# Chapter 6. Inner Product and Fourier Expansion

## 6.1. Norm and inner product

Untitled.ipynb

In [1]:
```
from numpy import array, dot, inner, conj, vdot
x, y = array([1+2j, 3+4j]), array([4+3j, 2+1j]); x, y
```

Out[1]:
```
(array([1.+2.j, 3.+4.j]), array([4.+3.j, 2.+1.j]))
```

In [2]:
```
conj(x).dot(y), dot(x.conj(), y), inner(conj(x), y)
```

Out[2]:
```
((20-10j), (20-10j), (20-10j))
```

In [3]:
```
vdot(x, y), vdot(1j*x, y), vdot(x, 1j*y)
```

Out[3]:
```
((20-10j), (-10-20j), (10+20j))
```

In [4]:
```
from numpy.linalg import norm
norm(x), norm(y)
```

Out[4]:
```
(5.477225575051661, 5.477225575051661)
```

In [5]:
```
norm([1, 2, 3], ord=1)
```

Out[5]:
```
6.0
```

In [6]:
```
norm([1, 2, 3], ord=2)
```

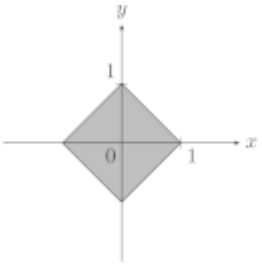
Out[6]:
```
3.7416573867739413
```

In [7]:
```
from numpy import inf
norm([1, 2, 3], ord=inf)
```

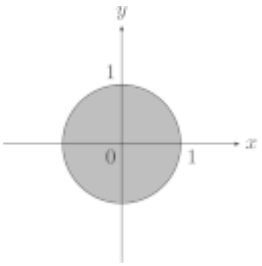
Out[7]:
```
3.0
```

---

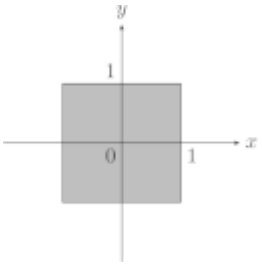
**Program:** norm1.py




**Program:** norm2.py

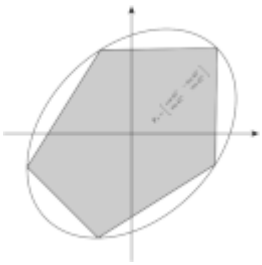



**Program:** normoo.py




In the above, we use the following module which is a wrapper of the library PyX.

**Program:** mypyx.py

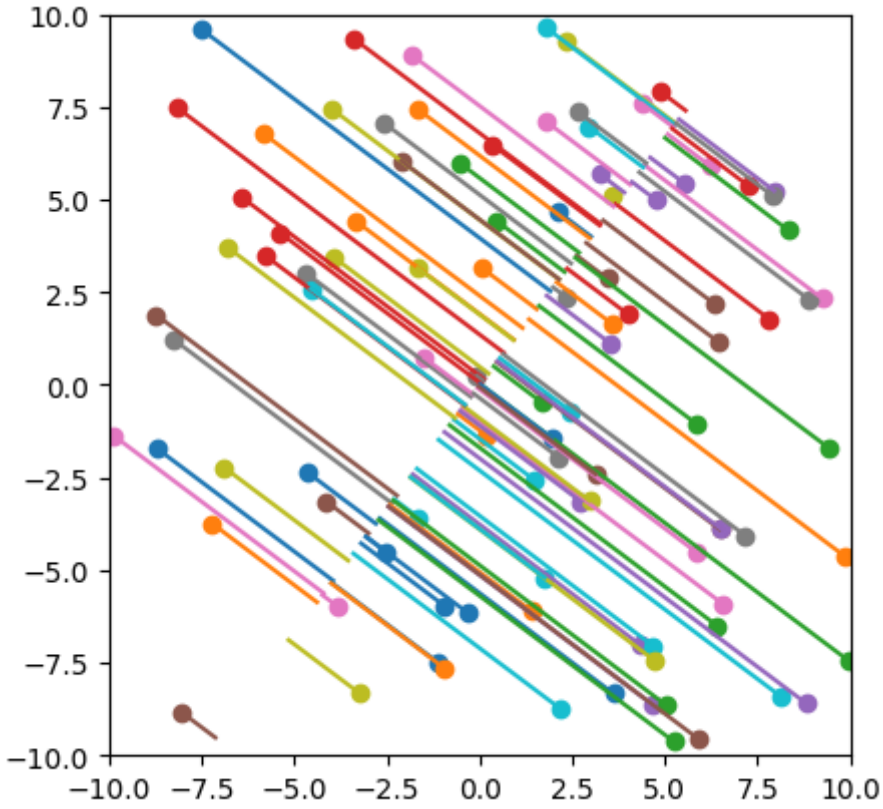


---

# 6.2. Orthonormal systems and Fourier transform

**Program:** proj.py

In [1]:
```
from numpy import random, array, inner, sqrt, dot
import matplotlib.pyplot as plt

random.seed(2021)
v = array([3, 4])
e =  v / sqrt(inner(v, v))
plt.axis('scaled'), plt.xlim(-10, 10), plt.ylim(-10, 10)
for n in range(100):
    v = random.uniform(-10, 10, 2)
    plt.scatter(*v)
    w = inner(e, v) * e
    #P = dot(e.reshape((2, 1)), e.reshape((1, 2)))
    #w = dot(P, v)
    plt.plot(*zip(v, w))
```




---

**Program:** gram_schmidt.py

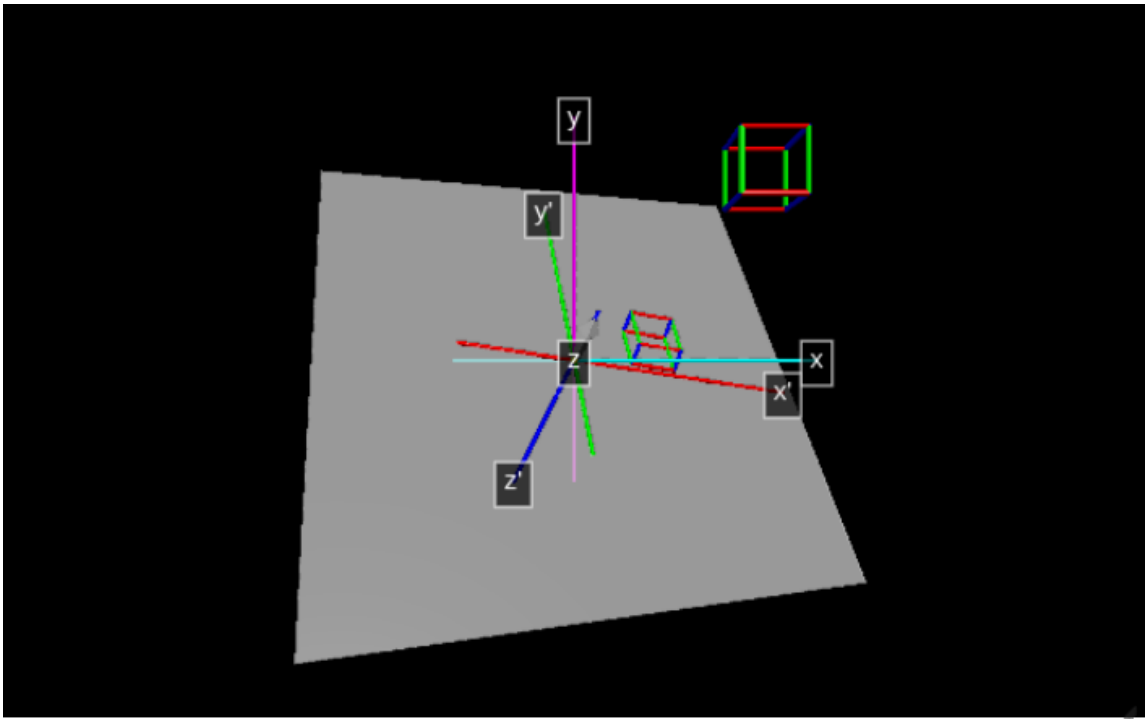
In [1]:
```
from numpy import array, vdot, sqrt


def proj(x, E, inner=vdot):
    return sum([inner(e, x) * e for e in E])


def gram_schmidt(A, inner=vdot):
    E = []
    while A != []:
        a = array(A.pop(0))
        b = a - proj(a, E, inner)
        c = sqrt(inner(b, b))
        if c >= 1.0e-15:
            E.append(b / c)
    return E


if __name__ == '__main__':
    A = [[1, 2, 3], [2, 3, 4], [3, 4, 5]]
    E = gram_schmidt(A)
    for n, e in enumerate(E):
        print(f'e{n+1} = {e}')
    print(array([[vdot(e1, e2) for e2 in E] for e1 in E]))
```

```
e1 = [0.26726124 0.53452248 0.80178373]
e2 = [ 0.87287156  0.21821789 -0.43643578]
[[1.00000000e+00 6.51017134e-17]
 [6.51017134e-17 1.00000000e+00]]
```

---

**Program:**proj2d.py

In [1]:
```
from vpython import proj, color, curve, vec, hat, box, arrow, label


def proj2d(x, e): return x - proj(x, e)


def draw_cube(o, x, y, z):
    axes, cols = [x, y, z], [color.red, color.green, color.blue]
    planes = [(y, z), (z, x), (x, y)]
    for ax, col, p in zip(axes, cols, planes):
        face = [o, o + p[0], o + p[0] + p[1], o + p[1]]
        for vertex in face:
            curve(pos=[vertex, vertex + ax], color=col)


o, e, u = vec(0, 0, 0), hat(vec(1, 2, 3)), vec(5, 5, 5)
x, y, z = vec(1, 0, 0), vec(0, 1, 0), vec(0, 0, 1)
box(pos=-0.1 * e, axis=e, up=proj2d(y, e),
    width=20, height=20, length=0.1, opacity=0.5)
arrow(axis=3 * e)
for ax, lbl in [(x, 'x'), (y, 'y'), (z, 'z')]:
    curve(pos=[-5 * ax, 10 * ax], color=vec(1, 1, 1) - ax)
    label(pos=10 * ax, text=lbl)
    curve(pos=[proj2d(-5 * ax, e), proj2d(10 * ax, e)], color=ax)
    label(pos=proj2d(10 * ax, e), text=f"{lbl}'")
c1 = [u, 2*x, 2*y, 2*z]
c2 = [proj2d(v, e) for v in c1]
draw_cube(*c1), draw_cube(*c2)
```

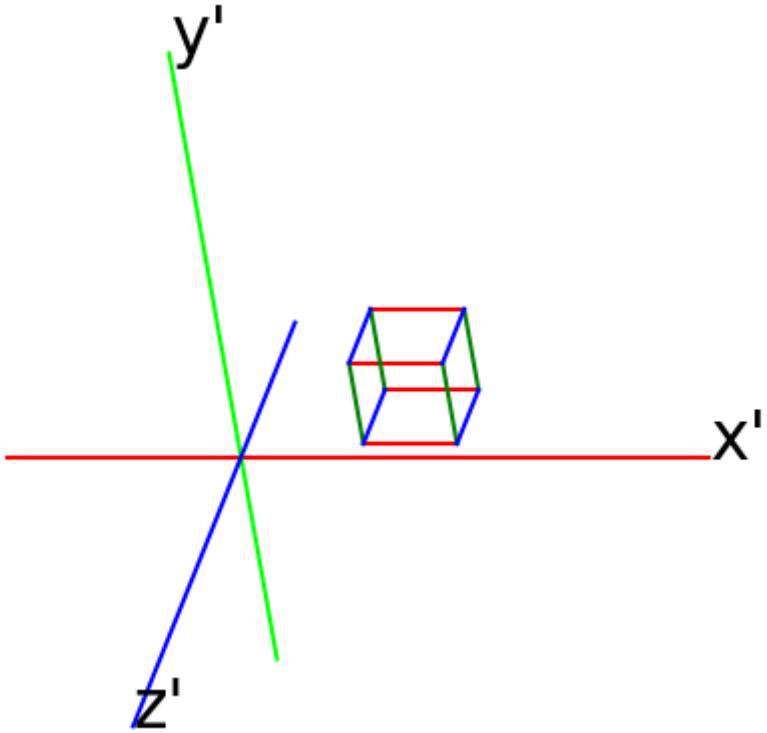



---

**Program:** screen.py

In [1]:
```
from numpy import array
```

```
import matplotlib.pyplot as plt
from gram_schmidt import gram_schmidt


def curve(pos, color=(0, 0, 0)):
    plt.plot(*zip(*pos), color=color)


def draw_cube(o, x, y, z):
    axes, cols = [x, y, z], ['r', 'g', 'b']
    planes = [(y, z), (z, x), (x, y)]
    for ax, col, p in zip(axes, cols, planes):
        face = [o, o + p[0], o + p[0] + p[1], o + p[1]]
        for vertex in face:
            curve(pos=[vertex, vertex + ax], color=col)


o, e, u = array([0, 0, 0]), array([1, 2, 3]), array([5, 5, 5])
x, y, z = array([1, 0, 0]), array([0, 1, 0]), array([0, 0, 1])
E = array(gram_schmidt([e, x, y, z])[1:])
plt.axis('equal'), plt.axis('off')
for ax, lbl in [(x, "x'"), (y, "y'"), (z, "z'")]:
    curve(pos=[E.dot(-5 * ax), E.dot(10 * ax)], color=ax)
    plt.text(*E.dot(10 * ax), lbl, fontsize=24)
c1 = [u, 2*x, 2*y, 2*z]
c2 = [E.dot(v) for v in c1]
draw_cube(*c2)
```




## 6.3. Function space

**Program:** integral.py

In [1]:
```
from numpy import array, sqrt
```

```
def integral(f, dom):
    N = len(dom) - 1
    w = (dom[-1] - dom[0]) / N
    x = array([(dom[n] + dom[n + 1]) / 2 for n in range(N)])
    return sum(f(x)) * w


def inner(f, g, dom):
    return integral(lambda x: f(x).conj() * g(x), dom)


norm = {
    'L1': lambda f, dom: integral(lambda x: abs(f(x)), dom),
    'L2': lambda f, dom: sqrt(inner(f, f, dom)),
    'Loo': lambda f, dom: max(abs(f(dom))),
}

if __name__ == '__main__':
    from numpy import linspace, pi, sin, cos
    dom = linspace(0, pi, 1001)
    print(f'<sin|cos> = {inner(sin, cos, dom)}')
    print(f'||f_1||_2 = {norm["L2"](lambda x: x, dom)}')
```

```
<sin|cos> = -3.6738427362813575e-18
||f_1||_2 = 3.214875266047432
```

---

Untitled.ipynb

In [1]:
```
from sympy import integrate, pi, sin, cos, sqrt
from sympy.abc import x
integrate('sin(x) * cos(x)', [x, 0, pi])
```

Out[1]: $0$

In [2]:
```
sqrt(integrate('x**2', [x, 0, pi]))
```

Out[2]: $\frac{\sqrt{3}\pi^{\frac{3}{2}}}{3}$

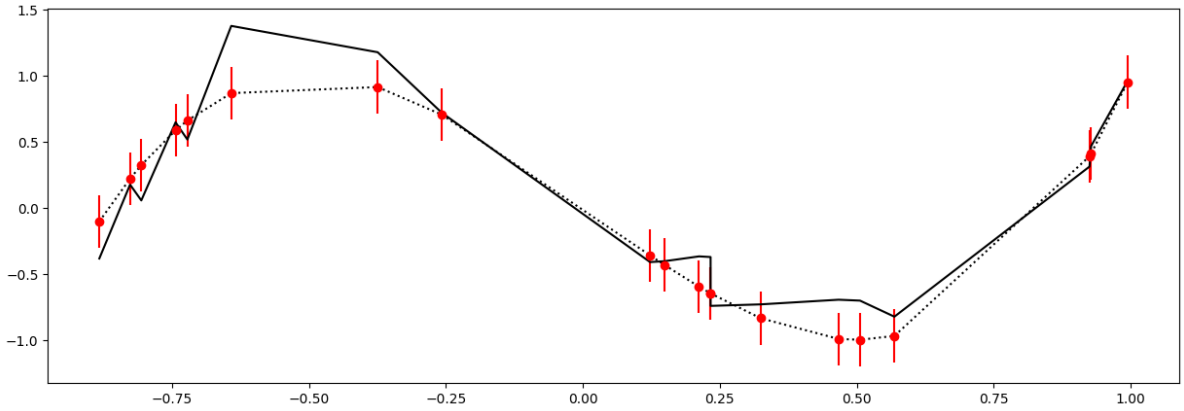
## 6.4. Least squares, Trigonometric series and Fourier series

**Program:** lstsqr.py

In [1]:
```
from numpy import linspace, cumsum, vdot, sort
from numpy.random import seed, uniform, normal
import matplotlib.pyplot as plt
from gram_schmidt import gram_schmidt, proj

n = 20
seed(2021)
x = sort(uniform(-1, 1, n))
z = 4 * x**3 - 3 * x
sigma = 0.2
y = z + normal(0, sigma, n)
E = gram_schmidt([x**0, x**1, x**2, x**3])
y0 = proj(z, E)
```

```
plt.figure(figsize=(15,5))
plt.errorbar(x, z, yerr=sigma, fmt='ro')
plt.plot(x, y, color='k', linestyle = 'solid')
plt.plot(x, y0, color='k', linestyle = 'dotted')
```

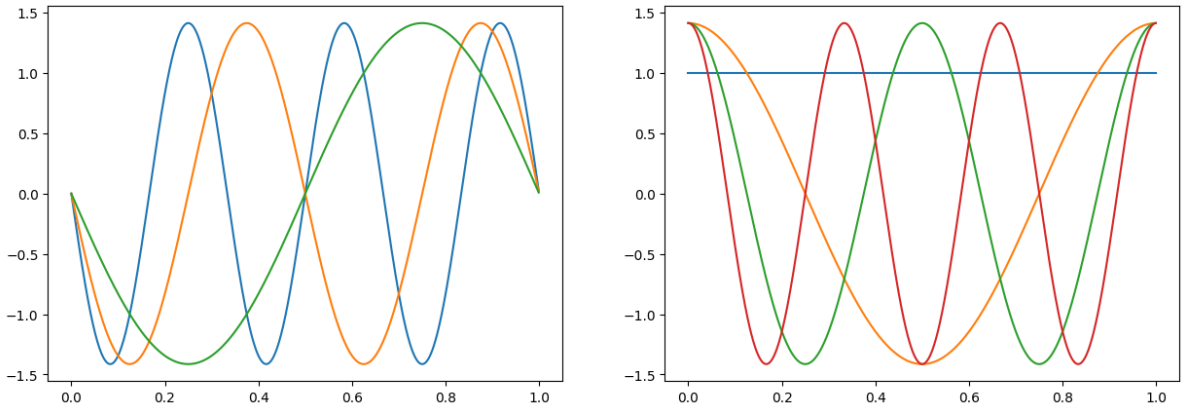
Out[1]: [<matplotlib.lines.Line2D at 0x7f6df5fdc0>]




---

**Program:**trigonometric.py

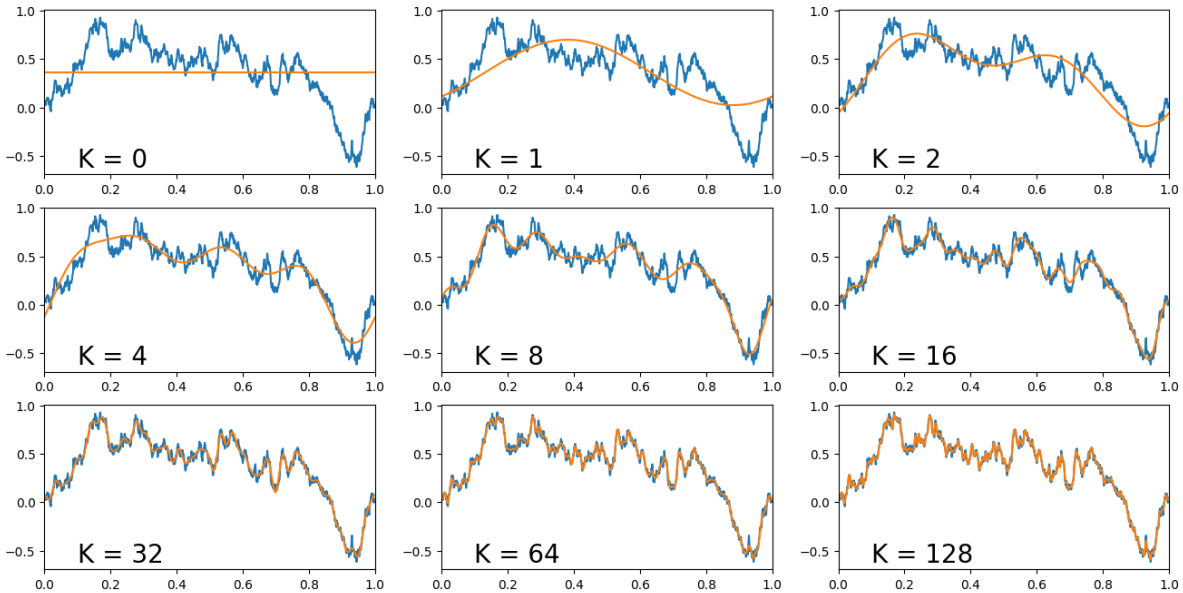
In [1]:
```
from numpy import inner, pi, sin, cos, sqrt, ones
from gram_schmidt import proj


def e(k, t):
    if k < 0:
        return sin(2 * k * pi * t) * sqrt(2)
    elif k == 0:
        return ones(len(t))
    elif k > 0:
        return cos(2 * k * pi * t) * sqrt(2)


def lowpass(K, t, f):
    n = len(t)
    E_K = [e(k, t) for k in range(-K, K + 1)]
    return proj(f, E_K, inner=lambda x, y: inner(x, y) / n)


if __name__ == '__main__':
    from numpy import arange
    import matplotlib.pyplot as plt
    t = arange(0, 1, 1 / 1000)
    plt.figure(figsize=(15, 5))
    plt.subplot(121)
    for k in range(-3, 0):
        plt.plot(t, e(k, t))
    plt.subplot(122)
    for k in range(4):
        plt.plot(t, e(k, t))
```



---

**Program:** brown.py

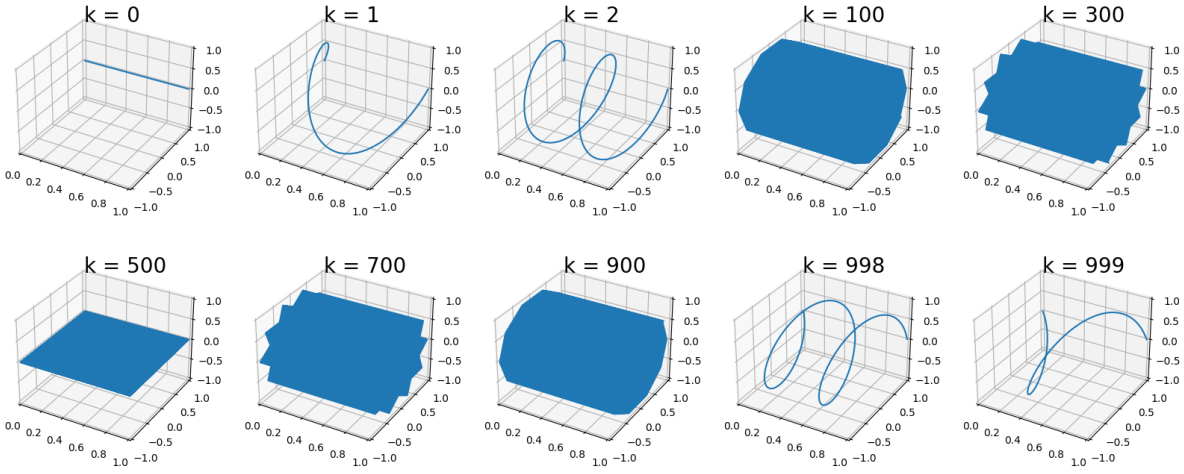
In [1]:
```
from numpy import arange, cumsum, sqrt
from numpy.random import seed, normal
import matplotlib.pyplot as plt
from trigonometric import lowpass
#from fourier import lowpass

seed(2021)

n = 1000
dt = 1 / n
t = arange(0, 1, dt)
f = cumsum(normal(0, sqrt(dt), n))

fig, ax = plt.subplots(3, 3, figsize=(16, 8))
for k, K in enumerate([0, 1, 2, 4, 8, 16, 32, 64, 128]):
    i, j = divmod(k, 3)
    f_K = lowpass(K, t, f)
    ax[i][j].plot(t, f), ax[i][j].plot(t, f_K)
    ax[i][j].text(0.1, min(f), f'K = {K}', fontsize = 20)
    ax[i][j].set_xlim(0, 1)
```




---

**Program:** fourier.py

In [1]:
```
from numpy import arange, exp, pi, vdot
from gram_schmidt import proj


def e(k, t): return exp(2j * pi * k * t)


def lowpass(K, t, z):
    dt = 1 / len(t)
    E_K = [e(k, t) for k in range(-K, K + 1)]
    return proj(z, E_K, inner=lambda x, y: vdot(x, y) * dt)


if __name__ == '__main__':
    import matplotlib.pyplot as plt
    from mpl_toolkits.mplot3d import Axes3D

    fig = plt.figure(figsize=(20, 8))
    t = arange(0, 1, 1 / 1000)
    for n, k  in enumerate([0, 1, 2, 100, 300, 500, 700, 900, 998, 999]):
        x = [z.real for z in e(k, t)]
        y = [z.imag for z in e(k, t)]
        #print(n, k)
        ax = fig.add_subplot(2, 5, n+1, projection="3d")
        ax.set_xlim(0, 1), ax.set_ylim(-1, 1), ax.set_zlim(-1, 1)
        ax.plot(t, x, y), ax.text(0, 1, 1, f'k = {k}', fontsize = 20)
```

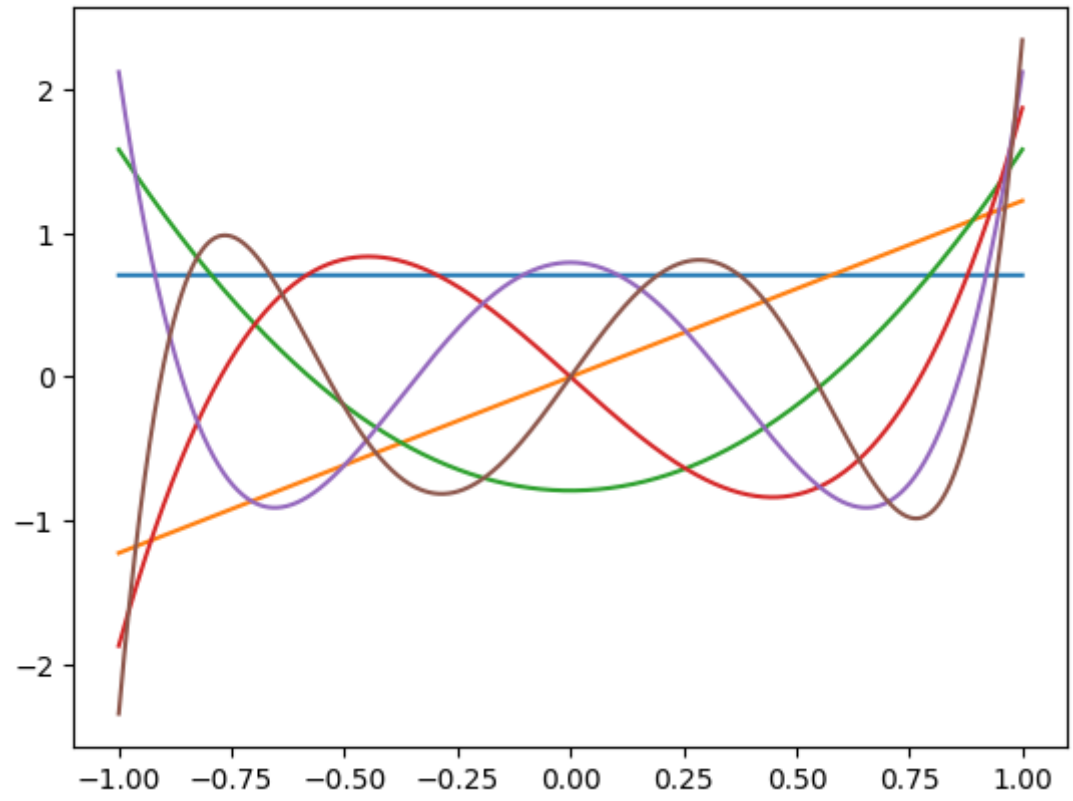


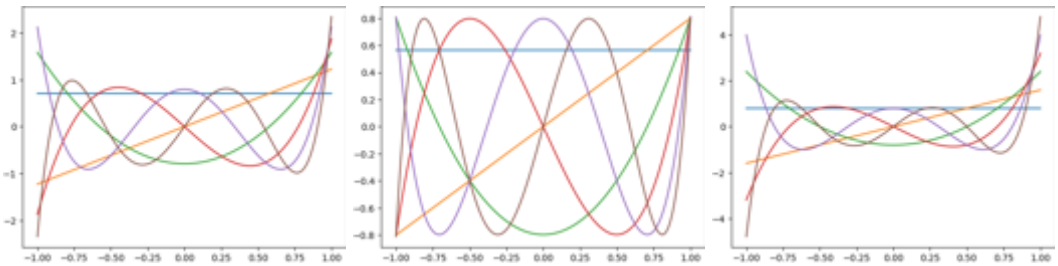

## 6.5. Orthogonal function system

**Program:** poly_np1.py

In [2]:
```
from numpy import array, linspace, sqrt, ones, pi
import matplotlib.pyplot as plt
from gram_schmidt import gram_schmidt

m = 10000
D = linspace(-1, 1, m + 1)
x = array([(D[n] + D[n+1])/2 for n in range(m)])

inner = {
    'Legendre': lambda f, g: f.dot(g) * 2/m,
    'Chebyshev1': lambda f, g: f.dot(g/sqrt(1 - x**2)) * 2/m,
    'Chebyshev2': lambda f, g: f.dot(g*sqrt(1 - x**2)) * 2/m,
}
```

```
A = [x**n for n in range(6)]
E = gram_schmidt(A, inner=inner['Legendre'])
for e in E:
    plt.plot(x, e)
```



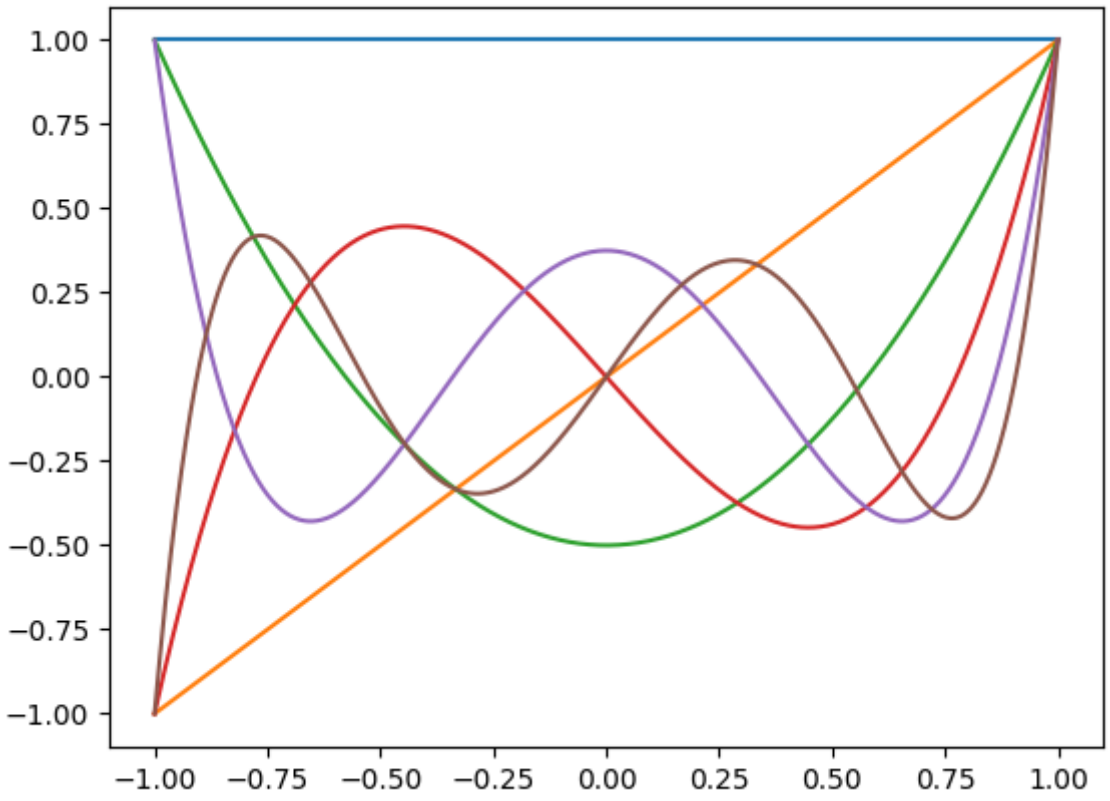



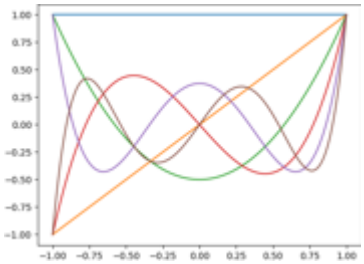
---

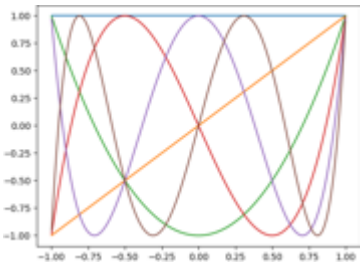
**Program:** poly_np2.py

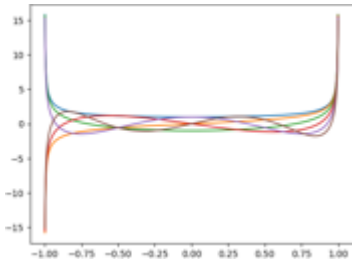
In [3]:
```
from numpy import linspace, exp, sqrt
from numpy.polynomial.legendre import Legendre
from numpy.polynomial.chebyshev import Chebyshev
from numpy.polynomial.laguerre import Laguerre
from numpy.polynomial.hermite import Hermite
import matplotlib.pyplot as plt

x1 = linspace(-1, 1, 1001)
x2 = x1[1:-1]
x3 = linspace(0, 10, 1001)
x4 = linspace(-3, 3, 1001)

f, x, w = Legendre, x1, 1
# f, x, w = Chebyshev, x2, 1
# f, x, w = Chebyshev, x2, 1/sqrt(1 - x2**2)
# f, x, w = Laguerre, x3, 1
# f, x, w = Laguerre, x3, exp(-x3)
```

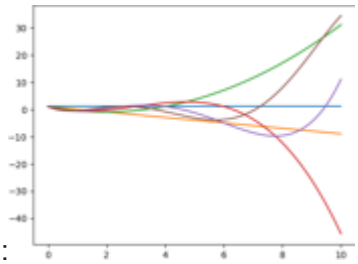
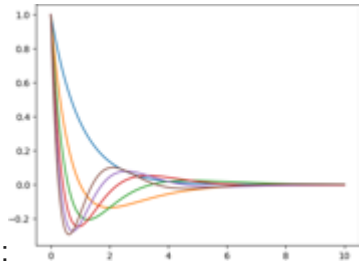
```
# f, x, w = Hermite, x4, 1
#f, x, w = Hermite, x4, exp(-x4**2)
for n in range(6):
    e = f.basis(n)(x)
    plt.plot(x, e * w)
```

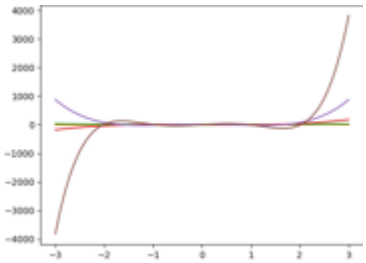
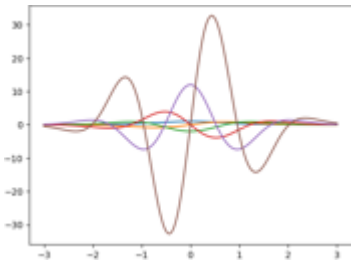



Legendre: 

Chebyshev:  Chebyshev(weighted):



Laguerre:  Laguerre(weighted): 

Hermite:  Hermite(weighted): 

---

**Program:** poly_sp1.py

In [1]:
```
from sympy import integrate, sqrt, exp, oo
from sympy.abc import x

D = {
    'Ledendre': ((x, -1, 1), 1),
    'Chebyshev1': ((x, -1, 1), 1 / sqrt(1 - x**2)),
    'Chebyshev2': ((x, -1, 1), sqrt(1 - x**2)),
    'Laguerre': ((x, 0, oo), exp(-x)),
    'Hermite': ((x, -oo, oo), exp(-x**2)),
}
dom, weight = D['Ledendre']


def inner(expr1, expr2):
    return integrate(expr1*expr2*weight, dom)


def gram_schmidt(A):
    E = []
    while A != []:
        a = A.pop(0)
        b = a - sum([inner(e, a) * e for e in E])
        c = sqrt(inner(b, b))
        E.append(b / c)
    return E

E = gram_schmidt([1, x, x**2, x**3])
for n, e in enumerate(E):
    print(f'e{n}(x) = {e}')
```

```
e0(x) = sqrt(2)/2
e1(x) = sqrt(6)*x/2
e2(x) = 3*sqrt(10)*(x**2 - 1/3)/4
e3(x) = 5*sqrt(14)*(x**3 - 3*x/5)/4
```

---

**Program:** poly_sp2.py

In [1]:
```
from sympy.polys.orthopolys import (
    legendre_poly,
    chebyshevt_poly,
    chebyshevu_poly,
    laguerre_poly,
    hermite_poly,
)
from sympy.abc import x

e = legendre_poly
for n in range(4):
    print(f'e{n}(x) = {e(n, x)}')
```

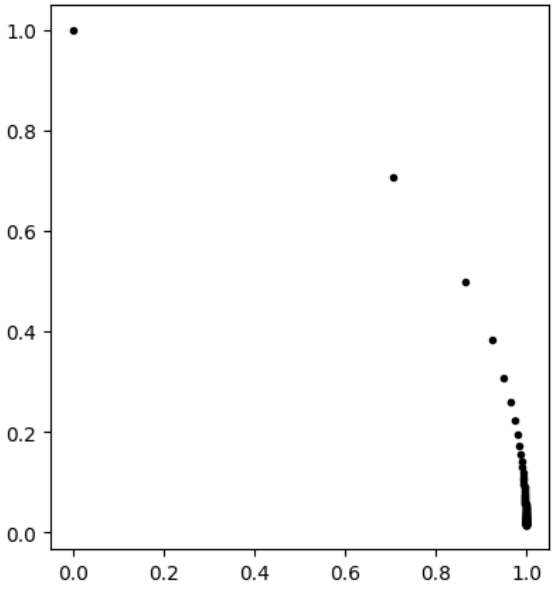
```
e0(x) = 1
e1(x) = x
e2(x) = 3*x**2/2 - 1/2
e3(x) = 5*x**3/2 - 3*x/2
```

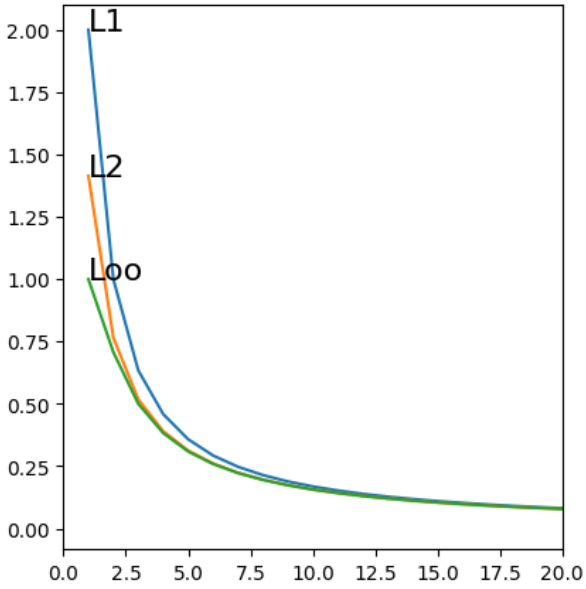
## 6.6. Convergence of vector sequences

**Programs:** limit.py

In [1]:
```
from numpy import array, sin, cos, pi, inf
from numpy.linalg import norm
import matplotlib.pyplot as plt


def A(t):
    return array([[cos(t), -sin(t)], [sin(t), cos(t)]])


x = array([1, 0])
P, L1, L2, Loo = [], [], [], []
M = range(1, 100)
for m in M:
    xm = A(pi / 2 / m).dot(x)
    P.append(xm)
    L1.append(norm(x - xm, ord=1))
    L2.append(norm(x - xm))
    Loo.append(norm(x - xm, ord=inf))

plt.figure(figsize=(10, 5))
plt.subplot(121)
for p in P:
    plt.plot(p[0], p[1], marker='.', color='black')
plt.subplot(122)
plt.xlim(0, 20)
plt.plot(M, L1), plt.text(1, L1[0], 'L1', fontsize=16)
plt.plot(M, L2), plt.text(1, L2[0], 'L2', fontsize=16)
plt.plot(M, Loo), plt.text(1, Loo[0], 'Loo', fontsize=16)
```

Out[1]:
```
([<matplotlib.lines.Line2D at 0x7f58d6bb80>], Text(1, 1.0, 'Loo'))
```



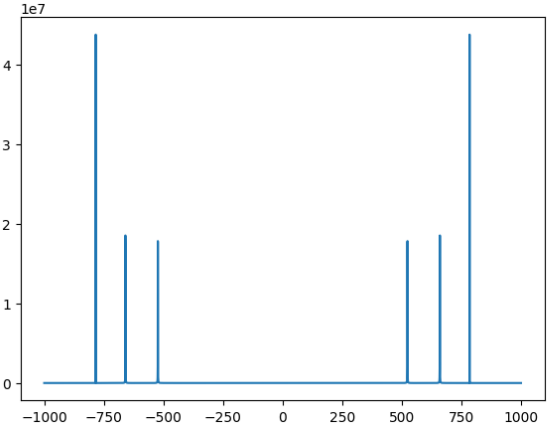



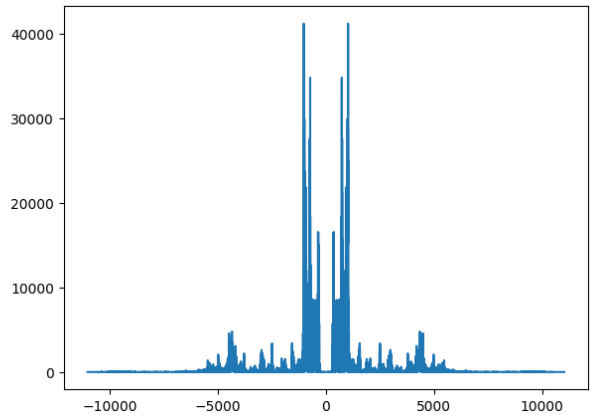

## 6.7. Fourier analysis

**Programs:** spectrum.py

In [1]:
```
import matplotlib.pyplot as plt
from sound import Sound

sound1, sound2 = Sound('CEG'), Sound('mono')

plt.figure(figsize=(15, 5))
plt.subplot(121)
plt.plot(*sound1.power_spectrum((-1000, 1000)))
plt.subplot(122)
plt.plot(*sound2.power_spectrum())
```

Out[1]:
```
[<matplotlib.lines.Line2D at 0x7f44bb0700>]
```



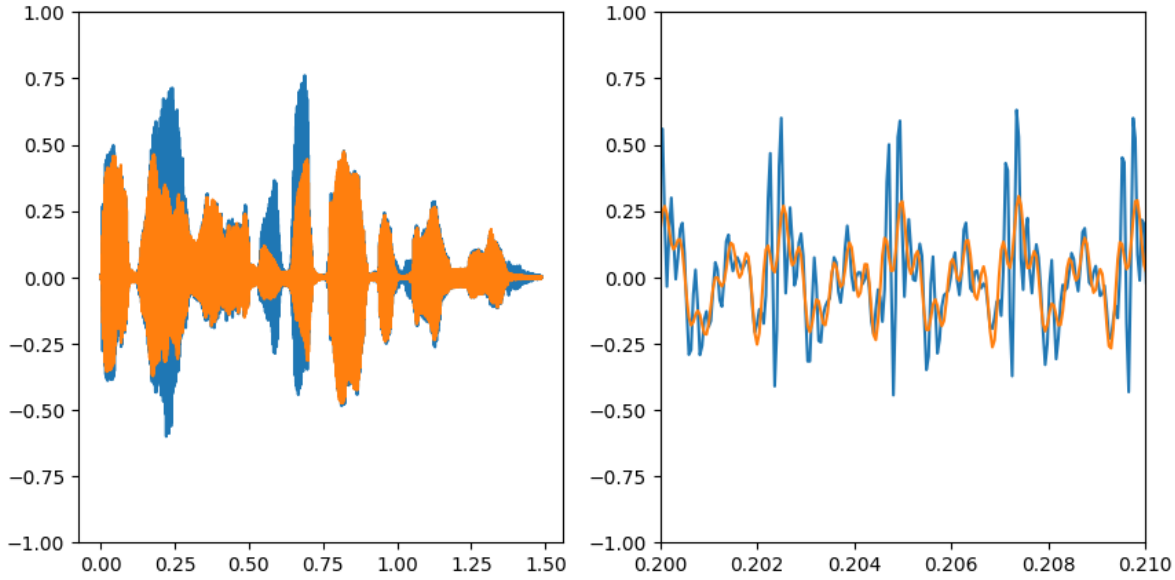

---

**Program:** [lowpass.py](./6.7/lowpass.

In [1]:
```
import matplotlib.pyplot as plt
from sound import Sound

sound = Sound('mono')
X, Y = sound.time, sound.data
Y3000 = sound.lowpass(3000)
```

```
plt.figure(figsize=(10, 5))
plt.subplot(121)
plt.ylim(-1, 1)
plt.plot(X, Y), plt.plot(X, Y3000)
plt.subplot(122)
plt.xlim(0.2, 0.21), plt.ylim(-1, 1)
plt.plot(X, Y), plt.plot(X, Y3000)
```

Out[1]:
```
([<matplotlib.lines.Line2D at 0x7f492ed900>],
 [<matplotlib.lines.Line2D at 0x7f492edc00>])
```




---

**Program:** sound.py

In [1]:
```
from numpy import arange, fft
import scipy.io.wavfile as wav


class Sound:
    def __init__(self, wavfile):
        self.file = wavfile
        self.rate, Data = wav.read(f'{wavfile}.wav')
        dt = 1 / self.rate
        self.len = len(Data)
        self.tmax = self.len / self.rate
        self.time = arange(0, self.tmax, dt)
        self.data = Data.astype('float') / 32768
        self.fft = fft.fft(self.data)

    def power_spectrum(self, rng=None):
        spectrum = abs(self.fft) ** 2
        if rng is None:
            r1, r2 = -self.len / 2, self.len / 2
        else:
            r1, r2 = rng[0] * self.tmax, rng[1] * self.tmax
        R = arange(int(r1), int(r2))
        return R / self.tmax, spectrum[R]

    def lowpass(self, K):
        k = int(K * self.tmax)
        U = self.fft.copy()
```

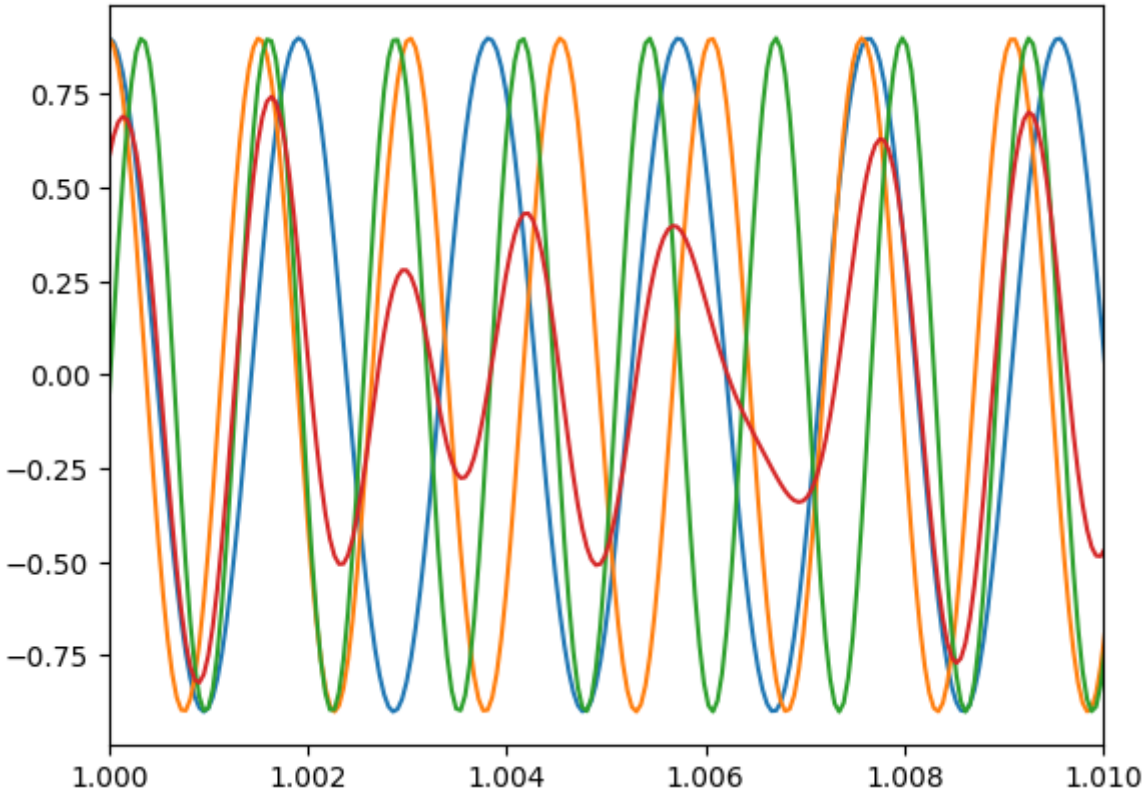
```
        U[range(k + 1, self.len - k)] = 0
        V = fft.ifft(U).real
        Data = (V * 32768).astype('int16')
        wav.write(f'{self.file}{K}.wav', self.rate, Data)
        return V
```

---

**Program:** cord.py

In [1]:
```
from numpy import arange, pi, sin
import scipy.io.wavfile as wav
import matplotlib.pyplot as plt

tmax = 2
rate = 22050
dt = 1 / rate
T = arange(0, tmax, dt)


def f(hz):
    return sin(2 * pi * hz * T) * 0.9


A = [f(440.000000 * 2**n) for n in range(-2, 3)]
B = [f(493.883301 * 2**n) for n in range(-2, 3)]
C = [f(523.251131 * 2**n) for n in range(-2, 3)]
D = [f(587.329536 * 2**n) for n in range(-2, 3)]
E = [f(659.255114 * 2**n) for n in range(-2, 3)]
F = [f(698.456463 * 2**n) for n in range(-2, 3)]
G = [f(783.990872 * 2**n) for n in range(-2, 3)]
CEG = (C[2] + E[2] + G[2]) / 3
Data = (CEG * 32768).astype('int16')
wav.write('CEG.wav', rate, Data)

for Y in [C[2], E[2], G[2], CEG]:
    plt.plot(T, Y)
plt.xlim(1, 1.01)
```

Out[1]:
```
(1.0, 1.01)
```




In [ ]: