In [114…
```
import numpy as np
import matplotlib.pyplot as plt

# ----------------------------
# 1. Define arbitrary complex states |a> and |b>
# ----------------------------
# Example complex states
a = np.array([1/np.sqrt(2), 1j/np.sqrt(2)])   # |a⟩
b = np.array([np.sqrt(0.6), np.sqrt(0.4)*np.exp(1j*np.pi/4)])  # |b⟩
#theta=np.pi / 2
#a=np.array([[1], [0]])
#b=np.array([[np.cos(theta)], [np.sin(theta)]])
print("state 1=",a)
print("state 2=",b)
```

```
state 1= [0.70710678+0.j         0.        +0.70710678j]
state 2= [0.77459667+0.j        0.4472136 +0.4472136j]
```

In [115…
```
# Normalize states
a = a / np.linalg.norm(a)
b = b / np.linalg.norm(b)
print("Normalized state 1=",a)
print("Normalized state 2=",b)
```

```
Normalized state 1= [0.70710678+0.j         0.        +0.70710678j]
Normalized state 2= [0.77459667+0.j        0.4472136 +0.4472136j]
```

In [116…
```
# Inner product <a|b>
X = np.vdot(a, b)
print("inner product of a and b as X=<a|b>=",X)
```

```
inner product of a and b as X=<a|b>= (0.8639503235220042-0.316227766016838j)
```

In [117…
```
# Projection of |b> onto |a>
b_proj_on_a = X * a
```

In [118…
```
# ----------------------------
# 2. Skew Gram Matrix (2x2)
# ----------------------------
q0 = 0.5
q1 = 0.5
Gs = np.array([[-q0, -np.sqrt(q0*q1)*X],
               [np.sqrt(q0*q1)*X.conjugate(), q1]])
print("Skew Gram Matrix=\n",Gs)
# Eigenvalues and eigenvectors of SGM
eigvals, eigvecs = np.linalg.eig(Gs)
print("eigvals=",eigvals)
eigvecs_scaled = eigvecs * 0.5  # scale for plotting
```

```
Skew Gram Matrix=
 [[-0.5       +0.j         -0.43197516+0.15811388j]
 [ 0.43197516+0.15811388j  0.5       +0.j        ]]
eigvals= [-0.1959527-2.15332782e-17j  0.1959527+4.92888538e-17j]
```

In [119…
```
# ----------------------------
# 3. Probabilities & Error Rate
# ----------------------------
eta_abs = np.abs(eigvals)
P_e = 0.5 * (1 - np.sum(eta_abs))
P_correct_0 = q0 + eigvals[0]
P_correct_1 = q1 + eigvals[1]

print("probability of error=",P_e)
```

```
print("probability of correct  decision for 0=>  P(0)=",P_correct_0)
print("probability of correct  decision for 1=>  P(1)=",P_correct_1)
```

```
probability of error= 0.30404730259178336
probability of correct  decision for 0=>  P(0)= (0.30404730259178336-2.153327820804061
e-17j)
probability of correct  decision for 1=>  P(1)= (0.6959526974082166+4.928885382366952
e-17j)
```

In [120…

```
# ----------------------------
# 4. Plot everything in 2D (Re vs Im)
# ----------------------------
plt.figure(figsize=(8,6))
```

Out[120…

```
<Figure size 800x600 with 0 Axes>
```

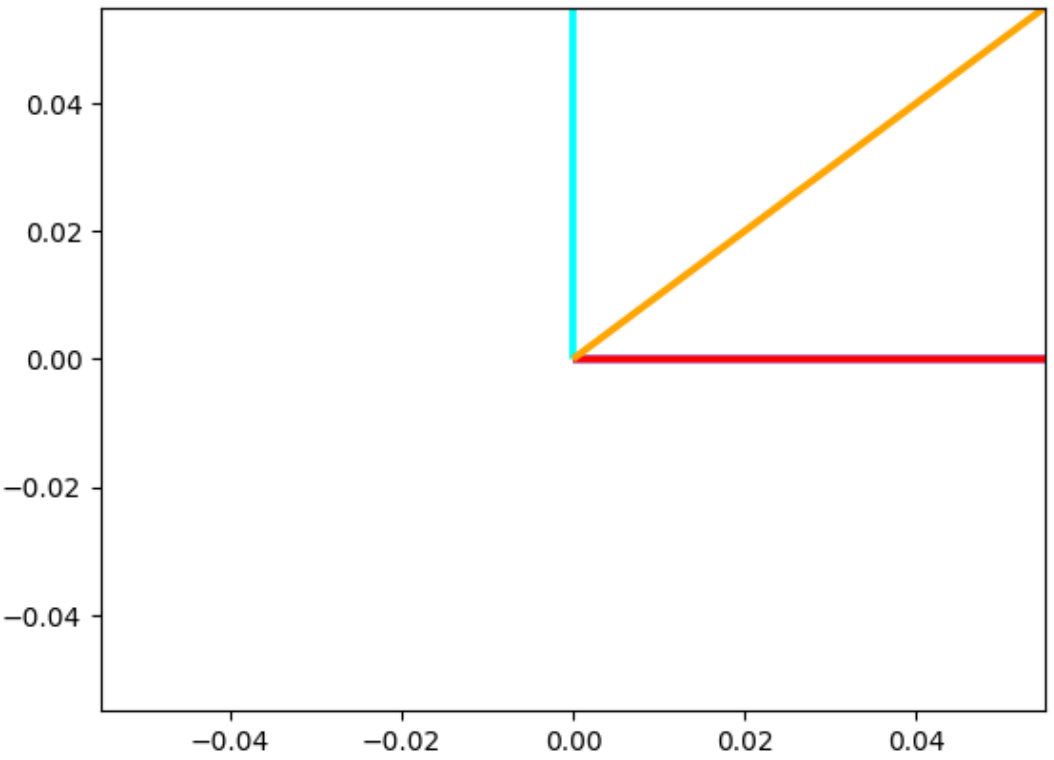
```
<Figure size 800x600 with 0 Axes>
```

In [143…

```
# Plot states (real vs imaginary parts)
plt.quiver(0,0,a[0].real,a[0].imag, angles='xy', scale_units='xy', scale=1, color='bl
plt.quiver(0,0,a[1].real,a[1].imag, angles='xy', scale_units='xy', scale=1, color='cy
plt.quiver(0,0,b[0].real,b[0].imag, angles='xy', scale_units='xy', scale=1, color='re
plt.quiver(0,0,b[1].real,b[1].imag, angles='xy', scale_units='xy', scale=1, color='or
```

Out[143…

```
<matplotlib.quiver.Quiver at 0x224ac5741a0>
```

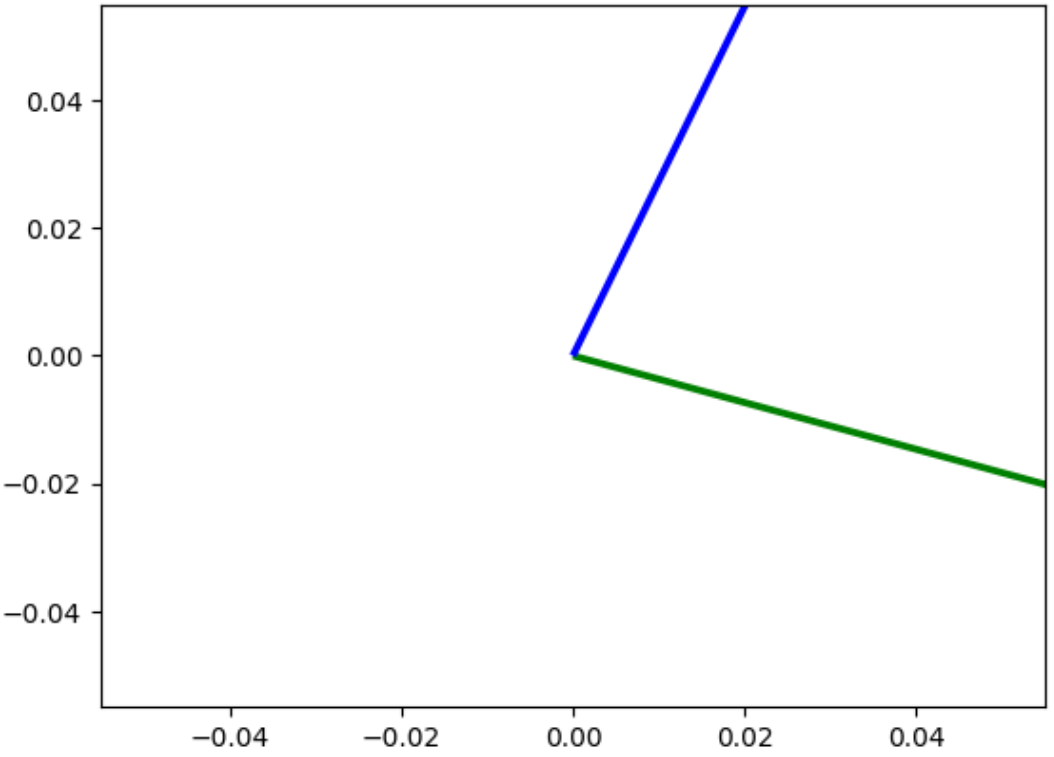


In [146…

```
# Projection of |b> onto |a> (real vs imaginary)
plt.quiver(0,0,b_proj_on_a[0].real,b_proj_on_a[0].imag, angles='xy', scale_units='xy'
plt.quiver(0,0,b_proj_on_a[1].real,b_proj_on_a[1].imag, angles='xy', scale_units='xy'
```

Out[146…

```
<matplotlib.quiver.Quiver at 0x224acbd7aa0>
```

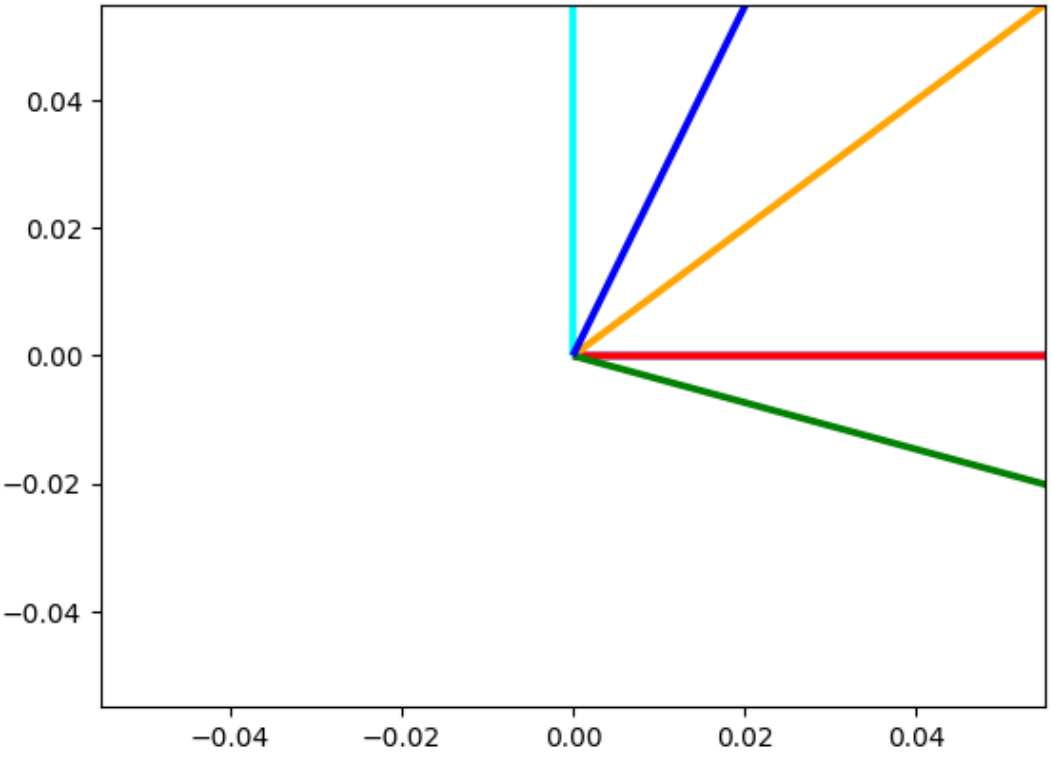


In [147…

```
# Plot states (real vs imaginary parts)
plt.quiver(0,0,a[0].real,a[0].imag, angles='xy', scale_units='xy', scale=1, color='bl
plt.quiver(0,0,a[1].real,a[1].imag, angles='xy', scale_units='xy', scale=1, color='cy
plt.quiver(0,0,b[0].real,b[0].imag, angles='xy', scale_units='xy', scale=1, color='re
plt.quiver(0,0,b[1].real,b[1].imag, angles='xy', scale_units='xy', scale=1, color='or

# Projection of |b> onto |a> (real vs imaginary)
plt.quiver(0,0,b_proj_on_a[0].real,b_proj_on_a[0].imag, angles='xy', scale_units='xy'
plt.quiver(0,0,b_proj_on_a[1].real,b_proj_on_a[1].imag, angles='xy', scale_units='xy'

# ------------------------------
# 4.
```

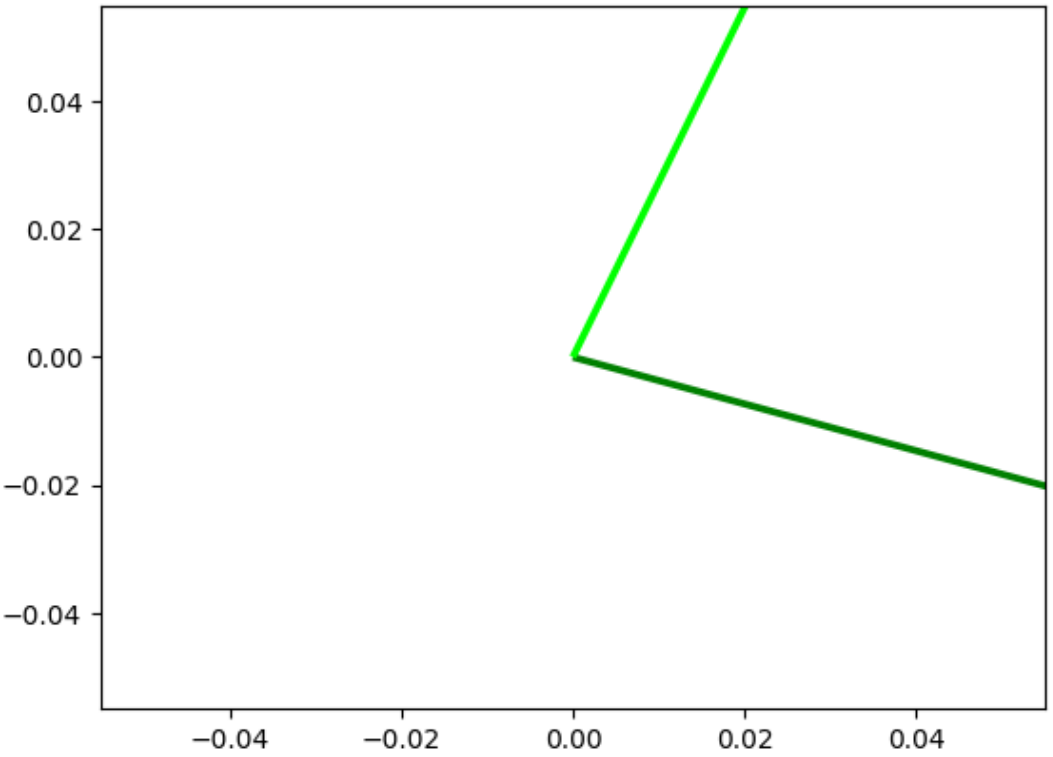
Out[147…

```
<matplotlib.quiver.Quiver at 0x224ae7aca70>
```



In [134…
```
# Projection of |b> onto |a> (real vs imaginary)
plt.quiver(0,0,b_proj_on_a[0].real,b_proj_on_a[0].imag, angles='xy', scale_units='xy'
plt.quiver(0,0,b_proj_on_a[1].real,b_proj_on_a[1].imag, angles='xy', scale_units='xy'
```

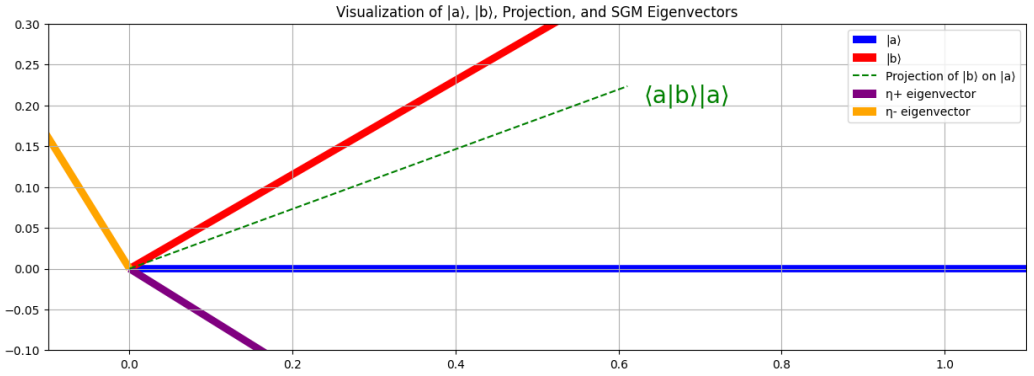
Out[134… <matplotlib.quiver.Quiver at 0x224ad1fd640>



In [133…
```
# ----------------------------
# 4. Plotting
# ----------------------------
plt.figure(figsize=(15,15))

# |a> and |b>
plt.quiver(0,0,a[0],a[1], angles='xy', scale_units='xy', scale=0.5, color='blue', lab
```

```
plt.quiver(0,0,b[0],b[1], angles='xy', scale_units='xy', scale=0.5, color='red', labe

# Projection as dashed line
plt.plot([0, b_proj_on_a[0]], [0, b_proj_on_a[1]], 'g--', label='Projection of |b⟩ on

# SGM eigenvectors
plt.quiver(0,0,eigvecs_scaled[0,0],eigvecs_scaled[1,0], angles='xy', scale_units='xy'
plt.quiver(0,0,eigvecs_scaled[0,1],eigvecs_scaled[1,1], angles='xy', scale_units='xy'

# Labels
plt.text(a[0]+0.45,a[1]+0.01,'|a⟩', color='blue',fontsize=20,fontweight='bold')
plt.text(b[0]+0.02,b[1]+0.01,'|b⟩', color='red',fontsize=20,fontweight='bold')
plt.text(b_proj_on_a[0]+0.02,b_proj_on_a[1]-0.02,'⟨a|b⟩|a⟩', color='green',fontsize=20)
plt.text(eigvecs_scaled[0,0]+0.02,eigvecs_scaled[1,0]+0.02,'η+', color='purple',fonts
plt.text(eigvecs_scaled[0,1]+0.02,eigvecs_scaled[1,1]+0.02,'η-', color='orange',fonts

plt.xlim(-0.1,1.1)
plt.ylim(-0.1,0.3)
plt.gca().set_aspect('equal')
plt.grid(True)
plt.title('Visualization of |a⟩, |b⟩, Projection, and SGM Eigenvectors')
plt.legend()
plt.show()
```

η- |b⟩



|a⟩

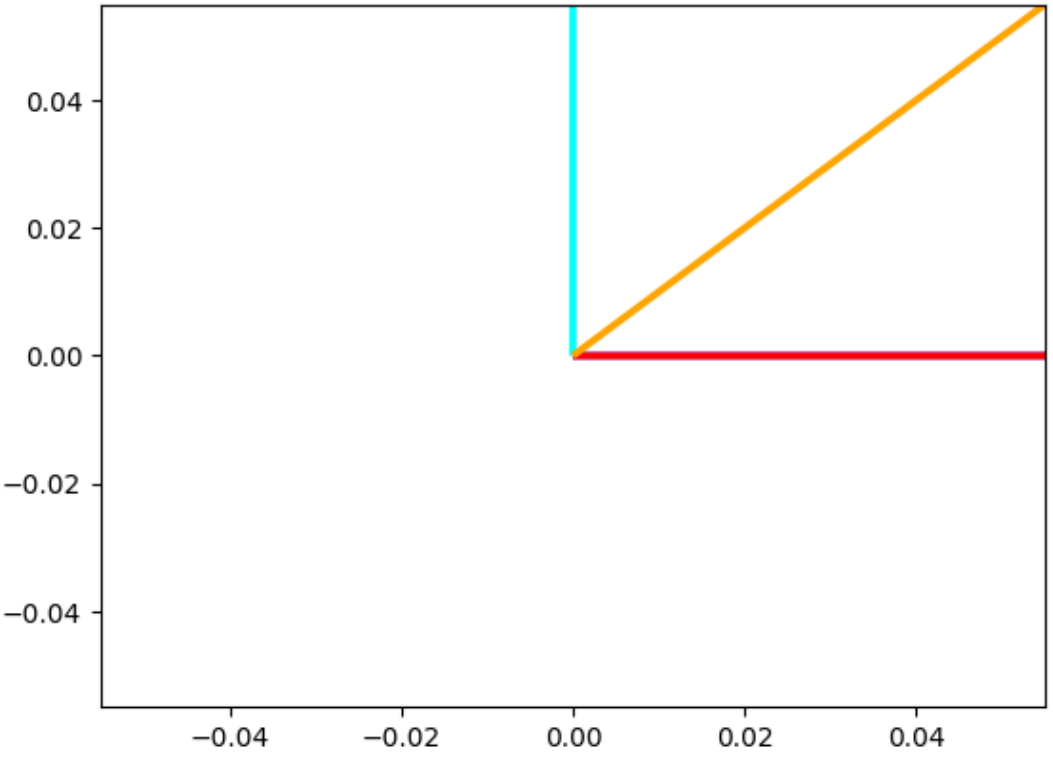
η+

In [141…

```
# Plot states (real vs imaginary parts)
plt.quiver(0,0,a[0].real,a[0].imag, angles='xy', scale_units='xy', scale=1, color='bl
plt.quiver(0,0,a[1].real,a[1].imag, angles='xy', scale_units='xy', scale=1, color='cy
plt.quiver(0,0,b[0].real,b[0].imag, angles='xy', scale_units='xy', scale=1, color='re
plt.quiver(0,0,b[1].real,b[1].imag, angles='xy', scale_units='xy', scale=1, color='or
```

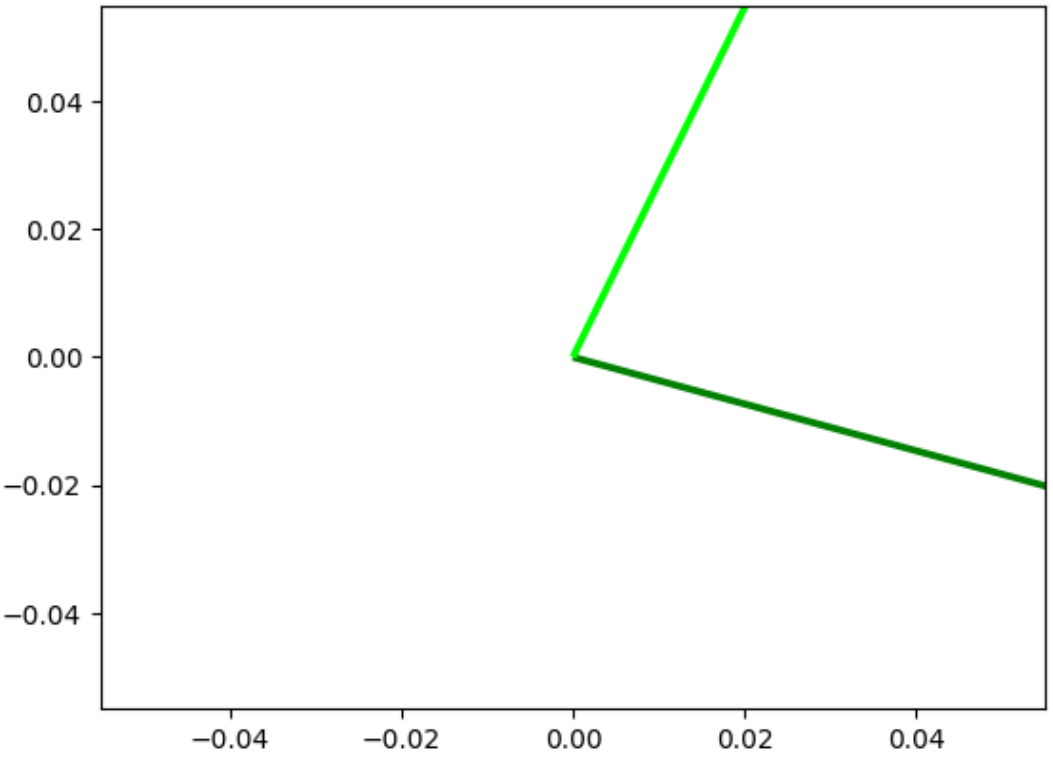
Out[141…

```
<matplotlib.quiver.Quiver at 0x224ac5465a0>
```



In [140…
```
# Projection of |b> onto |a> (real vs imaginary)
plt.quiver(0,0,b_proj_on_a[0].real,b_proj_on_a[0].imag, angles='xy', scale_units='xy'
plt.quiver(0,0,b_proj_on_a[1].real,b_proj_on_a[1].imag, angles='xy', scale_units='xy'
```

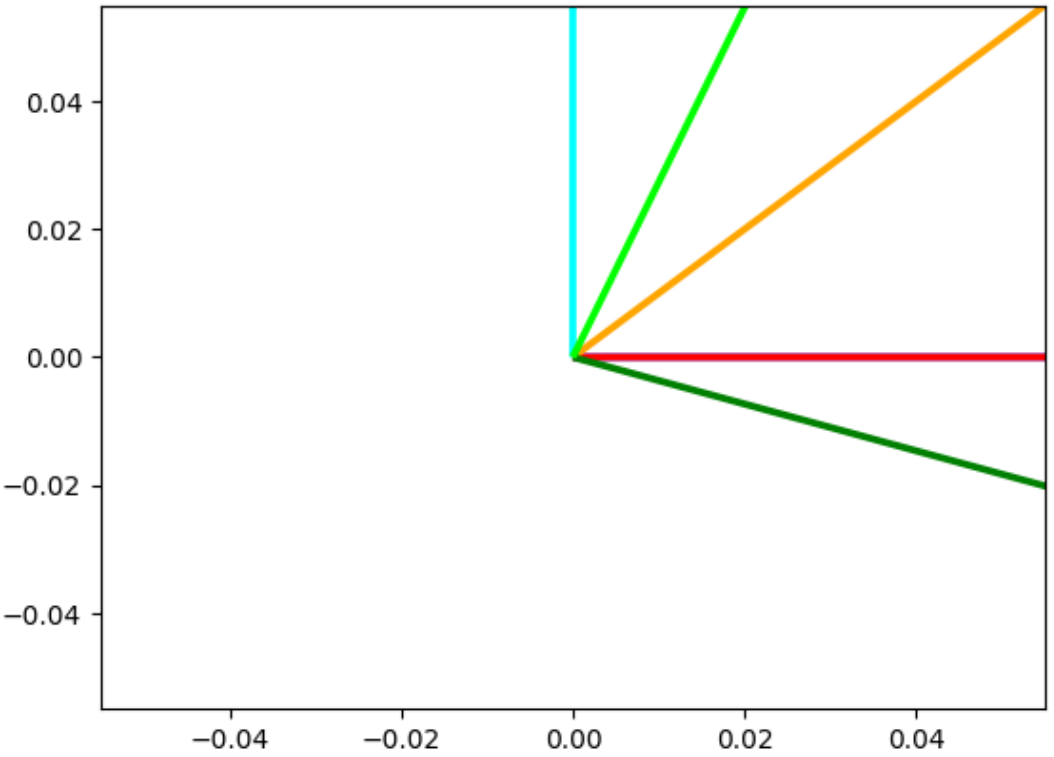
Out[140…
```
<matplotlib.quiver.Quiver at 0x224acae1970>
```



In [139…
```
# Plot states (real vs imaginary parts)
plt.quiver(0,0,a[0].real,a[0].imag, angles='xy', scale_units='xy', scale=1, color='bl
plt.quiver(0,0,a[1].real,a[1].imag, angles='xy', scale_units='xy', scale=1, color='cy
plt.quiver(0,0,b[0].real,b[0].imag, angles='xy', scale_units='xy', scale=1, color='re
plt.quiver(0,0,b[1].real,b[1].imag, angles='xy', scale_units='xy', scale=1, color='or

# Projection of |b> onto |a> (real vs imaginary)
```

```
plt.quiver(0,0,b_proj_on_a[0].real,b_proj_on_a[0].imag, angles='xy', scale_units='xy'
plt.quiver(0,0,b_proj_on_a[1].real,b_proj_on_a[1].imag, angles='xy', scale_units='xy'
```

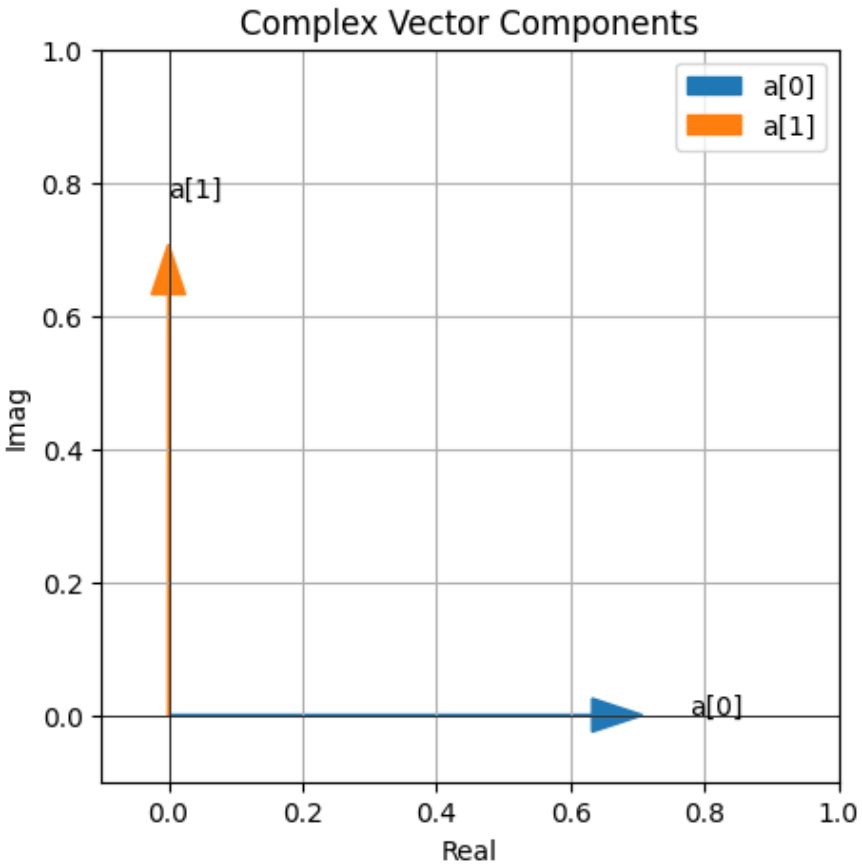
Out[139… `<matplotlib.quiver.Quiver at 0x224ac4b9970>`



In [ ]:

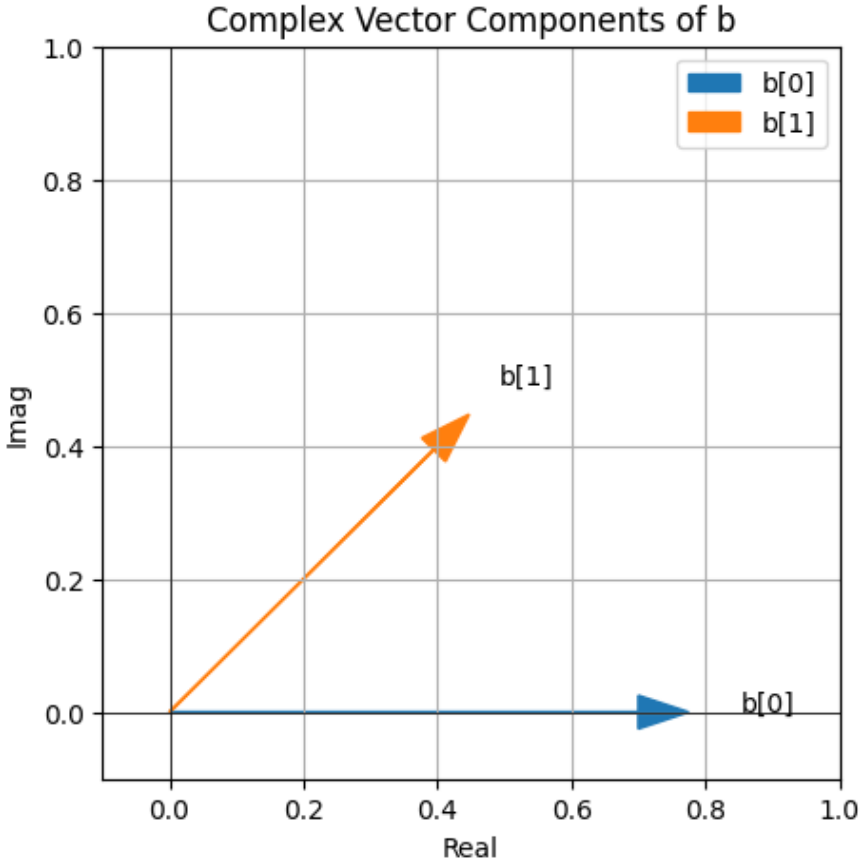
In [124…

```
import matplotlib.pyplot as plt
import numpy as np

# Define the vector
a = np.array([1/np.sqrt(2), 1j/np.sqrt(2)])

# Plot the complex components
fig, ax = plt.subplots(figsize=(5,5))
for i, val in enumerate(a):
    ax.arrow(0, 0, val.real, val.imag, head_width=0.05, length_includes_head=True, co
    ax.text(val.real*1.1, val.imag*1.1, f"a[{i}]")

# Axes and grid
ax.set_xlim(-0.1, 1)
ax.set_ylim(-0.1, 1)
ax.axhline(0, color='black', linewidth=0.5)
ax.axvline(0, color='black', linewidth=0.5)
ax.set_xlabel('Real')
ax.set_ylabel('Imag')
ax.set_title('Complex Vector Components')
ax.grid(True)
ax.legend()
plt.show()
```



In [125…

```
import matplotlib.pyplot as plt
import numpy as np

# Define the vector
b = np.array([np.sqrt(0.6), np.sqrt(0.4) * np.exp(1j * np.pi / 4)])

# Plot the complex components
fig, ax = plt.subplots(figsize=(5,5))
for i, val in enumerate(b):
    ax.arrow(0, 0, val.real, val.imag, head_width=0.05, length_includes_head=True, co
    ax.text(val.real*1.1, val.imag*1.1, f"b[{i}]")

# Axes and grid
ax.set_xlim(-0.1, 1)
ax.set_ylim(-0.1, 1)
ax.axhline(0, color='black', linewidth=0.5)
ax.axvline(0, color='black', linewidth=0.5)
ax.set_xlabel('Real')
ax.set_ylabel('Imag')
ax.set_title('Complex Vector Components of b')
ax.grid(True)
ax.legend()
plt.show()
```



In [ ]:
```
a = np.array([2+2j, 1-1j])
print(a)
print(np.linalg.norm(a))  # ≈ 3.162
a_normalized = a / np.linalg.norm(a)
print(np.linalg.norm(a_normalized))  # = 1
```

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

# Example vectors (already normalized if needed)
a = np.array([1/np.sqrt(2), 1j/np.sqrt(2)])
b = np.array([np.sqrt(0.6), np.sqrt(0.4) * np.exp(1j * np.pi / 4)])

# Normalize vectors
a = a / np.linalg.norm(a)
b = b / np.linalg.norm(b)

# Compute projection of b onto a
b_proj_on_a = np.vdot(a, b) * a  # a† b * a

# Plot the vectors
fig, ax = plt.subplots(figsize=(6,6))

# Original vectors
ax.arrow(0, 0, a[0].real, a[0].imag, head_width=0.05, color='C0', label='a[0]')
ax.arrow(0, 0, a[1].real, a[1].imag, head_width=0.05, color='C1', label='a[1]')
ax.arrow(0, 0, b[0].real, b[0].imag, head_width=0.05, color='C2', label='b[0]')
ax.arrow(0, 0, b[1].real, b[1].imag, head_width=0.05, color='C3', label='b[1]')

# Projection vector (same dimension as a)
ax.arrow(0, 0, b_proj_on_a[0].real, b_proj_on_a[0].imag, head_width=0.05, color='r',
ax.arrow(0, 0, b_proj_on_a[1].real, b_proj_on_a[1].imag, head_width=0.05, color='r',

# Axes
```

```
ax.axhline(0, color='black', linewidth=0.5)
ax.axvline(0, color='black', linewidth=0.5)
ax.set_xlabel('Real')
ax.set_ylabel('Imag')
ax.set_title('Projection of b onto a in Complex Plane')
ax.set_xlim(-0.2, 1)
ax.set_ylim(-0.2, 1)
ax.grid(True)
ax.legend()
plt.show()
```

In [ ]:

```
import numpy as np

# Example normalized vectors
a = np.array([1/np.sqrt(2), 1j/np.sqrt(2)])
b = np.array([np.sqrt(0.6), np.sqrt(0.4) * np.exp(1j * np.pi / 4)])

# Normalize just to be safe
a = a / np.linalg.norm(a)
b = b / np.linalg.norm(b)

# Inner product (overlap)
X = np.vdot(a, b)  # a† b

# Example probabilities/weights
q0 = 0.5
q1 = 0.5

# Construct Gs
Gs = np.array([[-q0, -np.sqrt(q0*q1)*X],
               [np.sqrt(q0*q1)*X.conjugate(), q1]])

print("G_s =\n", Gs)

# Check Hermitian property
is_hermitian = np.allclose(Gs, Gs.conj().T)
print("\nIs G_s Hermitian?", is_hermitian)

# Eigenvalues
eigvals = np.linalg.eigvals(Gs)
print("\nEigenvalues of G_s:", eigvals)
```

In [ ]: