Task 1

Task 1.1 Make the following systems stable, proposing appropriate control

- **a) $$\dot x = \begin{pmatrix} 10 & 0 \\ -5 & 10 \end{pmatrix} x + \begin{pmatrix} 2 \\ 0 \end{pmatrix} u**

$$We can start by writing $'u'$ as$$

u=-Kx

$$we know that 'K' matrix is given by:$$

K =

$$\begin{bmatrix} k_1 & k_2 \end{bmatrix}$$

$$Resubsituting these values into our original equation we will have:$$

\dot x = Ax-BKx

\dot x = (A-BK)x $$ \ If want to make our system stable then we need to make real part of eigenvalues of $(A-BK)$ less or equal to 0.

The code mentioned below helps us to find appropriate values of k_1
and k_2
assuming that resulting eigenvalues of $(A - BK)$
are -1
and -2
. We have already specified the values of matrix A and matrix B. Values of matrix K
will be found.

In [1]:

```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[10, 0], [-5, 10]])
B = np.array([[2], [0]])

# x_dot from state space
def StateSpace(x, t):
    return A.dot(x)

time = np.linspace(0, 1, 1000)
x0 = np.random.rand(n)   # initial state

solution = {"SS": odeint(StateSpace, x0, time)}

#desired eigenvalues
poles = np.array([-1, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
```

```
print("Eigenvalues of A - B*K:", e)
```

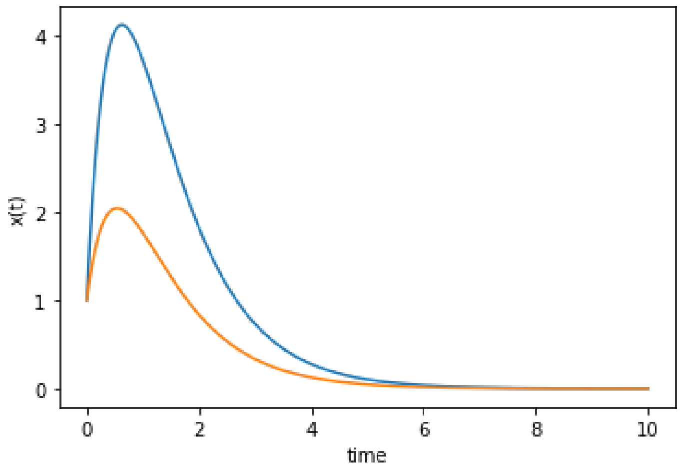
```
K: [[ 11.5 -13.2]]
Eigenvalues of A - B*K: [-2. -1.]
```

In [4]:

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def LTI(x, t):
    return (A - B.dot(K)).dot(x)


time = np.linspace(0, 10, 1000)     # interval from 0 to 10
x0 = np.array([1, 1])           # initial state

solution = odeint(LTI, x0, time)

plt.plot(time, solution)
plt.xlabel('time')
plt.ylabel('x(t)')
```

Out[4]:

```
Text(0, 0.5, 'x(t)')
```



Comparing this resulting graph with another:

Assume, that poles $s_1 = -10$

and $s_2 = -20$

In [5]:

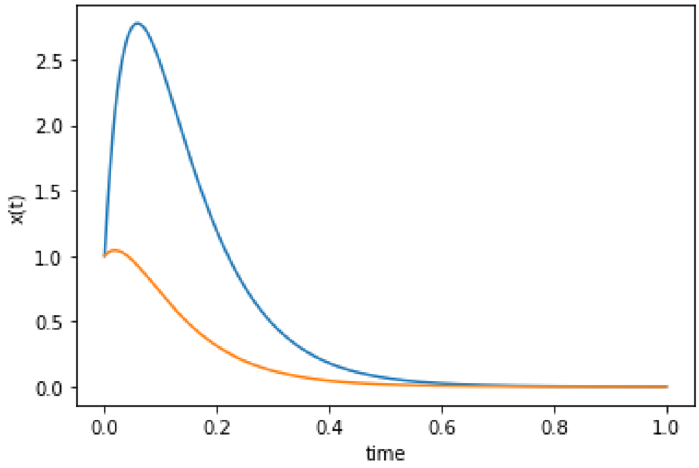
```
#desired eigenvalues
poles = np.array([-10, -20])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)

from scipy.integrate import odeint
import matplotlib.pyplot as plt

def LTI(x, t):
    return (A - B.dot(K)).dot(x)


time = np.linspace(0, 1, 1000)     # interval from 0 to 10
```

```
x0 = np.array([1, 1])          # initial state

solution = odeint(LTI, x0, time)

plt.plot(time, solution)
plt.xlabel('time')
plt.ylabel('x(t)')
```

```
K: [[ 25. -60.]]
eigenvalues of A - B*K: [-20. -10.]
```

Out[5]:

```
Text(0, 0.5, 'x(t)')
```



- **b)**

$$\dot{x} = \begin{pmatrix} 0 & -8 \\ 1 & 30 \end{pmatrix} x + \begin{pmatrix} -2 \\ 1 \end{pmatrix} u$$

Same steps as mentioned for task 1.1a, detailed solution with different values has been presented at 1.1d

In [11]:

```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[0, -8], [1, 30]])
B = np.array([[-2], [1]])

# x_dot from state space
def StateSpace(x, t):
    return A.dot(x) # + B*np.sin(t)

time = np.linspace(0, 1, 1000)
x0 = np.random.rand(n)   # initial state

solution = {"SS": odeint(StateSpace, x0, time)}

#desired eigenvalues
poles = np.array([-1, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)
```

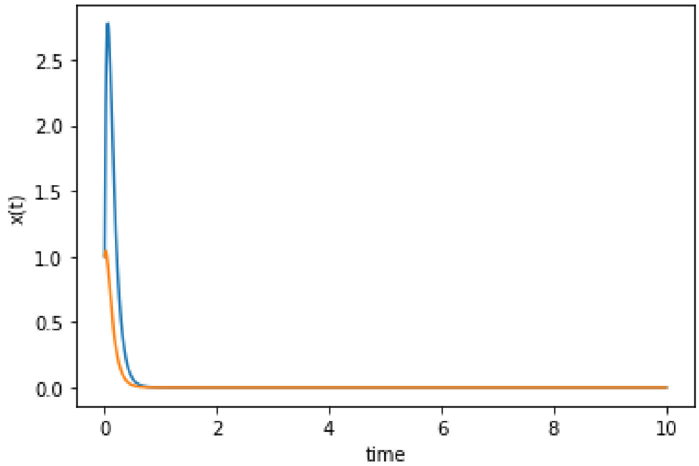
```
K: [[ 1.25 35.5 ]]
eigenvalues of A - B*K: [-1. -2.]
```

In [6]:

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def LTI(x, t):
    return (A - B.dot(K)).dot(x)

time = np.linspace(0, 10, 1000)     # interval from 0 to 10
x0 = np.array([1, 1])           # initial state

solution = odeint(LTI, x0, time)

plt.plot(time, solution)
plt.xlabel('time')
plt.ylabel('x(t)')
```

Out[6]:

```
Text(0, 0.5, 'x(t)')
```



- c)

$$\dot{x} = \begin{pmatrix} 2 & 2 \\ -6 & 10 \end{pmatrix} x + \begin{pmatrix} 0 \\ 5 \end{pmatrix} u$$

Same steps as mentioned for task 1.1a, detailed solution with different values has been presented at 1.1d

In [7]:

```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[2, 2], [-6, 10]])
B = np.array([[0], [5]])

# x_dot from state space
def StateSpace(x, t):
    return A.dot(x)# + B*np.sin(t)

time = np.linspace(0, 1, 1000)
x0 = np.random.rand(n)  # initial state

solution = {"SS": odeint(StateSpace, x0, time)}

#desired eigenvalues
```

```python
poles = np.array([-1, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)
```

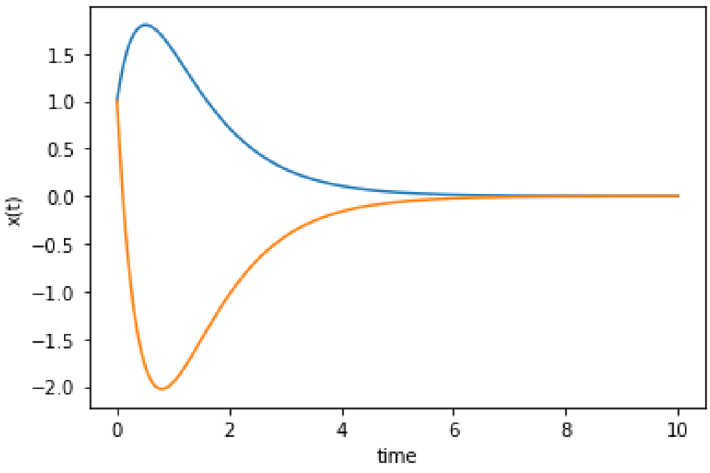
```
K: [[1.42108547e-15 3.00000000e+00]]
eigenvalues of A - B*K: [-1. -2.]
```

In [8]:

```python
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def LTI(x, t):
    return (A - B.dot(K)).dot(x)

time = np.linspace(0, 10, 1000)     # interval from 0 to 10
x0 = np.array([1, 1])           # initial state

solution = odeint(LTI, x0, time)

plt.plot(time, solution)
plt.xlabel('time')
plt.ylabel('x(t)')
```

Out[8]:

```
Text(0, 0.5, 'x(t)')
```



- **d)**

$$\dot{x} = \begin{pmatrix} 5 & -5 \\ 5 & 16 \end{pmatrix} x + \begin{pmatrix} -10 \\ 10 \end{pmatrix} u$$

Step 1:

$$det(sI - A) = (s-5)(s-15) + 30 = s^2 - 20s + 105 = 0$$

Solving this equation, we get:

Eigenvalues are $\lambda_1 = 10 + 2.2361i$
and $\lambda_2 = 10 - 2.2361i$

As a result we can clearly see that the system is unstable.

Step 2: Define $u = -[k_1 \quad k_2] \qquad x(t) = -Kx(t)$

, then

$$A_{cl} = A - BK = \begin{bmatrix} 5 & -5 \\ 6 & 15 \end{bmatrix} - \begin{bmatrix} -10 \\ 10 \end{bmatrix} \begin{bmatrix} k_1 & k_2 \end{bmatrix} = \begin{bmatrix} 5+10k_1 & -5+10k_2 \\ 6-10k_1 & 15-10k_2 \end{bmatrix}$$

Simplifying the above given equation we get:

$$det(sI - A_{cl}) = (s-5-10k_1)(s-15+10k_2) - (6-10k_1)(10k_2-5) = s^2 - 20s + 10sk_2 + 75 - 50k_2 - 10sk_1 + 150k_1 - 100k_1k_2 + 30 - 6($$

Thus, we choose such k_1
and k_2
that we can put $\lambda_i(A_{cl})$
anywhere in the complex plane

Step 3: To put the poles at $s = -1, -2$
, we compare the desired characteristic equation

$$(s+1)(s+2) = s^2 + 3s + 2 = 0$$

with the closed-loop one

$$s^2 + (10k_2 - 10k_1 - 20)s + (100k_1 - 110k_2 - 200k_1k_2 + 45) = 0$$

Simplifying:

$$10k_2 - 10k_1 - 20 = 3$$
$$100k_1 - 110k_2 + 105 = 2$$

$$\Longrightarrow \begin{aligned} k_1 &= -15 \\ k_2 &= -12.7 \end{aligned}$$

so that $K = [-15 \quad -12.7]$

In [9]:

```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[5, -5], [6, 15]])
B = np.array([[-10], [10]])

# x_dot from state space
def StateSpace(x, t):
    return A.dot(x)# + B*np.sin(t)

time = np.linspace(0, 1, 1000)
x0 = np.random.rand(n)   # initial state

solution = {"SS": odeint(StateSpace, x0, time)}

#desired eigenvalues
poles = np.array([-1, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)
```

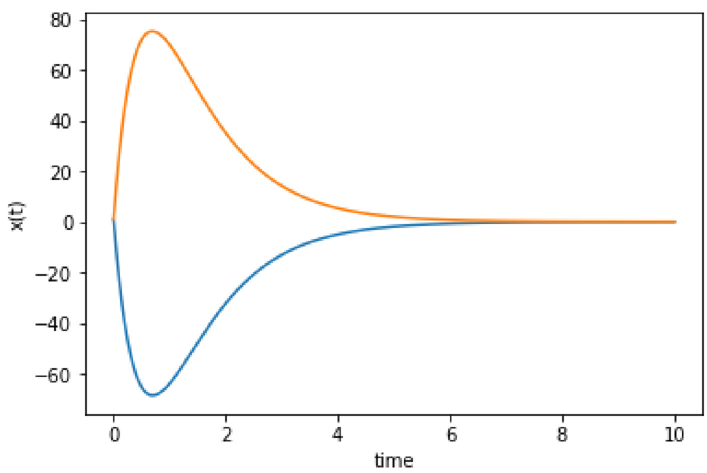
```
K: [[-15.  -12.7]]
eigenvalues of A - B*K: [-2. -1.]
```

In [10]:

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def LTI(x, t):
    return (A - B.dot(K)).dot(x)

time = np.linspace(0, 10, 1000)     # interval from 0 to 10
x0 = np.array([1, 1])          # initial state

solution = odeint(LTI, x0, time)

plt.plot(time, solution)
plt.xlabel('time')
plt.ylabel('x(t)')
```

Out[10]:

```
Text(0, 0.5, 'x(t)')
```



Task 1.2 Make the following systems stables, proposing appropriate control

- **a)**

$$\dot{x} = \begin{pmatrix} 10 & 0 \\ -5 & 10 \end{pmatrix} x + \begin{pmatrix} 2 & 1 \\ 0 & -1 \end{pmatrix} u$$

To solve this task we can use the same method as in the previous task, i.e. rewrite u **as**

$$u = -Kx$$

But since now B **is a** $2 * 2$ **square matrix ,** K **is also a** $2 * 2$ **matrix, therefore:**

$$K = \begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix}$$

Resubsituting these values:

$$\dot{x} = Ax - BKx$$

$$\dot{x} = (A - BK)x$$

\ If want to make our system stable then we need to make real part of eigenvalues of $(A - BK)$ less or equal to 0.

The code mentioned below helps us to find appropriate values of k_1 and k_2 assuming that resulting eigenvalues of $(A - BK)$ are -1 and -2. We have already specified the values of matrix A and matrix B. Values of matrix K will be found.

In [11]:

```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[10, 0], [-5, 10]])
B = np.array([[2, 1], [0, -1]])

# x_dot from state space
def StateSpace(x, t):
    return A.dot(x)

time = np.linspace(0, 1, 1000)
x0 = np.random.rand(n)  # initial state

solution = {"SS": odeint(StateSpace, x0, time)}

#desired eigenvalues
poles = np.array([-1, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)
```

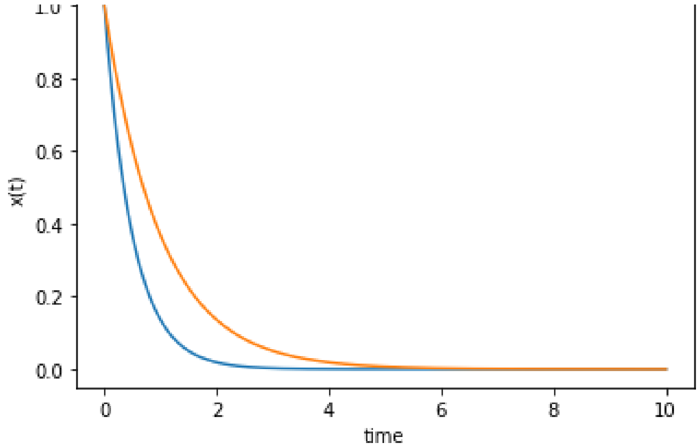
```
K: [[  3.5   5.5]
 [  5.  -11. ]]
eigenvalues of A - B*K: [-1. -2.]
```

In [12]:

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def LTI(x, t):
    return (A - B.dot(K)).dot(x)

time = np.linspace(0, 10, 1000)    # interval from 0 to 10
x0 = np.array([1, 1])          # initial state

solution = odeint(LTI, x0, time)

plt.plot(time, solution)
plt.xlabel('time')
plt.ylabel('x(t)')
```

Out[12]:

```
Text(0, 0.5, 'x(t)')
```



- **b)**

$$\dot{x} = \begin{pmatrix} 0 & -8 \\ 1 & 30 \end{pmatrix} x + \begin{pmatrix} -2 & 1 \\ 1 & 1 \end{pmatrix} u$$

Same steps as in 1.2a

In [13]:

```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[0, -8], [1, 30]])
B = np.array([[-2, 1], [1, 1]])

# x_dot from state space
def StateSpace(x, t):
    return A.dot(x)

time = np.linspace(0, 1, 1000)
x0 = np.random.rand(n)   # initial state

solution = {"SS": odeint(StateSpace, x0, time)}

#desired eigenvalues
poles = np.array([-1, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)
```

```
K: [[-0.33333333 13.        ]
 [ 1.33333333 18.        ]]
eigenvalues of A - B*K: [-2. -1.]
```

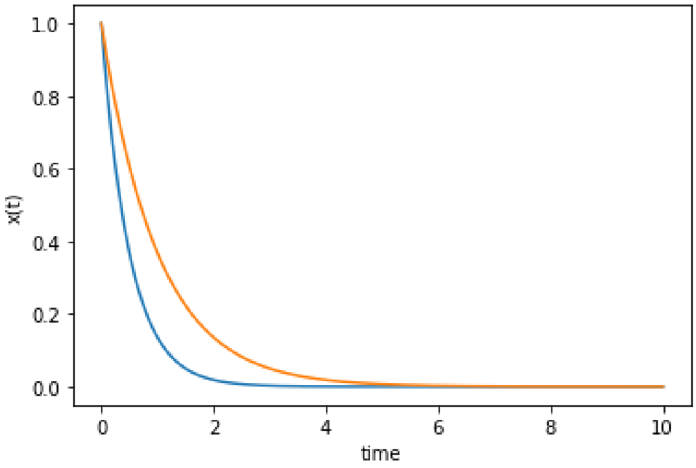
In [14]:

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def LTI(x, t):
    return (A - B.dot(K)).dot(x)

time = np.linspace(0, 10, 1000)     # interval from 0 to 10
x0 = np.array([1, 1])             # initial state
```

```
solution = odeint(LTI, x0, time)

plt.plot(time, solution)
plt.xlabel('time')
plt.ylabel('x(t)')
```

Out[14]:

```
Text(0, 0.5, 'x(t)')
```



- **c)**

$$\dot{x} = \begin{pmatrix} 2 & 2 \\ -6 & 10 \end{pmatrix} x + \begin{pmatrix} 0 & -1 \\ 5 & -1 \end{pmatrix} u$$

Same steps as in 1.2a

In [15]:

```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[2, 2], [-6, 10]])
B = np.array([[0, -1], [5, -1]])

# x_dot from state space
def StateSpace(x, t):
    return A.dot(x)

time = np.linspace(0, 1, 1000)
x0 = np.random.rand(n)  # initial state

solution = {"SS": odeint(StateSpace, x0, time)}

#desired eigenvalues
poles = np.array([-1, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)
```

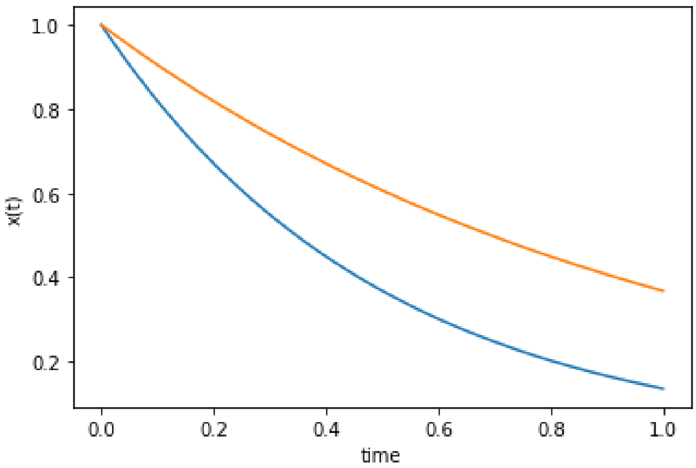
```
K: [[-2.   1.8]
 [-4.  -2. ]]
eigenvalues of A - B*K: [-2. -1.]
```

In [16]:

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def LTI(x, t):
    return (A - B.dot(K)).dot(x)

time = np.linspace(0, 1, 1000)     # interval from 0 to 10
x0 = np.array([1, 1])           # initial state

solution = odeint(LTI, x0, time)

plt.plot(time, solution)
plt.xlabel('time')
plt.ylabel('x(t)')
```

Out[16]:

```
Text(0, 0.5, 'x(t)')
```



- **d)**

$$\dot{x} = \begin{pmatrix} 5 & -5 \\ 6 & 15 \end{pmatrix} x + \begin{pmatrix} -10 & 3 \\ 10 & 3 \end{pmatrix} u$$

In [17]:

```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[5, -5], [6, 15]])
B = np.array([[-10, 3], [10, 3]])

# x_dot from state space
def StateSpace(x, t):
    return A.dot(x)

time = np.linspace(0, 1, 1000)
x0 = np.random.rand(n)   # initial state

solution = {"SS": odeint(StateSpace, x0, time)}

#desired eigenvalues
poles = np.array([-1, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
```

```
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)
```

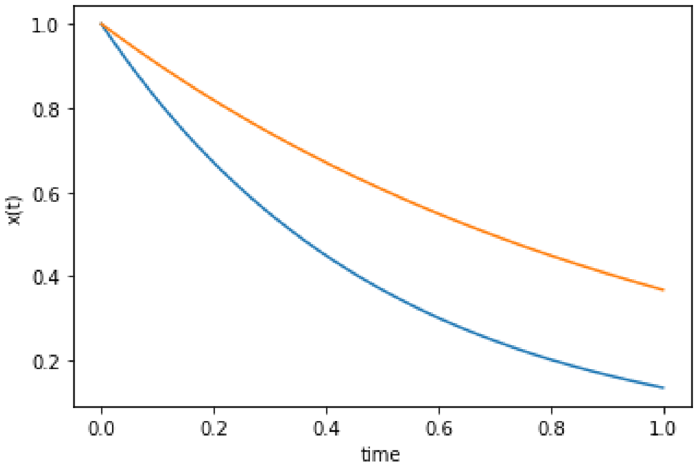
```
K: [[-0.05        1.05      ]
 [ 2.16666667  1.83333333]]
eigenvalues of A - B*K: [-2. -1.]
```

In [18]:

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def LTI(x, t):
    return (A - B.dot(K)).dot(x)

time = np.linspace(0, 1, 1000)     # interval from 0 to 10
x0 = np.array([1, 1])           # initial state

solution = odeint(LTI, x0, time)

plt.plot(time, solution)
plt.xlabel('time')
plt.ylabel('x(t)')
```

Out[18]:

```
Text(0, 0.5, 'x(t)')
```



Task 1.3 Give example of an unstable system that cannot be stabilized

of the form $\dot{x} = Ax + Bu$
, where $A \in \mathbb{R}^{2\times 2}$

- **a) where** $B \in \mathbb{R}^{2\times 1}$

$$\dot{x} = \begin{pmatrix} -5 & 0 \\ -0.1 & 1 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0.5 \end{pmatrix} u$$

- **b) where** $B \in \mathbb{R}^{2\times 2}$

$$\dot{x} = \begin{pmatrix} 4 & 0 \\ -10 & 1 \end{pmatrix} x + \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} u$$

- **c) where** $B \in \mathbb{R}^{2\times 3}$

$$\dot{x} = \begin{pmatrix} -7 & 0 \\ 5 & 1 \end{pmatrix} x + \begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \end{pmatrix} u$$

Task 2 Root Locus

Task 2.1 Plot root locus

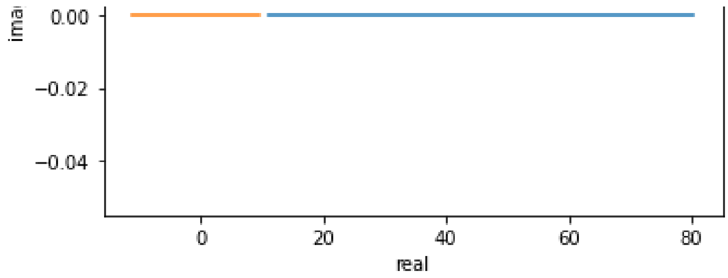
- **a) For a system with A with imaginary eigenvalues**

$$\dot{x} = \begin{pmatrix} 5 & -5 \\ 6 & 15 \end{pmatrix} x + \begin{pmatrix} -10 \\ 10 \end{pmatrix} u$$

In [20]:

```
import matplotlib.pyplot as plt
import numpy as np
from numpy.linalg import eig

A = np.array([[5, -5], [6, 15]])
B = np.array([[-10], [10]])
K0 = np.array([[1, 3]]);

eigen, v = eig(A)

print("eigenvalues of A:", eigen)

k_min = 1;
k_max = 10;
k_step = 0.1;

Count = np.floor((k_max-k_min)/k_step)
Count = Count.astype(int)

k_range = np.linspace(k_min, k_max, Count)

E = np.zeros((Count, 4))

for i in range(Count):
    K0[0, 0] = k_range[i]

    ei, v = eig((A - B.dot(K0)))

    E[i, 0] = np.real(ei[0])
    E[i, 1] = np.imag(ei[0])
    E[i, 2] = np.real(ei[1])
    E[i, 3] = np.imag(ei[1])

print("eigenvalues of A - B*K:", ei)

plt.plot(E[:, 0], E[:, 1])
plt.plot(E[:, 2], E[:, 3])
plt.xlabel('real')
plt.ylabel('imag')
plt.show()
```

```
eigenvalues of A: [10.+2.23606798j 10.-2.23606798j]
eigenvalues of A - B*K: [80.35533906  9.64466094]
```



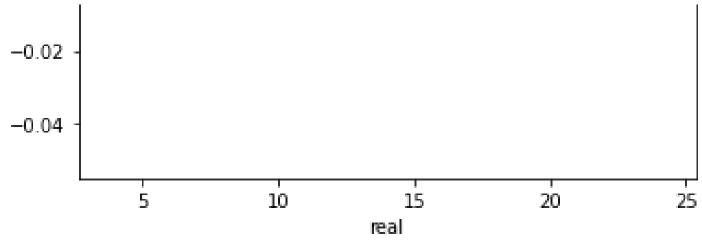
- **b) For a system with A with real eigenvalues**

$$\dot{x} = \begin{pmatrix} 1 & -3 \\ 6 & 12 \end{pmatrix} x + \begin{pmatrix} -1 \\ 2 \end{pmatrix} u$$

In [22]:

```
import matplotlib.pyplot as plt
import numpy as np
from numpy.linalg import eig

A = np.array([[1, -3], [6, 12]])
B = np.array([[-1], [2]])
K0 = np.array([[3, -3]]);

eigen, v = eig(A)

print("eigenvalues of A:", eigen)

k_min = 1;
k_max = 10;
k_step = 0.1;

Count = np.floor((k_max-k_min)/k_step)
Count = Count.astype(int)

k_range = np.linspace(k_min, k_max, Count)

E = np.zeros((Count, 4))

for i in range(Count):
    K0[0, 0] = k_range[i]

    ei, v = eig((A - B.dot(K0)))

    E[i, 0] = np.real(ei[0])
    E[i, 1] = np.imag(ei[0])
    E[i, 2] = np.real(ei[1])
    E[i, 3] = np.imag(ei[1])

print("eigenvalues of A - B*K:", ei)

plt.plot(E[:, 0], E[:, 1])
plt.plot(E[:, 2], E[:, 3])
plt.xlabel('real')
plt.ylabel('imag')
plt.show()
```

```
eigenvalues of A: [ 3. 10.]
eigenvalues of A - B*K: [ 4.68929156 24.31070844]
```



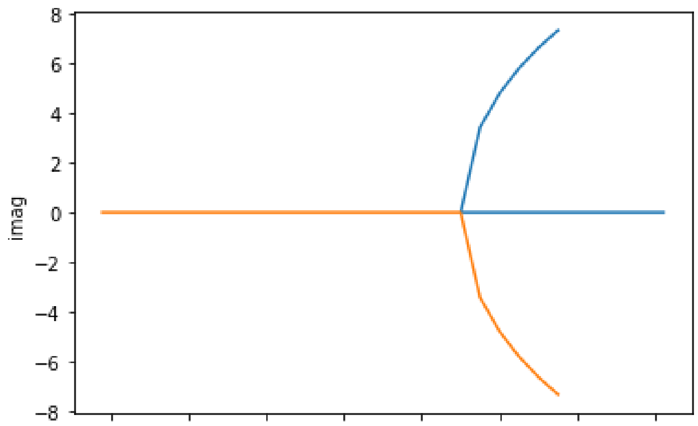
- **c) For a system where real parts of eigenvalues of** $(A-BK)$ **are all positive**

$$\dot{x} = \begin{pmatrix} 2 & 5 \\ 6 & 3 \end{pmatrix} x + \begin{pmatrix} -1 \\ 2 \end{pmatrix} u$$

In [24]:

```
import matplotlib.pyplot as plt
import numpy as np
from numpy.linalg import eig

A = np.array([[2, 5], [6, 3]])
B = np.array([[-1], [2]])
K0 = np.array([[1, 4]]);

k_min = 1;
k_max = 10;
k_step = 0.1;

Count = np.floor((k_max-k_min)/k_step)
Count = Count.astype(int)

k_range = np.linspace(k_min, k_max, Count)

E = np.zeros((Count, 4))

for i in range(Count):
    K0[0, 0] = k_range[i]

    ei, v = eig((A - B.dot(K0)))

    E[i, 0] = np.real(ei[0])
    E[i, 1] = np.imag(ei[0])
    E[i, 2] = np.real(ei[1])
    E[i, 3] = np.imag(ei[1])

print("eigenvalues of A - B*K:", ei)

plt.plot(E[:, 0], E[:, 1])
plt.plot(E[:, 2], E[:, 3])
plt.xlabel('real')
plt.ylabel('imag')
plt.show()
```

```
eigenvalues of A - B*K: [3.5+7.33143915j 3.5-7.33143915j]
```



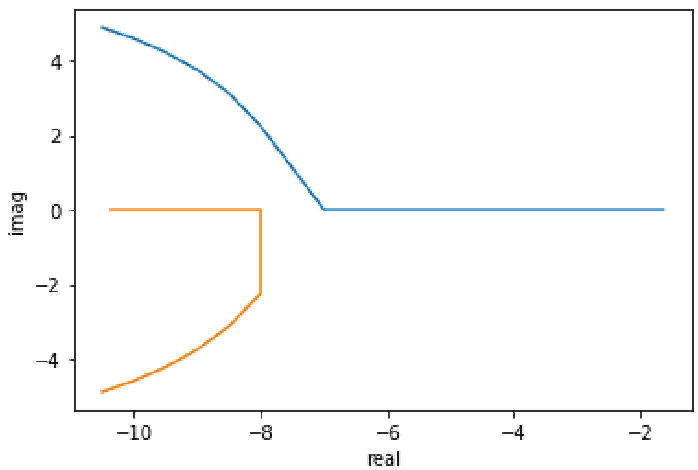
- **d) For a system where real parts of eigenvalues of $(A - BK)$ are all negative**

$$\dot{x} = \begin{pmatrix} -2 & -5 \\ -7 & 3 \end{pmatrix} x + \begin{pmatrix} 1 \\ -2 \end{pmatrix} u$$

In [29]:

```
import matplotlib.pyplot as plt
import numpy as np
from numpy.linalg import eig

A = np.array([[-2, -5], [-7, -3]])
B = np.array([[1], [-2]])
K0 = np.array([[5, -3]]);

k_min = 1;
k_max = 10;
k_step = 0.1;

Count = np.floor((k_max-k_min)/k_step)
Count = Count.astype(int)

k_range = np.linspace(k_min, k_max, Count)

E = np.zeros((Count, 4))

for i in range(Count):
    K0[0, 0] = k_range[i]

    ei, v = eig((A - B.dot(K0)))

    E[i, 0] = np.real(ei[0])
    E[i, 1] = np.imag(ei[0])
    E[i, 2] = np.real(ei[1])
    E[i, 3] = np.imag(ei[1])

print("eigenvalues of A - B*K:", ei)

plt.plot(E[:, 0], E[:, 1])
plt.plot(E[:, 2], E[:, 3])
plt.xlabel('real')
plt.ylabel('imag')
plt.show()
```

```
eigenvalues of A - B*K: [-10.5+4.87339717j -10.5-4.87339717j]
```



Reaction to inputs

Task 3 Step functions

Task 3.1 Simulate the first system of (1.1) with a step function as an input.

$$\dot{x} = \begin{pmatrix} 10 & 0 \\ -5 & 10 \end{pmatrix} x + \begin{pmatrix} 2 \\ 0 \end{pmatrix} u$$

$$f_1 = \begin{cases} 1, & t \geq t_1 \\ 0, & t < t_1 \end{cases}$$

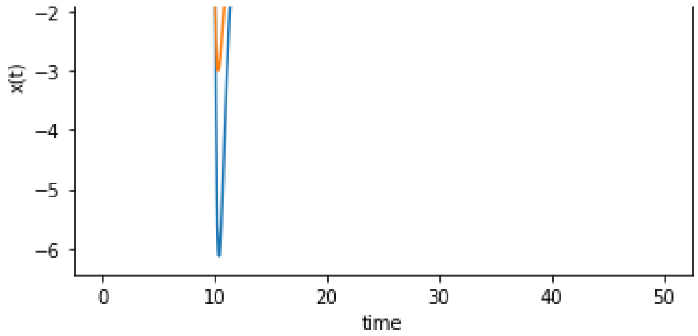
In [30]:

```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[10, 0], [-5, 10]])
B = np.array([[2], [0]])

#desired eigenvalues
poles = np.array([-3, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)

import matplotlib.pyplot as plt

def system_ode(x, t, A, B):
    if t < 10:
      dx = A.dot(x)
    else:
      dx = (A - B.dot(K)).dot(x)
    return dx


time = np.linspace(0, 50, 1000)     # interval from 0 to 10
x0 = np.array([1, 1])           # initial state

solution = odeint(system_ode, x0, time, args=(A, B,))

plt.plot(time, solution)
plt.xlabel('time')
plt.ylabel('x(t)')
```

```
K: [[ 12.5 -15.6]]
eigenvalues of A - B*K: [-3. -2.]
```

Out[30]:

```
Text(0, 0.5, 'x(t)')
```



Task 3.2 Linear combination of solutions

Simulate the first system of (1.1) with two different step functions f_1, f_2 as an input, and with $f_1 + f_2$ as an input. Compare the solutions for f_1, f_2 with the solution for $f_1 + f_2$.

$$\dot{x} = \begin{pmatrix} 10 & 0 \\ -5 & 10 \end{pmatrix} x + \begin{pmatrix} 2 \\ 0 \end{pmatrix} u$$

$$f_1 = \begin{cases} 1, & t \geq t_1 \\ 0, & t < t_1 \end{cases}$$

$$f_2 = \begin{cases} 1, & t \geq t_2 \\ 0, & t < t_2 \end{cases}$$

In [31]:

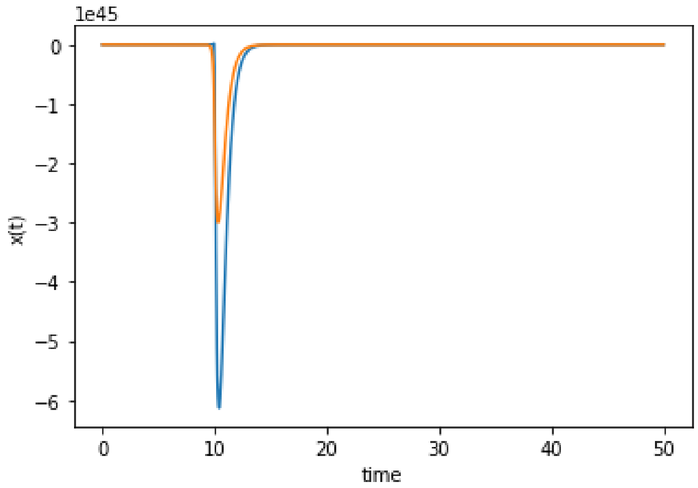
```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[10, 0], [-5, 10]])
B = np.array([[2], [0]])

#desired eigenvalues
poles = np.array([-3, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

import matplotlib.pyplot as plt

def system_ode_f1(x, t, A, B):
    if t < 10:
      dx = A.dot(x)
    else:
      dx = (A - B.dot(K)).dot(x)
    return dx


time = np.linspace(0, 50, 1000)     # interval from 0 to 10
x0 = np.array([1, 1])            # initial state
```

```
solution = odeint(system_ode_f1, x0, time, args=(A, B,))

plt.plot(time, solution)
plt.xlabel('time')
plt.ylabel('x(t)')
```

K: [[12.5 -15.6]]

Out[31]:

Text(0, 0.5, 'x(t)')



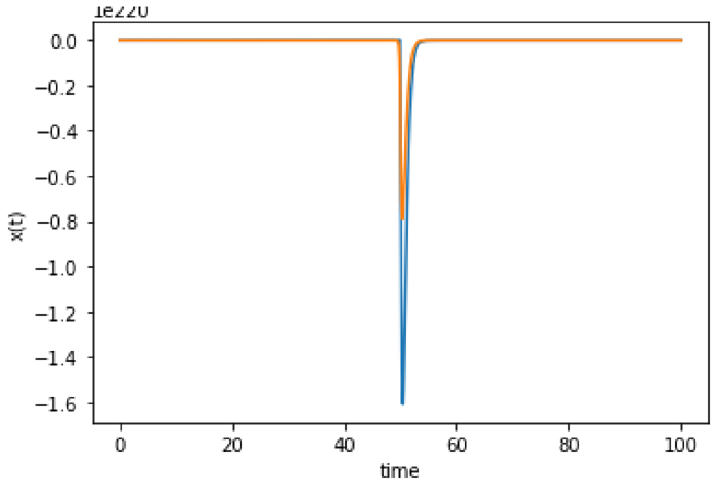
In [32]:

```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[10, 0], [-5, 10]])
B = np.array([[2], [0]])

#desired eigenvalues
poles = np.array([-3, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

import matplotlib.pyplot as plt

def system_ode_f2(x, t, A, B):
    if t < 50:
      dx = A.dot(x)
    else:
      dx = (A - B.dot(K)).dot(x)
    return dx


time = np.linspace(0, 100, 1000)     # interval from 0 to 10
x0 = np.array([1, 1])            # initial state

solution = odeint(system_ode_f2, x0, time, args=(A, B,))

plt.plot(time, solution)
plt.xlabel('time')
plt.ylabel('x(t)')
```

K: [[12.5 -15.6]]

Out[32]:

Text(0, 0.5, 'x(t)')



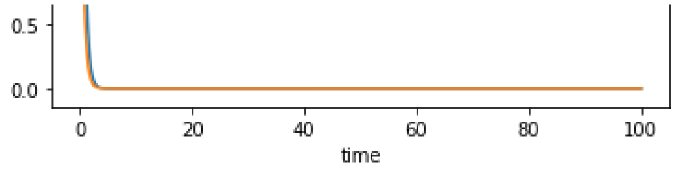
In [33]:

```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[10, 0], [-5, 10]])
B = np.array([[2], [0]])

#desired eigenvalues
poles = np.array([-5, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

import matplotlib.pyplot as plt

def system_ode_f12(x, t, A, B):
    if t < 10:
      dx = A.dot(x)
    if t>=50:
      dx = (A - B.dot(K) - B.dot(K)).dot(x)
    else:
      dx = (A - B.dot(K)).dot(x)
    return dx


time = np.linspace(0, 100, 1000)     # interval from 0 to 10
x0 = np.array([1, 1])              # initial state

solution = odeint(system_ode_f12, x0, time, args=(A, B))

plt.plot(time, solution)
plt.xlabel('time')
plt.ylabel('x(t)')
```

```
K: [[ 13.5 -18. ]]
```

Out[33]:

```
Text(0, 0.5, 'x(t)')
```



Task 4 Sinusoidal input

Taks 4.1 Simulate the first system of (1.1) with a sinusoidal input $u = sin(wt)$

.

How does the choice of w affects the result?

$$\dot{x} = \begin{pmatrix} 10 & 0 \\ -5 & 10 \end{pmatrix} x + \begin{pmatrix} 2 \\ 0 \end{pmatrix} u$$

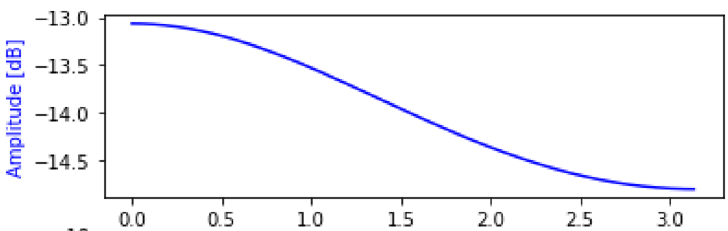
In [34]:

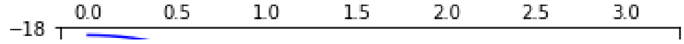
```
from scipy.signal import ss2tf
from scipy.signal import freqz
import matplotlib.pyplot as plt
import numpy as np
from numpy.linalg import eig


A = np.array([[10, 0], [-5, 10]])
B = np.array([[2], [0]])
C = np.eye(2)
D = np.zeros((2, 1))

num, den = ss2tf(A, B, C, D)

print("num:", num)
print("den:", den)

w1, h1 = freqz(num[0, :], den)
w2, h2 = freqz(num[1, :], den)

plt.subplot(211)
plt.plot(w1, 20 * np.log10(abs(h1)), 'b')
plt.ylabel('Amplitude [dB]', color='b')
plt.xlabel('Frequency [rad/sample]')

plt.subplot(212)
plt.plot(w2, 20 * np.log10(abs(h2)), 'b')
plt.ylabel('Amplitude [dB]', color='b')
plt.xlabel('Frequency [rad/sample]')
```

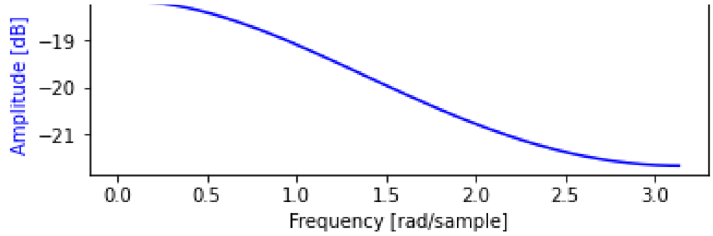
```
num: [[  0.    2. -20.]
 [  0.    0. -10.]]
den: [  1. -20. 100.]
```

Out[34]:

```
Text(0.5, 0, 'Frequency [rad/sample]')
```







Task 4.2 Make frequency diagrams for 2 of the systems you studied in the tasks 1.1 and 1.2

Using task 1.1 for reference:

- **a)**

$$\dot{x} = \begin{pmatrix} 0 & -8 \\ 1 & 30 \end{pmatrix} x + \begin{pmatrix} 2 \\ 0 \end{pmatrix} u$$

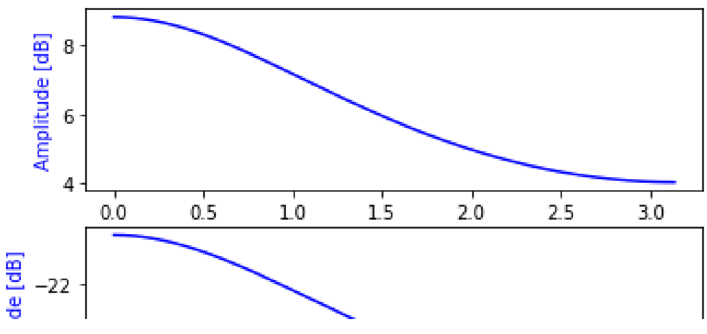
In [35]:

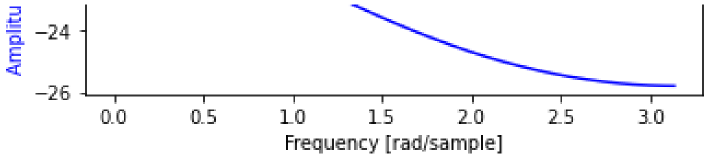
```
from scipy.signal import ss2tf
from scipy.signal import freqz
import matplotlib.pyplot as plt
import numpy as np
from numpy.linalg import eig


A = np.array([[0, -8], [1, 30]])
B = np.array([[2], [0]])
C = np.eye(2)
D = np.zeros((2, 1))

num, den = ss2tf(A, B, C, D)

print("num:", num)
print("den:", den)

w1, h1 = freqz(num[0, :], den)
w2, h2 = freqz(num[1, :], den)

plt.subplot(211)
plt.plot(w1, 20 * np.log10(abs(h1)), 'b')
plt.ylabel('Amplitude [dB]', color='b')
plt.xlabel('Frequency [rad/sample]')

plt.subplot(212)
plt.plot(w2, 20 * np.log10(abs(h2)), 'b')
plt.ylabel('Amplitude [dB]', color='b')
plt.xlabel('Frequency [rad/sample]')
```

```
num: [[  0.    2. -60.]
 [  0.    0.    2.]]
den: [  1. -30.    8.]
```

Out[35]:

```
Text(0.5, 0, 'Frequency [rad/sample]')
```





Using task 1.2 for reference:

- **b)**

$$\dot{x} = \begin{pmatrix} 2 & 2 \\ -6 & 10 \end{pmatrix} x + \begin{pmatrix} 0 & -1 \\ 5 & -1 \end{pmatrix} u$$

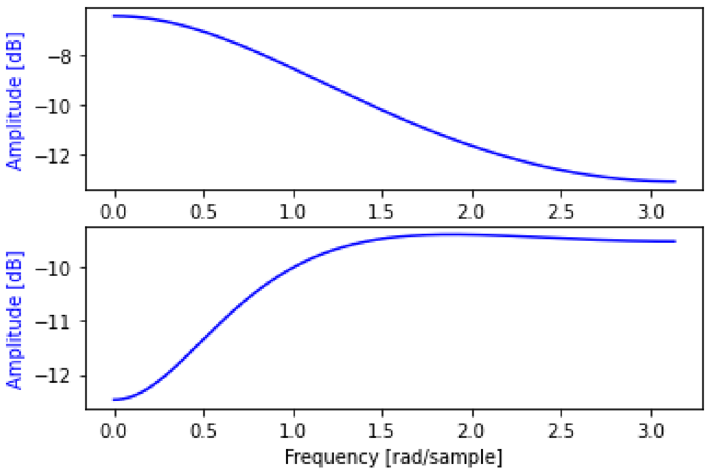
In [36]:

```
from scipy.signal import ss2tf
from scipy.signal import freqz
import matplotlib.pyplot as plt
import numpy as np
from numpy.linalg import eig


A = np.array([[2, 2], [-6, 10]])
B = np.array([[0, -1], [5, -1]])
C = np.eye(2)
D = np.zeros((2, 2))

num, den = ss2tf(A, B, C, D)

print("num:", num)
print("den:", den)

w1, h1 = freqz(num[0, :], den)
w2, h2 = freqz(num[1, :], den)

plt.subplot(211)
plt.plot(w1, 20 * np.log10(abs(h1)), 'b')
plt.ylabel('Amplitude [dB]', color='b')
plt.xlabel('Frequency [rad/sample]')

plt.subplot(212)
plt.plot(w2, 20 * np.log10(abs(h2)), 'b')
plt.ylabel('Amplitude [dB]', color='b')
plt.xlabel('Frequency [rad/sample]')
```

```
num: [[ 0.00000000e+00  1.77635684e-15  1.00000000e+01]
 [ 0.00000000e+00  5.00000000e+00 -1.00000000e+01]]
den: [  1. -12.  32.]
```

Out[36]:

```
Text(0.5, 0, 'Frequency [rad/sample]')
```



Task 5 Point to point control

Task 5.1 Design point-to-point control and simulate two systems:

- **a) where $B \in \mathrm{R}^{2 \times 1}$**

Let matrix $A = \begin{bmatrix} 10 & 0 \\ -5 & 10 \end{bmatrix}$

, $B = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$

and $x_{desired} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

. Also let us take poles $\begin{bmatrix} -1 \\ -2 \end{bmatrix}$

. \ \ As per lectures, We define control as

$$u = -K(x - x_{desired}) + u_{desired}$$

\ Also, we know that $\dot{x}_{desired} = 0$

, therefore:

$$\dot{x}_{desired} = Ax_{desired} + Bu_{desired}$$

****

$$0 = \begin{bmatrix} 10 & 0 \\ -5 & 10 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 \\ 1 \end{bmatrix} u_{desired}$$

****

$$0 = \begin{bmatrix} 10 \\ -5 \end{bmatrix} + \begin{bmatrix} -2u_{desired} \\ u_{desired} \end{bmatrix} \Rightarrow u_{desired} = 5$$

\ Since we know poles, we can find matrix $K = \begin{bmatrix} -4.9 & 13.2 \end{bmatrix}$

. \ \ The following code can help us to simulate the system.

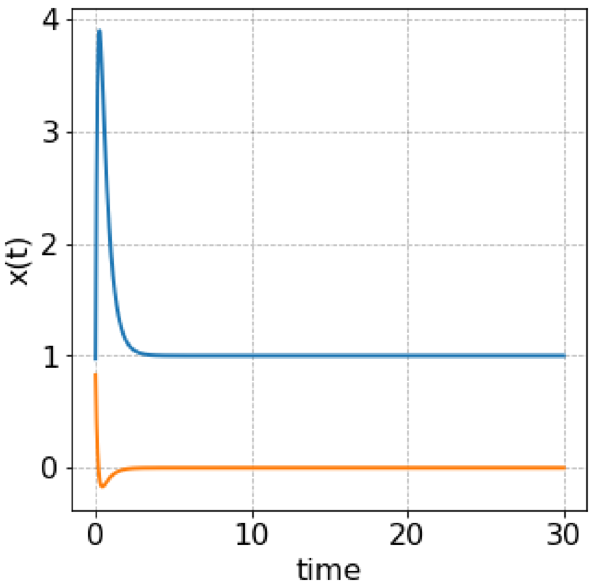
In [42]:

```
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles
import matplotlib.pyplot as plt
import numpy as np

A = np.array([[10, -0], [-5, 10]])
B = np.array([[-2], [1]])

#desired eigenvalues
poles = np.array([-7, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)
```

```
K: [[-4.3 20.4]]
eigenvalues of A - B*K: [-2. -7.]
```

In [43]:

```python
x_desired = np.array([1, 0])
u_desired = np.array([5])
n=2

def StateSpace(x, t):
    u = -K.dot(x - x_desired) + u_desired
    return A.dot(x) + B.dot(u)

time = np.linspace(0, 30, 30000)
x0 = np.random.rand(n)   # initial state

solution = {"solution_1": odeint(StateSpace, x0, time)}

plt.rcParams['figure.figsize'] = [5, 5]
plt.rcParams["font.size"] = 16
plt.rcParams["font.weight"] = 'normal'

# plt.subplot(221)
plt.plot(time, solution["solution_1"], linewidth=2)
plt.xlabel('time')
plt.ylabel('x(t)')
plt.grid(color='k', linestyle='--', linewidth=0.7, alpha=0.3)
# plt.title('autonomous')
```



- **b) where $B \in R^{2\times 2}$**

  \ Let take matrix $A = \begin{bmatrix} 10 & 0 \\ -5 & 10 \end{bmatrix}$

  , $B = \begin{bmatrix} 2 & 1 \\ 0 & -1 \end{bmatrix}$

  and $x_{desired} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

  . Also let us take poles $\begin{bmatrix} -1 \\ -2 \end{bmatrix}$

  . \ \ As per lectures, we can define control as

$$u = -K(x - x_{desired}) + u_{desired}$$

  \ Also, we know that $\dot{x}_{desired} = 0$

  , Therefore:

$$\dot{x}_{desired} = Ax_{desired} + Bu_{desired}$$

  ****

.

$$0 = \begin{bmatrix} 10 & 0 \\ -5 & 10 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 2 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} u_{d1} \\ u_{d2} \end{bmatrix}$$

\

$$0 = \begin{bmatrix} 10 \\ -5 \end{bmatrix} + \begin{bmatrix} 2u_{d1} + u_{d2} \\ -u_{d2} \end{bmatrix} \Rightarrow u_{d2} = 5, u_{d1} = -7.5 \Rightarrow u = \begin{bmatrix} -7.5 \\ 5 \end{bmatrix}$$

\ Since we know poles, we can find matrix $K = \begin{bmatrix} 3.5 & 5.5 \\ 5 & -11 \end{bmatrix}$

. \ \ The following code can help us to simulate the system.

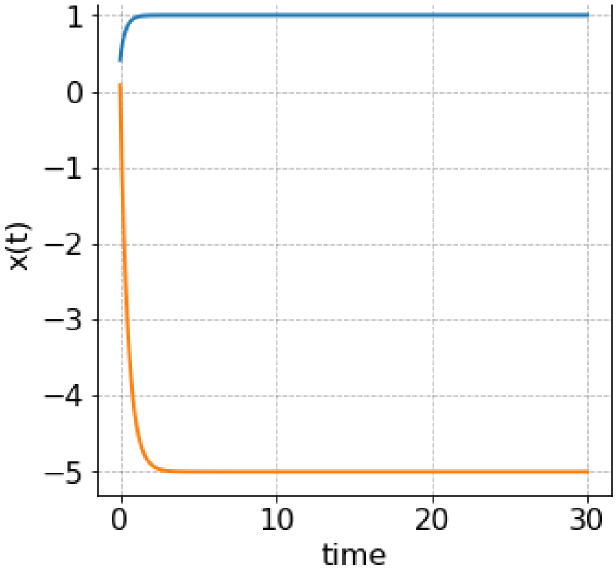
In [44]:

```
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles
import matplotlib.pyplot as plt

A = np.array([[10, 0], [-5, 10]])
B = np.array([[2, 1], [0, -1]])

#desired eigenvalues
poles = np.array([-3, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)
```

```
K: [[  4.   6.]
 [  5. -12.]]
eigenvalues of A - B*K: [-3. -2.]
```

In [46]:

```
x_desired = np.array([1, 0])
u_desired = np.array([-7.5, 5])
n=2

def StateSpace(x, t):
    u = -K.dot(x - x_desired) + u_desired
    return A.dot(x) + B.dot(u)

time = np.linspace(0, 30, 30000)
x0 = np.random.rand(n)   # initial state

solution = {"solution_1": odeint(StateSpace, x0, time)}

plt.rcParams['figure.figsize'] = [5, 5]
plt.rcParams["font.family"] = "DejaVu Sans"
plt.rcParams["font.size"] = 16
plt.rcParams["font.weight"] = 'normal'

# plt.subplot(221)
plt.plot(time, solution["solution_1"], linewidth=2)
plt.xlabel('time')
plt.ylabel('x(t)')
plt.grid(color='k', linestyle='--', linewidth=0.7, alpha=0.3)
# plt.title('autonomous')
```



Task 6 Discrete systems

Task 6.1 Find which of the following systems is stable:

- **a)**

$$x_{i+1} = \begin{pmatrix} 0.5 & 0.1 \\ -0.05 & 0.2 \end{pmatrix} x_i$$

In [47]:

```
from numpy.linalg import eig

A = np.array([[0.5, 0.1], [-0.05, 0.2]])

e, v = eig(A)
print("Eigenvalues of A:\n", abs(e))
```

```
Eigenvalues of A:
 [0.48228757 0.21771243]
```

Both $\lambda_1 = 0.48228757$
and $\lambda_2 = 0.21771243$
are less than 1, hence it is asymptotically stable

- **b)**

$$x_{i+1} = \begin{pmatrix} 1 & -2 \\ 0 & 0.3 \end{pmatrix} x_i$$

In [48]:

```
from numpy.linalg import eig

A = np.array([[1, -2], [0, 0.3]])

e, v = eig(A)
print("Eigenvalues of A:\n", abs(e))
```

```
Eigenvalues of A:
 [1.  0.3]
```

Both $\lambda_1 = 1$

and $\lambda_2 = 0.3$

are less than or equal to 1, hence it is Lyapunov stable

- **c)**

$$x_{i+1} = \begin{pmatrix} -5 & 0 \\ -0.1 & 1 \end{pmatrix} x_i + \begin{pmatrix} 0 \\ 0.5 \end{pmatrix} u_i, \quad u_i = \begin{pmatrix} 0 & 0.2 \end{pmatrix} x_i$$

In [51]:

```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[-5, 0], [-0.1, 1]])
B = np.array([[0], [0.5]])

K = np.array([[0, 0.2]])
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)
```

```
eigenvalues of A - B*K: [ 0.9 -5. ]
```

As absolute value of λ_2

is greater than 1, it is unstable

- **d)**

$$x_{i+1} = \begin{pmatrix} -2.2 & -3 \\ 0 & 0.5 \end{pmatrix} x_i + \begin{pmatrix} -1 \\ 1 \end{pmatrix} u_i, \quad u_i = 10$$

In [65]:

```
import numpy as np
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles

n = 2
A = np.array([[-2.2, -3], [0, 0.5]])
B = np.array([[-1], [1]])

K = np.array([[10]])
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)
```

```
eigenvalues of A - B*K: [ 1.34601913 -3.04601913]
```

As the absolute value of both of the λ_1

and λ_2

, is greater than 1, it is unstable

Task 6.2 Propose control that makes the following systems stable:

- **a)**

$$x_{i+1} = \begin{pmatrix} 1 & 1 \\ -0.4 & 0.1 \end{pmatrix} x_i + \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} u_i$$

For discrete system to be stable eigenvalues of the matrix has to be less than or equal to 1. \ Systems are written in form

$$x_{i+1} = Ax_i + Bu_i$$

\ Let us write u_i
as $-Kx_i$
and we get

$$x_{i+1} = (A - BK)x_i$$

\ Let us assume that eigenvalues of $(A - BK)$
are 0.5
and 0.25
, so we need to find appropriate values of matrix K
.

In [64]:

```
import numpy as np
from scipy.integrate import odeint
from scipy import signal

A = np.array([[1, 1],
              [-0.4, 0.1]])

B = np.array([[0.5],
             [0.5]])

C = np.array([[1, 1]])
D = np.array([[0]])

T = 0.1

A_d, B_d, C_d, D_d, _ = signal.cont2discrete((A,B,C,D), T)

e, v = eig(A_d)
print("Original eigenvalues of A:\n", e)

#desired eigenvalues
poles = np.array([0.5, 0.75])
place_obj = signal.place_poles(A_d, B_d, poles)

#found control gains
K = place_obj.gain_matrix
print("\nGain matrix K:\n", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A_d - B_d.dot(K)))
print("\nPlaced eigenvalues of A - B*K:\n", abs(e))
```

```
Original eigenvalues of A:
 [1.05549745+0.04693824j 1.05549745-0.04693824j]

Gain matrix K:
 [[19.09892426 -3.90787582]]

Placed eigenvalues of A - B*K:
 [0.5  0.75]
```

- **b)**

$$x_{i+1} = \begin{pmatrix} 0.8 & -0.3 \\ 0 & 0.15 \end{pmatrix} x_i + \begin{pmatrix} -1 \\ 1 \end{pmatrix} u_i$$

In [63]:

In [63]:

```
import numpy as np
from scipy.integrate import odeint
from scipy import signal

A = np.array([[0.8, -0.3],
              [0, 0.15]])

B = np.array([[-1],
             [1]])

C = np.array([[2, 1]])
D = np.array([[2]])

T = 0.1

A_d, B_d, C_d, D_d, _ = signal.cont2discrete((A,B,C,D), T)

e, v = eig(A_d)
print("Original eigenvalues of A:\n", e)

#desired eigenvalues
poles = np.array([0.5, 0.25])
place_obj = signal.place_poles(A_d, B_d, poles)

#found control gains
K = place_obj.gain_matrix
print("\nGain matrix K:\n", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A_d - B_d.dot(K)))
print("\nPlaced eigenvalues of A - B*K:\n", abs(e))
```

```
Original eigenvalues of A:
 [1.08328707 1.01511306]

Gain matrix K:
 [[-46.85547663 -35.7527522 ]]

Placed eigenvalues of A - B*K:
 [0.25 0.5 ]
```

Task 6.3 Design point-to-point control and simulate two discrete systems:

- **a) where $B \in \mathrm{R}^{2 \times 1}$**

$$x_{i+1} = \begin{pmatrix} 1 & -3 \\ 5 & -15 \end{pmatrix} x_i + \begin{pmatrix} -1 \\ -5 \end{pmatrix} u_i$$

Drive it towards the point $x^* = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

$\dot{x}^* = 0$

, so dynamics obtains the form:

$$0 = \begin{pmatrix} 1 & -3 \\ 5 & -15 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} -1 \\ -5 \end{pmatrix} u^*$$

In other words, $u^* = -2$

.

In [61]:

```
from numpy.linalg import eig
from scipy.integrate import odeint
```

```python
from scipy.signal import place_poles
import matplotlib.pyplot as plt

A = np.array([[1, -3], [5, -15]])
B = np.array([[-1], [-5]])

#desired eigenvalues
poles = np.array([-5, -2])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)
```

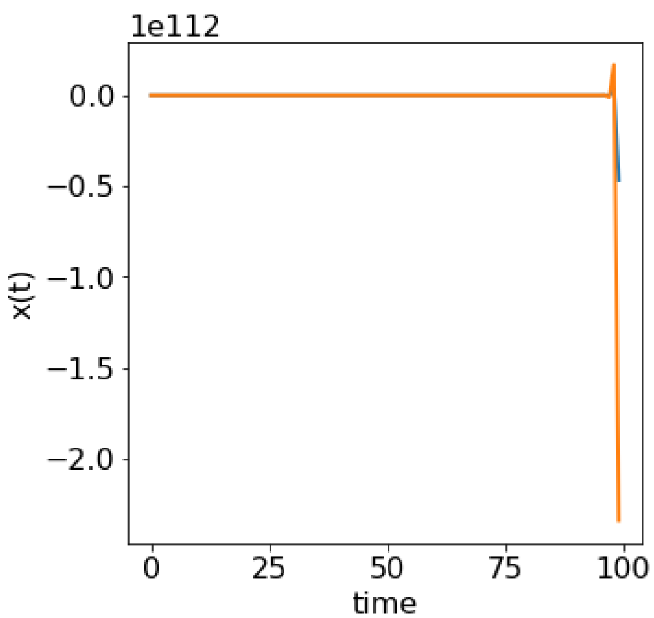
```
K: [[-3.53291421e+14  7.06582843e+13]]
eigenvalues of A - B*K: [-3.17256912+1880063.52215183j -3.17256912-1880063.52215183j]
```

In [62]:

```python
x_desired = np.array([1, 1])
u_desired = np.array([-2])
n=2

Count = 100
time = np.zeros((Count))
dt = 0.01

x0 = np.random.rand(n)  # initial state
solution = np.zeros((Count, 2))
solution[0, :] = x0

for i in range(0, Count-1):
    x = solution[i, :]
    x = A.dot(x)
    solution[i+1, :] = np.reshape(x, (1, 2))
    time[i] = dt*i

plt.plot(range(0, Count), solution, linewidth=2)
plt.xlabel('time')
plt.ylabel('x(t)')
```

Out[62]:

```
Text(0, 0.5, 'x(t)')
```



- **b) where** $B \in \mathrm{R}^{2 \times 2}$

$$\dot{x}_{i+1} = \begin{pmatrix} -1 & 4 \\ -3 & 12 \end{pmatrix} x_i + \begin{pmatrix} 1 & 2 \\ -1.5 & 4 \end{pmatrix} u_i$$

Drive it towards the point $x^* = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$

$\dot{x}^* = 0$

, so dynamics obtains the form:

$$0 = \begin{pmatrix} -1 & 4 \\ -3 & 12 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} 1 & 2 \\ -1.5 & 4 \end{pmatrix} u^*$$

In other words, $u^* = \begin{pmatrix} 2 \\ -4.5 \end{pmatrix}$

.

In [59]:

```
from numpy.linalg import eig
from scipy.integrate import odeint
from scipy.signal import place_poles
import matplotlib.pyplot as plt

A = np.array([[-1, 4], [-3, 12]])
B = np.array([[1, 2], [-1.5, 4]])

#desired eigenvalues
poles = np.array([-5, -3])
place_obj = place_poles(A, B, poles)

#found control gains
K = place_obj.gain_matrix;
print("K:", K)

#test that eigenvalues of the closed loop system are what they are supposed to be
e, v = eig((A - B.dot(K)))
print("eigenvalues of A - B*K:", e)
```

```
K: [[ 3.14285714 -2.        ]
 [ 0.42857143  3.        ]]
eigenvalues of A - B*K: [-3. -5.]
```

In [60]:

```
x_desired = np.array([1, 2])
u_desired = np.array([2, -4.5])
n=2

Count = 100
time = np.zeros((Count))
dt = 0.01

x0 = np.random.rand(n)   # initial state
solution = np.zeros((Count, 2))
solution[0, :] = x0

for i in range(0, Count-1):
    x = solution[i, :]
    x = A.dot(x)
    solution[i+1, :] = np.reshape(x, (1, 2))
    time[i] = dt*i

plt.plot(range(0, Count), solution, linewidth=2)
plt.xlabel('time')
plt.ylabel('x(t)')
```

Out[60]:

```
Text(0, 0.5, 'x(t)')
```