis for its span because the elements are orthogonal w.r.t. the inner product, hence linearly independent.

Prop: $(w_j, w_\ell) = 2\pi\delta_{j\ell}$

Proof: $(w_j, w_\ell) = \int_0^{2\pi} e^{ijx}e^{-i\ell x}\,dx = \int_0^{2\pi} e^{i(j-\ell)x}\,dx$

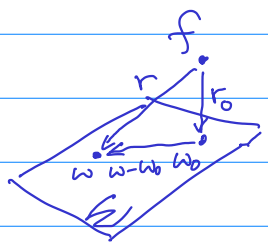
$$= \begin{cases} \int_0^{2\pi} e^0\,dx = \int_0^{2\pi} 1\,dx = 2\pi, & j=\ell \\ \int_0^{2\pi} \cos(j-\ell)x\,dx + i\int_0^{2\pi} \sin(j-\ell)x\,dx = 0+0 = 0 & j\neq\ell \end{cases}$$ ☺

The "best" approximation to f in the subspace, i.e. the one with the smallest error is one whose error is orthogonal to the subspace.

Proof: $W \subset V$, $f \in V$, $w_0 = Tw$, $w = Tw$

Set $r_0 = w_0 - f$, $r = w - f$

Let r_0 be orthogonal to all vectors $w \in W$. *



Can write $r = r_0 + (w - w_0)$

$$\|r\|^2 = (r,r)$$
$$= (r_0 + (w-w_0), r_0 + (w-w_0))$$
$$= (r_0, r_0) + (w-w_0, r_0) + (r_0, w-w_0) + (w-w_0, w-w_0)$$
$$= \|r_0\|^2 + \|w-w_0\|^2$$

This is minimized if $w = w_0$.

Best approximation $g_{2\kappa} \in W_{2\kappa}$ for $f \in V$ is an element of $W_{2\kappa}$ such that

$$(g_{2\kappa} - f, w_\ell) = 0 \quad \forall \ell \in -\kappa, \dots, 0, \dots, \kappa$$

or

$$* \quad (g_{2\kappa}, w_\ell) = (f, w_\ell) \quad \forall \ell.$$

Writing $\left[g_{2\kappa}(x) = \sum_{j=-\kappa}^{\kappa} \alpha_j w_j(x) \right]$ (plug into *), we can readily find the $\{\alpha_j\}$ ...

$$\left(\sum_j \alpha_j w_j, w_\ell \right) \overset{\text{want}}{=} \boxed{(f, w_\ell)}$$

$$= \sum_j \alpha_j (w_j, w_\ell) = \alpha_j 2\pi \delta_{j\ell} = \boxed{2\pi \alpha_\ell}$$

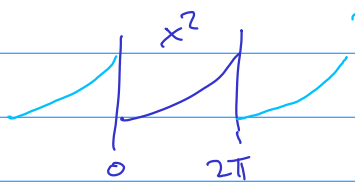
$$\rightarrow \boxed{\alpha_\ell = \frac{1}{2\pi}(f, w_\ell)} \; *$$

With this formula for the $\{\alpha_j\}$, we define

Def 4.18 $$g_\infty(x) = \sum_{j=-\infty}^{\infty} \alpha_j e^{ijx}$$

and call it the complex Fourier series of f,

Examples:



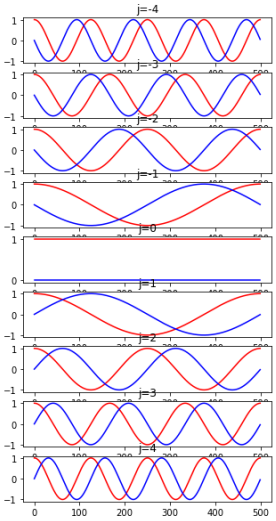
periodic extension of x^2 to $\mathbb{R}$ is not continuous.

If periodic extension of f is continuous and even better, smooth, then we expect better results.

4.5.1 Least squares trigonometric approximation

A subspace of C[0,2π] spanned by complex exponentials

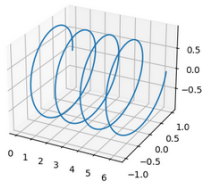
```
%matplotlib inline
from nsm import *
plt.figure(figsize=(5,10))
k = 4
x = np.linspace(0,2*np.pi,500,endpoint=False)
w = np.exp(1j*x)
for j in range(-k,k+1):
    plt.subplot(2*k+1,1,j+k+1)
    plt.plot(np.real(w**j),'r')          # real part shown in red
    plt.plot(np.imag(w**j),'b')          # imag part shown in blue
    plt.title(f'j={j}')
```




```
%matplotlib notebook
ax = plt.figure().add_subplot(projection='3d')
k = 4
x = np.linspace(0,2*np.pi,500,endpoint=False)
w = np.exp(1j*x)
for j in [k]:
    plt.plot(x,np.real(w**j),np.imag(w**j));
```

```
<IPython.core.display.Javascript object>
```

Plotted in Cx[0,2pi]



These are the functions we are building our approximation with.

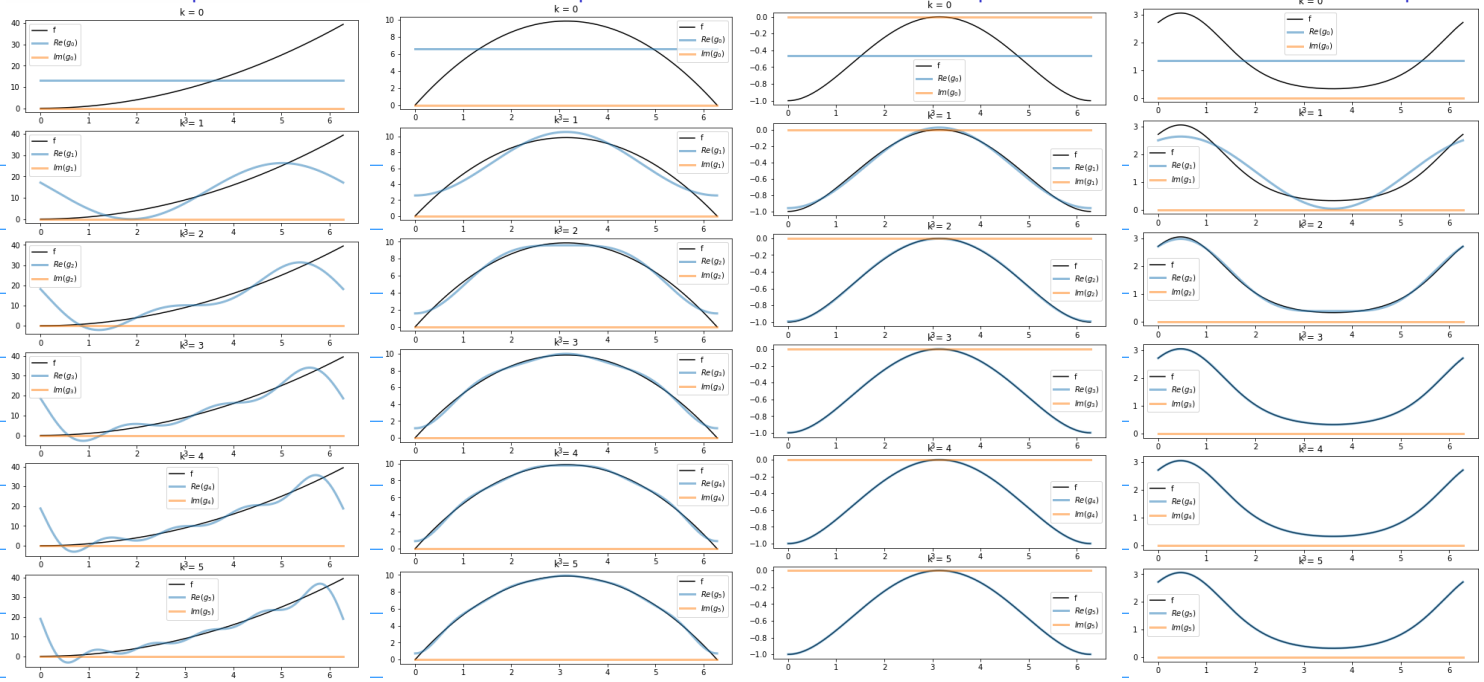
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import quadrature
```

```
%matplotlib inline
twopi = 2*np.pi
i = 1j
def f1(x): return np.exp(np.cos(x)+np.sin(x)/2)  # a smooth periodic function that is not a combination of trig funct
def f2(x): return x**2  # a function with discontinuous periodic extension
def f3(x): return x*(twopi-x)  # a function with continuous periodic extension but jump in derivative
def f4(x): x1 = (x-np.pi)/np.pi; return x1**4 - 2*x1**2

f = f2

x = np.linspace(0,twopi,1000)
kstop = 6
plt.figure(figsize=(8,16))
for k in range(kstop):
    plt.subplot(kstop,1,k+1)
    plt.plot(x,f(x),'k',label='f');
    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)   # gaussian quadrature
        alphaj = twopiaj/twopi
        alpha.append(alphaj)
        gk += alphaj*wj
    plt.plot(x,np.real(gk),label=f'$Re(g_{k})$',lw=3,alpha=0.5)
    plt.plot(x,np.imag(gk),label=f'$Im(g_{k})$',lw=3,alpha=0.5)
    plt.title(f'k = {k}')
    #print(np.round(a,4))
    plt.legend()
```

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import quadrature

%matplotlib inline
twopi = 2*np.pi
i = 1j
def f1(x): return np.exp(np.cos(x)+np.sin(x)/2)  # a smooth periodic function that is not a combi
def f2(x): return x**2  # a function with discontinuous periodic extension
def f3(x): return x*(twopi-x)  # a function with continuous periodic extension but jump in deriva
def f4(x): x1 = (x-np.pi)/np.pi; return x1**4 - 2*x1**2

f = f2

x = np.linspace(0,twopi,1000)
kstop = 6
plt.figure(figsize=(8,16))
for k in range(kstop):
    plt.subplot(kstop,1,k+1)
    plt.plot(x,f(x),'k',label='f');
    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)   # gaussian quadrature
        alphaj = twopiaj/twopi
        alpha.append(alphaj)
        gk += alphaj*wj
    plt.plot(x,np.real(gk),label=f'$Re(g_{k})$',lw=3,alpha=0.5)
    plt.plot(x,np.imag(gk),label=f'$Im(g_{k})$',lw=3,alpha=0.5)
    plt.title(f'k = {k}')
    #print(np.round(a,4))
    plt.legend()
```

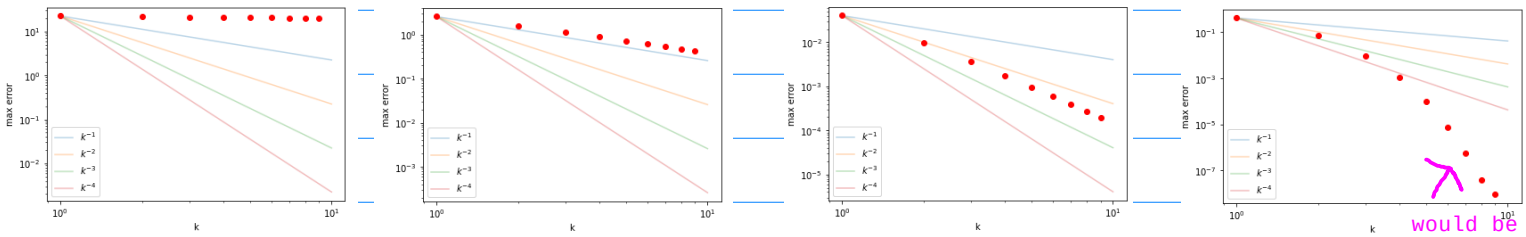
Convergence as k increased in the 4 examples above is illustrated on the next page.

f2 discontinuous periodic extension | f3 C^0 per. ext. | f4 C^3 per. ext. | f1 C^inf per. ext.



Rate of convergence: infinity norm as a power of 1/k as slope of log-log plot

```
f=f1
x = np.linspace(0,twopi,1000)
kstop = 10
#plt.figure(figsize=(8,16))
kk = np.linspace(1,kstop,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)   # 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();
```



would be steeper here if my quadrature for the alphas wer more accurate.

In agreement with empirical observations just made, we have this

Thm 4.1&7 (Proof in next Homework)

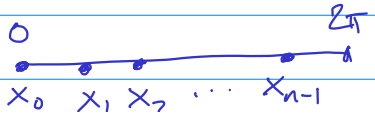
If f is continuous on $[0,2\pi]$ and $f^{(p)}(0) = f^{(p)}(2\pi)$ for all $p = 0, 1, \ldots, n-1$ then $\|f - g_{2k}\|_\infty = O\left(\frac{1}{k^{n-1}}\right)$.

And if $f \in C^\infty(\mathbb{R})$ then $\|f - g_{2k}\|_\infty \to 0$ faster than any power of $\frac{1}{k}$.

("spectral" convergence)

2. Interpolation with trig polynomials

Define $x_k = \frac{2\pi k}{n}$, $k = 0, 1, \ldots, n-1$



Thm 4.18 Define $p(x) = \overset{①}{\sum_{j=0}^{N-1}} c_j e^{ijx}$, $c_j = \overset{②}{\frac{1}{N}\sum_{k=0}^{N-1}} f(x_k) e^{-ijx_k}$, $x_k = \overset{③}{\frac{2\pi k}{N}}$.

$p(x_\nu) = f(x_\nu)$, $\nu = 0, \ldots, N-1$. I.e. p interpolates the data.

Proof.

$$\boxed{p(x_\nu)} \overset{①}{=} \sum_{j=0}^{N-1} c_j e^{ijx_\nu} \overset{②}{=} \frac{1}{N}\sum_{j=0}^{N-1}\sum_{k=0}^{N-1} f(x_k) e^{-ijx_k} e^{ijx_\nu}$$

$$= \sum_{k=0}^{N-1} f(x_k) \frac{1}{N}\sum_{j=0}^{N-1} e^{-ijx_k} e^{ijx_\nu} \quad \text{(re-ordering sums)}$$

$$\overset{③}{=} \sum_{k=0}^{N-1} f(x_k) \left[\frac{1}{N}\sum_{j=0}^{N-1} e^{-ijk\frac{2\pi}{N}} e^{ij\nu\frac{2\pi}{N}}\right]$$

$$= \sum_{k=0}^{N-1} f(x_k) \left[\frac{1}{N}\sum_{j=0}^{N-1} e^{-ikx_j} e^{i\nu x_j}\right]$$

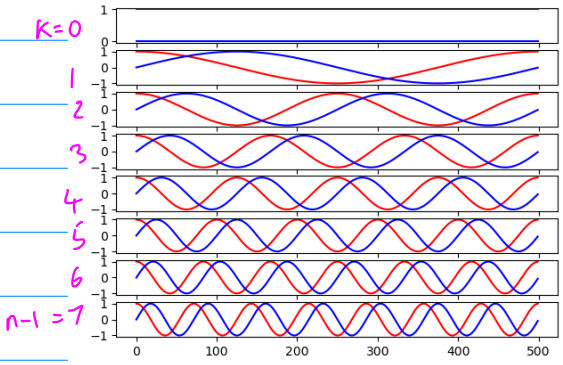
$$= \sum_{k=0}^{N-1} f(x_k) \left[\frac{1}{N} N\delta_{k\nu}\right]$$

$$= \boxed{f(x_\nu)}. \quad ☺$$

4.5.2 Discrete Fourier transform

```
from nsm import *   # numpy as np, sympy as sp, matplotlib.pyplot as plt

n = 8
x = np.linspace(0,2*np.pi,500,endpoint=False)
w = np.exp(1j*x)
for j in range(n):
    plt.subplot(n,1,j+1)
    plt.plot(np.real(w**j),'r')
    plt.plot(np.imag(w**j),'b')
```

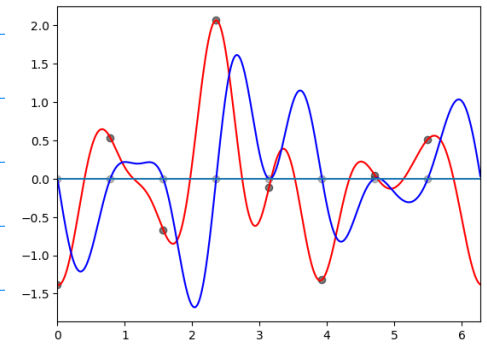
The functions exp(ijx) for j=0,1,...n-1



red real part
blue imag part

```
y = np.random.randn(n)
x = np.linspace(0,2*np.pi,n,endpoint=False)
plt.plot(x,y,'ko',alpha=0.5)
plt.plot(x,y*0,'ko',alpha=0.25)
nn = 1000
xx = np.linspace(0,2*np.pi,nn,endpoint=False)
ww = np.exp(1j*xx)
c = A@y
#print(c)
p = np.zeros(nn,dtype=complex)
for j in range(n):
    print(c[j])
    p += c[j]*ww**j
plt.plot(xx,np.real(p),'r')
plt.plot(xx,np.imag(p),'b')
plt.axhline(0)
plt.xlim(0,2*np.pi)
```

```
(-0.04019157128643404+0j)
(-0.1328542936776702-0.21156295561908092j)
(-0.10867570278701828+0.42071622935739034j)
(-0.18608452827529598-0.3901145891202525j)
(-0.4890368466098044-2.909595696206072e-16j)
(-0.18608452827529676+0.3901145891202522j)
(-0.10867570278701653-0.42071622935739032j)
(-0.1328542936776707+0.21156295561908128j)

(0.0, 6.283185307179586)
```



This is indeed an interpolant,
but it leaves a lot to be desired!
Unwarranted wiggles, and strange imaginary part
for data that is real.

We will construct a better one next class (Day 17).