Day 24

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
sns.set_theme()
```

In [3]:
```
ti = np.linspace(0,10,100001)
```

In [4]:
```
def vin(ti): return(np.floor(2*ti)%2*-2+1)
```

In [5]:
```
fig0, ax0 = plt.subplots()

ax0.plot(ti, vin(ti))
```

Out[5]: [<matplotlib.lines.Line2D at 0x7ff9ad6b11d0>]

In [8]:
```
# Runge-Kutta 4nd order

def f(vout,t):
    return(1/.01*(vin(t)-vout))

# define boundary conditions

a = 0.0 # starting point
b = 10.0 # ending point
N = 100000 # number of points between a and b
dt = (b-a)/N

x = 0.0 # initial condition


tpoints = np.arange(a, b, dt)
xpoints = []

for t in tpoints:
    xpoints.append(x)
    k1 = dt*f(x,t)
    k2 = dt*f(x+0.5*k1,t+0.5*dt)
    k3 = dt*f(x+0.5*k2,t+0.5*dt)
    k4 = dt*f(x+k3, t+dt)
    x = x + (k1+2*k2+2*k3+k4)/6
```

In [9]:
```
fig1, ax1 = plt.subplots()

ax1.plot(tpoints, vin(tpoints))
ax1.plot(tpoints, xpoints)
```

Out[9]: [<matplotlib.lines.Line2D at 0x7ff9ac281358>]

Now do the above plot, but use a Sin wave, and rather than adjust RC, instead adjust the frequency of the wave. See how the low pass filter lets low frequencies go but stifles higher frequencies.

In [14]:
```
# Runge-Kutta 4nd order

def vin(t):
    f = 1
    return(np.sin(2*np.pi*f*t))

def f(vout,t):
    return(1/0.01*(vin(t)-vout))

# define boundary conditions

a = 0.0 # starting point
b = 1.0 # ending point
```

```
N = 100000 # number of points between a and b
dt = (b-a)/N

x = 0.0 # initial condition


tpoints = np.arange(a, b, dt)
xpoints = []

for t in tpoints:
    xpoints.append(x)
    k1 = dt*f(x,t)
    k2 = dt*f(x+0.5*k1,t+0.5*dt)
    k3 = dt*f(x+0.5*k2,t+0.5*dt)
    k4 = dt*f(x+k3, t+dt)
    x = x + (k1+2*k2+2*k3+k4)/6
```

In [15]:

```
fig2, ax2 = plt.subplots()
ax2.plot(tpoints[:100], vin(tpoints[:100]))
ax2.plot(tpoints[:100], xpoints[:100])
```

Out[15]: [<matplotlib.lines.Line2D at 0x7ff9ac452e80>]

In [16]:

```
plt.close('all')
```

In [ ]:

Loading [MathJax]/extensions/Safe.js